What is a File?

Richard Harper[1], Eno Thereska[1], Siân Lindley[1], Richard Banks[1], Phil Gosset[1], William Odom[2], Gavin Smyth[1], Eryn Whitworth[3]

[1]Microsoft Research Cambridge
7 JJ Thomson Avenue
Cambridge, CB3 0FB, UK
Email: intialsur-name@microsoft.com
Except r.harper@

[2]Carnegie Mellon University
Human-Computer Interaction Institute
Pittsburgh, PA, USA
Email wo@willodom.com

[3]The University of Texas at Austin School of Information
1616 Guadalupe
Austin, TX 78701-1213
Email eryn@mail.ischoolutexas.edu

ABSTRACT

For over 40 years the notion of the *file*, as devised by pioneers in the field of computing, has been the subject of much contention. Some have wanted to abandon the term altogether on the grounds that metaphors about files can confuse users and designers alike. More recently, the emergence of the 'cloud' has led some to suggest that the term is simply obsolescent. In this paper we want to suggest that, despite all these conceptual debates and changes in technology, the term file still remains central to systems architectures and to the concerns of users. Notwithstanding profound changes in what users do and technologies afford, we suggest that files continue to act as a cohering concept, something like a 'boundary object' between computer engineers and users. However, the effectiveness of this boundary object is now waning. There are increasing signs of slippage and muddle. Instead of throwing away the notion altogether, we propose that the definition of and use of files as a boundary object be reconstituted. New abstractions are needed, ones 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.

Author Keywords

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

ACM Classification Keywords

H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

CSCW '13, February 23–27, 2013, San Antonio, Texas, USA.

INTRODUCTION

As long ago as 1981, Frank Halasz and Tom Moran argued that the term file was harmful to good HCI [5]. In their view, users have some kind of mental model or 'internal picture' of files, and this is necessarily different from the 'abstract conceptual model' that is embedded in a computer. When the term is used to conjure 'literary metaphors' [p. 385] in, say, the design of interfaces, muddles inevitably result. For the model that ends up being presented to the user will never 'fit' the actual system – it can never do so, in their view. The two are fundamentally irreconcilable; the metaphorical model and the 'real' model embedded in the system.

This view certainly seems cogent and doubtlessly is one that many in HCI would accept. After all it turns around a basic premise in much of the psychology used in HCI. This holds that the external world needs to be represented in some kind of internal form or 'qualia' in the human mind and good design ensures that there is a fit between this internal model and the external. Since there is unlikely to ever be a 'good internal model' of computer files, then Halasz and Moran's case is that it would be unwise to try and design towards that model; it is best to design on the basis of the 'real thing' even though that prohibits certain types of metaphor – like those associated with the term file.

But today, as new applications and technologies confront the user, the apparent cogency of Halasz and Moran's case seems to be weakening. For it turns out that it is not clear what the entities that users deal with might be from anyone's point of view – the users *or* the engineers. What a file might be is becoming muddled, lost perhaps; certainly confused.

For example, one of Microsoft's products is *OneNote*. To a user, a OneNote *Notebook* looks very much like the thing produced in word processing applications – a document of sorts; one might even suggest a file-like thing. One of the appeals of OneNote is that it is somewhat more flexible than typical word processing applications, such as Microsoft's own *Word*. Users can easily add images and pictures,

cut and paste from the Web, even add their own scribbles to a Notebook which the system can convert through OCR to typescript. All this can be brought together as a single thing. The application treats the thing that a user thus creates in a particular way. A Notebook is not a single entity; it is not, as it were, a single file. It is a collection of files – and indeed this is how the engineers who designed and built One-Note think of it: as a thing that consists of a number of 'read-write objects'. The important point to understand, however, is that, in OneNote, it is 'Sections' that define what a 'file' is. A user creates Sections within OneNote (although they may also interact with a Page, which they may think of as components of a Section).

Thus far, so good; in this scenario what a file is from the user's and the system engineer's point of view is pretty close; there is some correspondence between the 'mental model' and the 'real model'. However, when a user tries to save a Notebook on to a cloud service, like *SkyDrive* (as it happens also a Microsoft product), something else comes into play: namely, a different understanding of what are the file entities that makes a Notebook. In SkyDrive – to put it in simple terms – instead of a Section being a file (irrespective of how many pages it consists of), a file is sometimes redefined by the cloud storage application itself, one result of this being that on occasion a Section will be stored as several files. The cloud system stores its data in terms of the file abstractions, as bundles of files, but these don't correspond to the originating application's way of doing so: in one, 'files' take a different form to another.

That the cloud system defines files in its own way would not matter to the user as long as conversion back again is consistent. However, it does become an issue if a user's attempt to save to SkyDrive is interrupted (as often happens when accessing cloud services). For when they go back to SkyDrive after the interruption, they might find a version of their Notebook already there. This version will, though, be partial; incomplete in some respects and reflecting not their own understanding of the structure of their 'thing' nor the view of the file structure that developers of OneNote had in mind but something else: what the engineers behind SkyDrive conceived of as the file entities constitutive of a Notebook. It is this vision that will have determined which entities (files) were copied over before the interruption began.

At first glance, this would seem to confirm Halasz and Moran's fear: don't expect the user's mental model to fit systems. But this is not all that is going on here. What this scenario illustrates is that computer scientists don't have a common understanding of files *either*. This issue, this divergence of basic concepts, is not so much new, as it is one that comes to be highlighted with the massively networked, distributed world we are coming to where differences in basic concepts in applications start clashing when brought together.

This is not the only change that needs to be borne in mind, however. If the examples above evoke the PC and how it works, there are now other devices that are becoming almost as common and these have no files at all. The iPad, for example, has no equivalent to the Windows File Explorer for the simple reason that it has no files to view. It is fileless. Or at least this is what some would have us believe. For underneath the hood of this [see 7] and similar devices – like the more advanced smart phones – there are things that get called files although this is a label for 'abstractions' that allow the systems in question to run efficiently – address book information can be re-used in various applications, for example, as can user identity information. Unfortunately, it turns out just what these abstractions are and how they might be used is far from fixed or agreed. Attempts to ensure that new application developers avoid some of the difficulties that these ambiguities produce through, for example, sandboxing their applications (and hence hiding any abstractions from the developers' grasp), cause considerable complaint and besides, the boundaries of the sandpits are routinely broken by developers who find ways to illicitly access file stores via the OS.

A Way Forward

So what is one to make of this all? We don't think that the way forward in the current context is to abandon the term files altogether. We think that the term – and related ones such as 'to file' and 'filing'– can be linked in a way that can allow computer scientists and users to orient to a shared object or set of objects, even though the tasks they have in mind are in many respects quite distinct. In this regard, we think that the term can label what Star and Greisemer called in 1989 a 'boundary object' [18, see also Star's revised and more nuanced view, 17]: a device of sorts that can, amongst other things, allow two or more distinct communities to interact around a mutually comprehended and agreed 'object' and which can, at the same time, bring them together through focused arrangements and processes of action. In this view, files are not merely the neutral ground between users and engineers but a label for a space of organised collaboration.

Our argument will not be that the term file is already a boundary object of sorts however (it might well have been – but historical questions are not our main interest), so much as that it can become an *effective* one if it is given renewed meaning and vitality. A new set of definitions for the term file is required: ones that will allow computer scientists to engineer what users require, that will provide a meaningful base for those users to act upon and which can enable an evolving interaction of design and practice to emerge.

More than this, however: our goal is not only to propose how a refined definition of the term, a renewed boundary object, can enable these two sides to collaborate. It can also be, in our view (and consequent on this), part of a larger effort to define a new *grammar of action* for an HCI of the 21st century. This will be of concern to users and engineers.

It seems to us that the practices of users are now profoundly more social in nature than they were when the term 'file' was first coined by computer scientists and the first digital 'file' presented to users. Today, users don't just want to create and store files. They also want to share those files with friends and buddies on social networks. Sometimes they want to give those files to their friends, and sometimes they want to keep ownership or possession of them, even while their buddies view them. And all of this might happen in ways that make the question of where a file is stored sometimes relevant and sometimes not. At the same time, the tasks that computer scientists have to attend to given this evolution in user practice, the diversity of devices that engineers need to ensure run efficiently as the bedrock for these new practices, and the need to enable interaction between applications both between devices and on occasion processed remotely on the web, requires that from their view, a revived and invigorated notion of files needs to be deployed that can allow them to engineer in effective and hopefully novel ways.

Overview of the Paper

Our goals are then quite bold, albeit our topic might seem prosaic, almost obsolescent if some commentators are to be believed. Given this, we will structure the paper as follows. First, we want to remark on something that might seem orthogonal to our main goal, though as the paper unfolds we hope that this will be seen to be not the case.

We commence with the question of how words are to be understood. If one were to believe Halasz and Moran, words label things; thus the thing labelled by the word file in the outside world is (or ought to be) mapped into a representation of that thing (i.e., a qualia) in the mental world. But we think words are best understood as performative, as ways of doing things as well as sometimes labels for things. If this is so, we argue that when we approach the problem of what files 'are' (or might be – we shall assume it for the purposes of argument) we need to be sensitive to how the word itself is used – and as we shall see, that use is quite diverse. It is important we do not muddle up uses as we try to plan a way forward for future use; rather we need to see what are the kinds of action that maps across and binds some uses while distinguishing others. These actions can be the basis of the grammar of action we will want to propose later on.

Having set the conceptual scene, we then turn to the more substantive topic of what files do (or did) in early computer system design. We remark on how the role of files has evolved and altered over the years. Though only a sketch, we will look at early attempts to move away from file systems to, for instance, database systems. The latter can be seen as emphasising relationships among files, rather than the file and its location within a folder or file hierarchy, in that the unit of most interest was the type of relationship itself. We then remark, again in passing, on attempts to design interfaces (primarily for the desktop) that sought to move away from the file metaphor or which sought to reinvigorate it, claiming that it has some cognitive value, Halasz and Moran notwithstanding. We will note that many of these papers did not attend to the question (or the role) of files from the engineer's point of view and so don't effectively comment on files as a boundary object. Nevertheless, this research does point towards new user practices.

We shall then turn to the technologies of the current time: to 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. Building on examples such as provided above with regard to OneNote, we further illustrate some of the many difficulties that the old file abstraction has and is creating in this new world. We argue that these difficulties attest not to an 'incorrect model' of a file, so much as to the stresses that emerge when engineers and users try to orientate to what they think is the 'same model', when so many changes and dynamics are afoot that the model (or models) in question are made muddled, hybrid not to say contradictory. Besides, we go on to argue that the technology is changing, what applications allow users to do is changing, what users themselves want to do is also altering and this is begging the question of not so much what files are as what they ought to be or could be. The role of a file as a boundary object in the current context is thus losing its efficacy, we suggest, but if reinvigorated could lead us to inquire into new possibilities. It needs to be made more than simply a term used in common (or simply used by one side) obviously; it needs to be a label for some thing and for some practices that bind and enable, that direct and constrain, and that facilitate coherent innovation.

We then outline what we think might be a way forward, a path that allows 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 data-as-files might be stored, and how it (they) might be accessed, copied, moved, made secure and so on, are thus dealt with effectively. But here we don't offer technological descriptions, say, as illustrative of where a new file abstraction might come to play a role, or recommendations for the design of a data store that combines file entities with other data types, such as graph relations (something that is required in many web-based application experiences [see 19]). Rather, we offer what we think ought to be the foundation of all such considerations: more nuanced vocabulary about what a system needs to enable and allow. This can be achieved, we believe, only in part by redefining what is meant by the word file and the doings associated with that usage; what is also required, and this is the rub of our paper, is a new grammar of action: a grammar for both users and engineers that provides the common ground of cohering their plans, their doings, their goals, around files and the other things they want to do in the age of the cloud.

WORDS AND MEANING: FILES AS A PERFORMATIVE CATEGORY

When one reads papers like Halasz and Moran's 'Analogy Considered Harmful' one is led to thinking of language as being a kind of tool, one that when used properly labels things. From this point of view, one of the troubles that HCI has to contend with is that users don't always know how to deploy language terms properly; there are infelicitous in their use and their understanding. Experts, meanwhile, technicians, computer scientists, HCI professionals and so forth, don't suffer from this egregious practice: their training ensures they use words correctly.

This is not the only way of approaching the general relationship between the use of terms in everyday life and in specialist or expert contexts, however. Other approaches treat user understandings as distinct from the technical but not as competitors to one another, with the user model often seeming weak or fallible in the way evoked by Halasz and Moran. Here, philosophers like Hanfling [6], following Wittgenstein [22], suggest that everyday language terms are not, as is often claimed by some (most notoriously the Churchlands [2]), part of 'common sense' attempts to do science – bad science to boot. Everyday terms do have a use, a purpose if you like; but in this view scientific usages have another.

The moral from this is that one ought to allow and assume this diversity. Nor should one rush to judge the former by their applicability to the latter. Terms are to be assessed by what they are used to do in the places in which those terms do 'work'. The term file and its various uses in everyday and technical contexts illustrates this kind of diversity and workload; it highlights too the broad distinctions between the everyday and the technical action with language.

In ordinary life, words like 'file' are often used to label things. In this respect there is an empirical referent at issue, one that subdivides the world into things that can be filed and those that cannot. But the ways in which the term is used are more nuanced than this. Thus, a letter from the Bank might be called a file but a love letter not so. The term implies, that is to say, something about its contents. Further, the term is also deployed to highlight particular relationships, between a 'file' and a person, for example. When this happens what is being pointed towards is accountability; the onus that someone might have to look after something is being emphasised. These usages (and doubtless others), which are commonplace in everyday life, are glossed over by the language of empirical reference.

In computer science, meanwhile, the term file is used to label the minimal digital entity that can be 'persisted'. A file is a label for a bundle of data, that is to say, but what that bundle consists in (i.e., what the binary data represent) doesn't really matter nor is it implied in the use of the term. A love letter and correspondence from the Bank are the same. And while the computer scientist does worry about accountability, in their case, they are concerned not with how an individual needs to look after a file but how a system does. The engineer seeks to answer questions like 'where does the file system put a bundle?', 'how does it name it?', 'retrieve' and 'make that file secure?' As it happens, some of these bundles can themselves be bundled into larger entities, and these start to look like the things that users 'see' when they interact with digital objects like Microsoft Word 'files'. Hence the term files is often labelled an *abstraction* in computer science, both because it labels a category that is a step higher than digital bytes, and because it can encompass various types of associations, bundles that look like 'user files' from above, from the users' point of view.

The point of these remarks is to allow us to assert the claim that, if and when language is understood as being made up of terms that are used in diverse and rich ways (as is the case with the term files), then one should not necessarily reject attempts to make linkages between instances of language use just because the empirical referent in question is diverse or complex, or because one use is empirical and another is not. Rather, one needs to approach the possibility of connection between uses of words carefully. Perhaps linkages of some kind are useful, perhaps they are not; it depends on what similarity (or otherwise) in actions are implied in each case.

Files as a *Way* of Bundling

This is illustrated when discussions about the cloud beg questions about what is the stuff that users want backed up and how it might be that computer scientists can engineer systems that are secure, economic and pliable with regard to this stuff. In these discussions one will find that computer scientists (if one can treat them as a single group for the moment) use the word file to label things that need to be dealt with quite carefully, and the doings that they are thinking of when they use the word are themselves quite particular and specialised. We have begun to remark on what those details are. Let us push that further. We have seen that computer scientists need to bundle the stuff they store. They need to put it somewhere and know where it is – though they care little for what it is. But at the same time, they do care that, whatever it is, it 'survives'. And this is an issue for computer scientists – not because they are neglectful in their design. Far from it. The problem is that the stuff they deal with, the stuff that comes in the form of bits and bytes, exists in an environment where that stuff (leaving aside what it consists of) can disappear.

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, computer scientists engineer computer systems (all systems, not only the flaky PC but also cloud storage systems and servers) to store data in such a fashion (or arrangement) that if part of

the data is destroyed or even the whole, this event does not catastrophically affect the system's functioning, its utility. For the architecture duplicates data, so that if some is lost (as it is assumed will happen), then a copy will be available (somewhere or other) and can thus be used as a replacement. This in turn means that there has to be a system element to manage these duplications and stores: this keeps these versions up to date after changes are made, for example, and knows where they are stored. 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, duplicated, preserved and so on, by the computer system and its component applications.

Some of the actions done with this abstraction, these things called files and the systems of which they are a part, are mirrored in the things that users can see (and do) and some are not. 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 at 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).

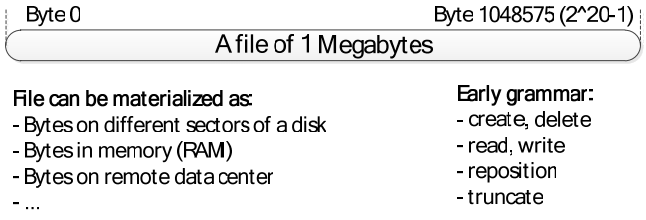

Figure 1: operating system view of a file abstraction as a consecutive, uninterrupted range of bytes [14, pp. 372-372]. The abstraction relieves users from understanding how the bytes that make up a file are scattered across memory or disk, giving them a simple logical object to operate on.

It is important to bear in mind that all of this work, all these doings, are not necessarily seen or understood by users – though these doings allow users to get their doings done whatever they might be. Questions of where bytes are stored are systematically hidden from the user's view.

Files as Boundary Objects

The point is that the term abstraction does label how computer scientists view the issues at hand, the issue of what a file is for both them and the user. 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, while the computer scientist has another. It is in fact more complicated than that, however; it is not two abstractions. They are distinct yet bound in various ways. One element that coheres both views is that the thing itself – the OneNote file, say – is *at the point of use* for both the user and the computer scientist effectively a unitary object, a single entity, not a composite. So although a file may be made up of bits brought together from other files – and again, those other files might not be singular – those bits are brought together in real time when the user 'engages' with them. Never mind what a file system does with the entities constitutive of this file; the point is that *at the moment 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 the same thing, though how this gets assembled, what 'it' (i.e., the file) is going to become and where it came from, is different. The talent 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. Things get read and saved, users can act, the system can function.

This description is obviously a simplification of file systems design just as the one above was of the orientations of everyday life. But it is sufficient, we hope, to convey the claim that one might think of the word file – or rather the concept 'file' – as something like a boundary object [18; see also 17]. Without wishing to make any historical claims at this point, this object would appear to stand in the middle ground between the user and the computer scientist in a complex and delicate way, foreshadowing certain moments of cohesion while allowing differentiated paths of activity. 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.

Previous Attempts to Reconstitute Files

We have suggested that in being an abstraction, a boundary object, there is a profoundly complex yet binding differentiation 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 alter the form of this differentiation, and crudely speaking to bring these different orientations to files, 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 three major and related themes of inquiry on this topic: the first concerning how to make documents on computers 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 distributed systems where a file can be an 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 [4], 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 and Murphy's 'Hierarchical File Systems are Dead' [13], 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 [21]. 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.

Though there was much hyperbole at the time, for a variety of reasons this way of treating files did not get released. However, various efforts that resonate with the aspirations behind WinFS have since appeared, at least within the research community. Jones *et al.'s* 'Don't take my Folders away' of 2005 [8] explored some of the cognitive muddles that went with folders and file hierarchies, for example, and pointed towards ways that current file directories prohibit multiple locations of file instances, though users themselves would prefer that such hybridity was allowed. Such possibilities needed to be met without negating the importance of place in the mental furniture of the mind. Jones *et al.* did not make any suggestion as to how to engineer such a system, however. Cutrell *et al.* [3] proposed a system that combined web-functionality with the PC with their *Phlat* system, allowing rich searching and browsing through augmenting PC files with the kinds of metadata hitherto associated with websites. Meanwhile, Voida *et al.* [20] sought to devise a desktop interface that reframed the entities engaged with (such as a Word document or an email) around semantically defined activities – thus evoking Bellotti *et al.'s* email threading paper of 2003 [1]. Like Jones *et al.*, none of these attempted to address the design of data stores, focusing instead on the interface.

More recently, Oleksik *et al.* [10] have presented a system that enables a form of threading and association between

digital entities. Users create this through a tagging system. These tags hide the precise provenance or location of a digital object – a PowerPoint file, a Website (URL) or a Word file say – and instead present the user with a collage of thumbnails bundled together into a 'cloudlet' or similar graphical representation. These concepts are expressly designed with the technical affordances of multiple devices and cloud infrastructures in mind, and while the research leaves aside strong claims about engineering, it would appear that this new mode of interaction would further distance the connection between the user and file entities storing data. The abstractions would consist of metadata and, though looking similar to the user, being all represented as thumbnails, these metadata would actually point to different locations and entities from the system point of view. Some of these indicators would be files to which the system could read and write, while others would be to the capacity of the system to request copies and/or viewing rights from content actually stored on a webserver.

What these studies all affirm, however, is the merits of reconsidering the basic concept of files. This is especially the case as knowledge work develops beyond the production of documents, to hybrid and heterogeneous interactions around multiple digital entity types, which themselves have complex relations to devices and locations. In the view of these papers, files are only one of the abstractions that might cohere interfaces between the user and the systems engineer.

A File as a Leaky Abstraction

That this is so reflects how it is that, in recent years, there has been an increasing proliferation of devices and computers: heterogeneity certainly seems to be a word for labelling this new world. 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, spread sheets. 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, would appear to be a case in point.

However as Harter *et al.* note [7], the emergence of this complexity and diversity can disguise the persistence of the term and function of files in computer architectures. They find on the Apple desktop that the relationship between different types of digital entities (such as word processing and graphical objects) most often turns around file type definitions and abstractions, though these are ragged and inconsistent, with users thinking one thing and the system treating file types in another way. Besides, Harter *et al.* also note that the architecture often breaks down when the OS operates upon different data types since there is no consistency between the apps and the OS. This is all the more startling when one considers the hyperbole given to Apple's new data store, the iCloud.

For if it is the case that users are confronted with more data types, they are also increasingly encountering a new term, one describing a raft of services and technologies all of which were meant to let them 'back up' these 'things', these '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 [16], 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"

It may be unsurprising to a HCI audience that what users think of as a file could differ from what engineers devise their file abstractions to allow. Indeed, that this is so may be of little importance; as long as the user can use a file sensibly and the engineer design systems that are effective, what is the issue? As we have seen however, issues surrounding what a file means in regard to computers, though often subtle, are leading 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 are leading to muddles that cut across what it means to save one type of file as opposed to another, and highlight sensitivities regarding whether certain types of object that we might consider file-like, can be 'saved' at all.

"...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." [14, p. 372]

The term 'users' in the above statement probably refers to fellow engineers, and in this case the abstraction places much emphasis on a file describing a single unit of data. But the example that we presented at the outset, of how OneNote constructs its file storage system, illustrates how the very same application can have different underlying structures. Let us take another example to show what a smallest unit – the file – could be. Like all of the products in Microsoft's productivity suite, Word uses a 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 particular, special to Word.

At first glance it would appear that a Word file as conceived of in the application (and the OS) is 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 as well. 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.

Be that as it may, one can imagine that a typical user might find any of these particularities, the smallest allotment being a Word file, 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 expectation is that their commands 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.

Implications for a Grammar of Action

What this example (and the OneNote example from the introduction) shows is that the particular concept of a file and the associated assumptions that go with it are not universal, reflecting some common agreement amongst those who design applications. To coin a phrase from the philosopher Oswald Hanfling, the 'grammar of action' associated with the use of the concept can be and often is different in different instances [6]. The differences in grammar turn around what is understood by the term file, and relatedly, what action is meant when the terms 'create' and 'save' are used. Systems designers have come up with different ways of enabling these actions in different applications.

One needs to consider these complexities not only when they manifest themselves when new technologies, like the cloud, are being introduced. One ought to also consider them when one tries to understand what motivated earlier attempts to design actions around a file, undertaken when file systems were less complex and less diverse. The development of Xerox Star [22], 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.* [15] 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? 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…" [14, 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 Odom *et al.* [11]. 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 reported to be printed out and pasted into scrapbooks alongside photos. 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 alongside this metadata – the tags and comments – 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 into blocks 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, which 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 (see also [9])? 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 duplicate, or having and owning (as it were) the originating thing itself and keeping traces of when copies as somehow distinct 'lesser' entities are produced. Each type of 'copy' has implications for the ensuing social life of the digital entity in question.

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, from interviews reported in [12],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 most computer systems have not been designed so that a file is truly eviscerated when the 'delete' command is selected (instead simply re-addressing 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 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 (from [12]), 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 a computer 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. In the UK, where this research was conducted, 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, and further, that unpacking such a concept presents an occasion for rethinking what ownership might look like. Translating what was once a relationship between a user and a physical thing into one between a user and a digital thing is not simply a matter of replication and, even if it were, the reinforcing of one model of possession would mean disregarding other ways in which ownership is accomplished. Expanding the grammar of action to encompass possession means considering how to enable the doings that underpin what ownership looks like in the many parts of the world where file infrastructures are used. It means acknowledging the complexities that are associated with this concept and designing for a diverse range of actions. And it opens up the possibility that cloud computing could enable new kinds of practices to emerge, which change ideas about how individuals relate to 'their' data, and to each other via it.

CONCLUSION

Whatever future work does need undertaking – and there are 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 computer.

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 or enable a sense of shared ownership. 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

Many thanks to reviewers and to colleagues in SDS.

REFERENCES

1. Bellotti, V., Ducheneaut, N., Hoard, M. and Smith, L. 2003. Taking email to task: the design and evaluation of a task management centred email tool. In *Proc. CHI 2003*, ACM Press, 345-352.
2. Churchland, P. and Churchland P. 1995. *The Engine of Reason, The Seat of the Soul: A Philosophical Journey into the Brain*. Boston: MIT Press.
3. Cutrell, E., Robbins, D., Dumais, S. and Sarin, R. 2006. Fast, flexible filtering with Phlat. In *Proc. CHI 2006*, ACM Press, 261-270.
4. 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 (2000), 140-170.
5. Halasz, F. and Moran, P. 1981. Analogy considered harmful. In *Proc. CHI 1981*, ACM Press, 383-386.
6. Hanfling, O. 2000. *Philosophy and Ordinary Language: The Bent and Genius of Our Tongue*. London: Routledge.
7. Harter, T. Dragga, C., Vaughn, Arpaci-Dusseau, A. and Arpaci-Dusseau, R. 2011. A file is not a file: understanding the I/O behaviour of Apple desktop applications. In *Proc. SOSP 2011*, ACM Press, 71-83.
8. Jones, W., Phuwanartnurak, J., Gill, R. and Bruce, H. 2005. Don't take my folders away!: organizing personal information to get things done. *CHI '05 Extended Abstracts*, ACM Press, 1505-1508.
9. Marshall, C., McCown, F. and Nelson, M. 2007. Evaluating personal archiving strategies for internet-based information. In *Proc. IS&T Archiving 2007*, 151-156.
10. Oleksik, G., Wilson, M., Tashman, C., Rodrigues, M., Kazai, G., Smyth, G. Milic-Frayling, N. and Jones, R. 2009. Lightweight tagging expands information and activity management practices, In *Proc. CHI 2009*, ACM Press, 279-288.
11. Odom, W., Zimmerman, J., Forlizzi, J. 2011. Teenagers and their virtual possessions. In *Proc. CHI 2011*, ACM Press, 1491-1500.
12. Odom, W., Harper R., Sellen. A., Thereska. E. 2012. Lost in translation: understanding the possession of digital things in the cloud. In *Proc. CHI 2012*, ACM Press, 781-790.
13. Seltzer, M. and Murphy, N. 2009 Hierarchical file systems are dead. In *Proc. HotOS 2009*, USENIX Association, 1-1.
14. Silberschatz, A, Galvin P.B and Gagne G. (2002) *Operating System Concepts, (6th Ed)*. New York: Wiley.
15. Smith, D.C., Irby, C., Kimball, R. and Harlsem, E. 1982. The Star user interface: an overview. In *Proc. AFIPS 1982*, ACM Press, 515-528.
16. Spolsky, J. 2004. *Joel on Software*. Berkeley: Apress.
17. Star, S.L. 2010. This is not a boundary object: reflections on the origin of a concept. *Science Technology & Human Values 35*, 5 (2010), 601-617.
18. Star, S.L and Greisemer, J.R. 1989. The structure of ill-structured solutions: boundary objects and heterogeneous problem solving. In L. Gasser and M.N. Huhns (Eds.) *Distributed Artificial Intelligence: Vol. 2*. San Francisco: Morgan Kaufmann, 37-54.
19. Thereska, E., Gosset, P. and Harper, R., 2012. Multi-structured redundancy. Presented at *HotStorage 2012*, June 2012, Boston, MA. https://www.usenix.org/system/files/conference/hotstorage12/hotstorage12-final6.pdf
20. Voida, S., Mynatt, E.D. and Edwards, W.K. Re-framing the desktop interface around the activities of knowledge work. In *Proc. UIST 2008*, ACM Press, 211-220.
21. WinFS. Wikipedia, accessed 23 August 2012. http://en.wikipedia.org/wiki/WinFS
22. Wittgenstein, L. 1953. *Philosophical Investigations*. Trans. A.N. Anscombe. Oxford: Blackwell.
23. Xerox Star. Wikipedia, accessed 23 August 2012. http://en.wikipedia.org/wiki/Xerox_Star.