HUMAN-CENTRED COMPUTER ARCHITECTURE: REDESIGNING THE MOBILE DATASTORE & SHARING INTERFACE

By
Thomas Oliver Reitmaier

Dissertation submitted for the degree of
Doctor of Philosophy
in the Department of Computer Science
University of Cape Town

October 2018

ABSTRACT

This dissertation develops a material perspective on Information & Communication Technologies and combines this perspective with a Research through Design approach to interrogate current and develop new mobile sharing interfaces and datastores. Through this approach I open up a line of inquiry that connects a material perspective of information with everyday sharing and communication practices as well as with the mobile and cloud architectures that increasingly mediate such practices. With this perspective, I uncover a shifting emphasis of how data is stored on mobile devices and how this data is made available to apps through sharing interfaces that prevent apps from obtaining a proper handle of data to support fundamentally human acts of sharing such as gifting.

I take these insights to articulate a much wider research agenda to implicate, beyond the sharing interface, the app model and mobile datastore, data exchange protocols, and the Cloud. I formalise the approach I take to bring technically and socially complex, multi-dimensional and changing ideas into correspondence and to openly document this process.

I consider the history of the File abstraction and the fundamental grammars of action this abstraction supports (e.g. move, copy, & delete) and the mediating role this abstraction – and its graphical representation – plays in binding together the concerns of system architects, programmers, and users. Finding inspiration in the 30 year history of the file, I look beyond the Desktop to contemporary realms of computing on the mobile and in the Cloud to develop implications for reinvigorated file abstractions, representations, and grammars of actions. First and foremost, these need to have a social perspective on files.

To develop and hone such a social perspective, and challenge the assumption that mobile phones are *tele*phones – implying interaction at a distance – I give an interwoven account of the theoretical and practical work I undertook to derive and design a grammar of action – showing – tailored to co-present and co-located interactions. By documenting the process of developing prototypes that explore this design space, and returning to the material perspective I developed earlier, I explore how the grammars of show and gift are incongruent with the specific ways in which information is passed through the mobile's sharing interface.

This insight led me to prototype a mobile datastore – My Stuff – and design new file abstractions that foreground the social nature of the stuff we store and share on our mobiles. I study how that stuff is handled and shared in the Cloud by developing, documenting, and interrogating a cloud service to facilitate sharing, and implement grammars of actions to support and better align with human communication and sharing acts.

I conclude with an outlook on the powerful generative metaphor of casting mobile media files as *digital possessions* to support and develop human-centred computer architecture that give people better awareness and control over the stuff that matters to them.

PUBLICATIONS

Some ideas, figures, and tables of this dissertation have previously appeared in the following publications:

Reitmaier, T., Benz, P., & Marsden, G. 2013. “Designing and Theorizing Co-Located Interactions.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI ’13*, 381–390. New York, NY, USA: ACM. doi: 10.1145/2470654.2470709

Robinson, S., Pearson, J., Reitmaier, T., Ahire, S., & Jones, M. 2018. “Make Yourself at Phone: Reimagining Mobile Interaction Architectures With Emergent Users.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI ’18.* New York, NY: ACM. doi: 10.1145/3173574.3173981

Walton, M., N. J. Bidwell, A. Venter, T. Reitmaier. 2014. “Mobile Technologies and Society in South Africa: Towards Transdisciplinary Perspectives on Mobile Social Media”, Panel Discussion at the South African Communications Association 40th Annual Conference: SACOMM ’14, Potchefstroom, South Africa, October 1

ACKNOWLEDGMENTS

I would like to thank the many people and organisations who have supported me throughout this research.

This dissertation would never have come together without the friendship, support and insights from many others.

First and foremost I am deeply grateful to Edwin Blake, Richard Harper, and Matt Jones, who have at different times, and in different places, given insights and guidance that are intertwined throughout this dissertation. Without their support I would have never got this far.

My friends and colleagues at the ICT4D Centre – particularly Anja, Shikoh, Grace, Fiona, Tsabi, Alette, and Ntwa have helped both during this work and beyond during much needed lunch and coffee breaks. Thank you all. I am equally grateful to my tremendous colleagues at Swansea University: Jen, Simon, Doon, and Stephen.

The financial support of Microsoft Research through their PhD Scholarship programme and the University of Cape Town's Computer Science Department has funded this research throughout, and for this I am particularly grateful.

My partner Alexandra, my parents Jürgen and Angela, and my sister Claudia have all accompanied me on this journey and have supported and encouraged me in countless ways. I cannot begin to express the gratitude I have for them.

Finally, I thank the late Gary Marsden who provided the inspiration and environment for this research to flourish and for me to grow as a person and researcher. I could not have asked for a better mentor and friend. It is to you, Gary, to whom this dissertation is dedicated.

CONTENTS

1 INTRODUCTION

The bit changes everything. More than 20 years ago, Nicolas Negroponte (1996) made this argument in his bestselling and widely influential book: *Being Digital*. In it he popularised the human-computer interface that underpins ubiquitous computing and characterised and promoted a utopian, interconnected world of free-flowing information enabled by *the bit* – a number that can either be zero or one. *Being Digital* can be seen as a synthesis of the many columns he wrote for *Wired*, a popular technology magazine, celebrating *the bit* and characterising its difference from and relation to *the atom*[1]. Over the past 20 years technology pundits have celebrated how digital technologies substitute bits for atoms, a move that transforms physical objects into their digital equivalents and, consequently, the society we live in into an information society[2]. Electronic health records, online-only newspapers, on-demand movie rentals, MP3s, electronic voting, etc are all evidence of this shift. For instance, when I moved to South Africa, I ripped all of my CDs in order to travel lightly. My music is now stored as bits on my hard-drive – where it only occupies hard-drive space – whereas my CDs are collecting dust in boxes in my parents' flat in Germany; bits have replaced atoms.

"A bit", according to Negroponte, "has no colour, size, or weight, and it can travel at the speed of light. It is the smallest atomic element of the DNA of information" (1996, 14). As atoms are the building blocks of the physical world, following Negroponte (1996) we might reason that by substituting bits for atoms means that the world we live in is, to some extent, becoming less physical. My now digital music collection exemplifies this. However even Negroponte – a self proclaimed technology optimist – concedes that "you cannot experience a bit"; instead, that bit "must [first] be turned back into atoms for human beings to enjoy it" (Negroponte 1995). This happens at the human-computer interface where bits and people meet. So while our experience of bits may no longer be physical, they still are *material*.

Most contemporary accounts of technology development, however, tend to de-emphasise the material properties of networks and devices as well as the information that flows through or are stored on them. Such accounts only pay attention to their material properties when things break down, for instance when in March 2013 a major fibre-optic cable, which links the African and European continents, broke just off the coast of Egypt (Mawson 2013) or when Amazon Web Services suffered a major outage in September of 2015 (Fiveash 2015). All of a sudden the material properties of the Internet are foregrounded. Just as we often forget that our devices and networks are material, we also tend to think of our data as somehow 'immaterial'. While my music and photo collections are encoded as a sequence of bits, these bits are stored on my hard drive, where the two magnetic polarities are used to represent either 0 or 1. This hard drive and the information stored on it is, of course, physical. However, such 'implementation details' are often glossed over in favour of more abstract – and therefore less physical – accounts of data, especially when we adopt the ineffable metaphors of the cloud computing paradigm. However much we might favour bits over atoms or abstract away from the physical properties of data storage and transfer, Negroponte original observation still holds – that for people to enjoy their digital 'stuff', the bits that constitute that stuff must be turned back into atoms. Or put another way, we experience our digital 'stuff', however 'immaterial' it may seem on the surface, as material.

1. see Negroponte (1993–1998)
2. on this point see Dourish & Mazmanian (2013).

For the Computer Scientist such a view might seem non-sensical. How can something digital be material at the same time? However, when we walk across campus and engage with disciplines outside of the science faculty, we encounter commentators from the social sciences and the humanities for whom studying the material reality of 'stuff' – here referring to mostly physical stuff – is as much about studying artefacts as it is about the social phenomena and practices they intertwine with. Drawing heavily on the work of feminist theorist Karan Barad (2003), Lucy Suchman brought this line of reasoning that transcends object-subject dualisms to the attention of mainstream HCI and CSCW scholars. In her efforts "to rethink the intricate, and increasingly intimate, configurations of the human and the machine" Lucy Suchman comes to the conclusion that "Human–machine configurations matter [...] because cultural conceptions have material effects" (2007, 1). This quote deserves some unpacking, for it links together *the digital* with *the material* and demonstrates the two related meanings of the words 'matter/material'. In the first instance, something 'matters' or is 'material' when it is of importance or consequence. In the second instance 'material' refers to what we ostensibly call the *What* – or 'stuff' – of physical forms. In more recent work on the material nature of information, Dourish & Mazmanian argue that a material perspective of information doesn't position these two meanings in opposition, for "that which carries 'material consequences' is always also physical and that which is physical occasions 'material consequences' " (2013, 96).

Over the past years Paul Dourish, Melissa Mazmanian and their colleagues have published a collection of interrelated articles that investigate the materialities of information. These investigations include inquiries into the materiality of internet routing (Dourish 2015b), the materiality of information *representation* (Dourish & Mazmanian 2013), a differential analysis of the Internet that positions it as a feudal system (Dourish 2014), and finally the materiality of database technologies at a moment when these materialities are shifting (SQL vs NoSQL) (Dourish 2014). Taken together these articles form a comprehensive portfolio that underline the importance of a *material* perspective of information, databases, networks, and protocols that challenges pervasive assumptions held by technologists and sociocultural analysts, alike: from the former's perspective that technological objects are mostly seen as only that, namely technological (Dourish 2015b, 274); and from the latter's perspective that technological arrangements while seen as already social are rarely scrutinised at the foundational level of computational objects and processes that practically instantiate technology (Dourish 2015b, 294). A *material* perspective thus opens up a fruitful in-between space nestled between computer science on the one hand and the social sciences and humanities on the other. Both provide an equally important basis that lets a *material* perspective speak of specific technological arrangements in all their technical detail, without regressing to technological determinism, and at the same time appreciate the social reality and sociocultural consequences of these arrangements.

One important contribution of this dissertation is to study from a material perspective the arrangements, constraints, and consequences of contemporary *mobile* architectures and cloud infrastructures. But we also see an important contribution in reconfiguring some of these arrangements with the goal of making and exploring alternative possibilities that find their inspiration in and aim to better support human communication practices as well as the values that surround these practices that we have observed in our own or our colleagues' research projects in South Africa. The material perspective as advocated by Dourish, Mazmanian and their colleagues is foundational to tackling and solidifying the former contribution. But that perspective, as we argue next, also meshes well with the Research through Design approach we adopt in this research to tackle and solidify the later contribution. In a nutshell, Research through Design (RtD) is "a research approach that employs methods and processes from design practice as a legitimate form of enquiry" (Zimmerman et al. 2010, 310). Design practices, of course, have a long tradition of emphasising *material engagements*. The design theorist Donald Schön (1983), for example, famously characterised design as a reflective conversation with materials, whereby a designer, in Schön's case an architect, approaches a design problem with the methods and tools of their training while remaining responsive to the design situation's and the design material's 'talk back'.

At this point we think it timely to introduce the subject of our study – our design material, if you will – the mobile smartphone. The mobile phone is a computational device that has brought about a sea-change in how many of us live our lives, shaping some of our most common everyday experiences: how we interact with each other through mobile media (e.g. Walton 2014; Farman 2012), mobile messaging (e.g. Taylor & Harper 2003; O'Hara et al. 2014), and calls (Ling 2008). Its touch-screen interface, combined with a myriad of built-in sensors, and its portable personal nature render the mobile distinct from the desktop computer of the pc era. Finally, it is impossible to speak of the mobile smartphone without also talking about the Internet, the Cloud, and the various technologies, protocols, and architectures these are composed of. In short, throughout this dissertation we find ourselves in conversation with and listening to the 'talk back' human practices and the values that surround these just as much as with the interfaces, architectures.

Dourish (2014), in a later article exemplifies this approach and looks at the materialities of database systems. He too sees the connection between a material perspective and design discourses and draws on Schön's (1983) characterising of design as a conversation with materials and contends that when speaking of materials of information, we are talking about those things that we, as members of contemporary society, find ourselves in conversation with. Dourish (2014), however, misquotes Schön by characterising that conversation as 'reflexive' instead of 'reflective'. Although unfortunate, this misquote presents an interesting point of departure for us. Dourish & Mazmanian's professional as well as everyday orientations towards digitally rendered data are crucial to how they unpack and situate their critical judgments of materials of information and their representations (2013). It exemplifies the benefits of extending the 'reflective conversation' into one that is also reflexive. An orientation that I follow.

It is here, where there are overlaps and shared epistemic stances between a material perspective, as articulated by Dourish & Mazmanian (2013), a Research through Design approach that integrates theories from other disciplines (Zimmerman et al. 2010, 316), and what Bardzell & Bardzell (2015) label Humanistic hci, where Humanistic approaches and epistemologies are not only taken seriously but also put in service of hci. In delineating a Humanistic hci, Bardzell & Bardzell observe that it can be difficult to distinguish between theories and methods; for in practice, the relation between researcher, theory, methods, and data is often blurred. Humanistic approaches in particular are necessarily and deliberately not well formalised to leave room for 'creative adapting' that is crucial to situating research (2015, 33). A consequence of this approach is that it is not always possible to state upfront the theories and bodies of work I engaged with, as is typical in a literature review, but rather to situate them throughout the dissertation. In critically reflecting on one of my past research projects (Reitmaier 2011), I have found that these observations resonate with my experiences of putting methods – in my case design methods – and the theories they mesh with, and the reflexive conversations they depend on, into practice to design a mobile digital storytelling system to suit the needs and functions of a rural African community.

Perhaps at this point is a good time for us to follow what Bardzell & Bardzell call "counteracting the god trick" (Bardzell et al. 2015, 8) and introduce your narrator as well as the places and people that have shaped this research. This strategy takes its cues from feminist philosophy and reframes objectivity, by positioning it as a contingent accomplishment where claims can only be understood and accounted for "in relation to [. . .] contingent positions, rather than as the voice of a hidden all-knowing god" (Bardzell et al. 2015, 8). This positionality inevitably always introduces biases that are crucial to account for and acknowledge, rather than counteract and ignore.

My perspectives as a person and as a computer scientist have been strongly shaped by my decision to abandon studying *Informatik* (Computer Science) at the RWTH Aachen in Germany midway through the Diploma, after spending a year studying abroad at the University of Cape Town (UCT) in South Africa through which I obtained an BSc Honours degree. Instead, I pursued a research intensive Masters degree at UCT. Studying and practicing Computer Science in post-apartheid South Africa laid bare how little I knew of the physical, social and cultural realities of people living in diverse communities and how poorly the mobile phone fit into these realities. I benefited tremendously by studying with and learning from great teachers: Gary Marsden and Nicola Bidwell and the communities they worked with. By collaboratively designing, developing, and evaluating a mobile digital storytelling system I learned to appreciate how important anthropological and sociological theories and perspectives are for the study of computer science – an insight I pursued by reading over 40 books on the topic since completing my Masters. A brief research internship at the interdisciplinary Computer Mediated Living group at Microsoft Research in Cambridge, UK that combines the ideas and insights of sociologists, psychologists, engineers, and designers further inspired me to bring my research in conversation with disciplines and perspectives outside of computer science's typical remit.

I draw on these readings and perspectives both critically and generatively throughout this research, but they also present a bias that the astute reader might notice, that I tend to gravitate towards and find inspiration in the work of anthropologists and sociologists more generally and within HCI towards research that has socio-cultural inclination. My hope is that my undergraduate studies of Computer Science and HCI as well as my experience in designing and programming mobile, web, and desktop applications provide a technical foundation through which I can engage with such diverse commentators.

Next, I want to draw attention to the fact that this research was primarily conducted at the Centre in ICT for Development at the University of Cape Town and is thus shaped by countless interactions I've had with colleagues and collaborators at and traveling through the Centre, which focuses on producing new technologies for the developing world and studies how technologies are creatively (re)appropriated by under-resourced communities. But it is also shaped by a second research internship at the Human Experience and Design group under the mentorship of Richard Harper and thus brings the above alternative perspectives into conversation with those located at more dominate and resource-rich centres of technological research and innovation.

Finally, I acknowledge that this thesis is also an expression of grief and honour for my original supervisor, the late Gary Marsden and the thankfulness I have for Edwin Blake (UCT), Richard Harper (Microsoft Research), and Matt Jones (Swansea University) for taking me under their wings and helping me complete this research. But this also means that I have been more reluctant to change or challenge the ideas that Gary and I developed together. Perhaps, holding on to them more than Gary would have liked. Without Gary's HCI expertise and supervision, this research undertaking became by necessity more technical and conceptual.

For these reasons, I at times depart from the Western academic convention of writing in the third-person to emphasise and acknowledge to the reader, and ultimately take responsibility for, the active role I played in conducting and mediating this research. When I deem such emphasis less important, I transition back to the more conventional third-person voice.

With your narrator introduced, let us return to the material/design approaches we follow in our research. To distinguish Research through Design from design practice, some have argued that RtD should include a 'theoretical scaffolding' (Zimmerman et al. 2010, 311). The material perspective we summarised above, not only shares a stance that emphasises material engagements, but can further serve as such a 'theoretical scaffolding' that opens up a cross disciplinary, reflexive conversation between computer science and the social sciences and humanities that can appreciate and integrate the various disciplinary orientations towards the mobile phone. Bardzell & Bardzell conclude their chapter on 'Humanistic HCI and Methods' with the following outlook (2015, 64):

> We believe that it is this relationship between design and humanistic ways of knowing that has formed a large part of the backbone of so-called 'third wave HCI,' with its emphases on aesthetics, experience, sociotechnical systems, futuring, and social change. Underneath it all is a desire – a need – to explore alternative ways of being, to imagine their potentials both good and bad, and to build research and design agendas that guide humans toward preferred ways of being. One way to meet that need is to make – to design, to craft, to make art, to prototype – and then to engage such works with critical rigor.

What Bardzell & Bardzell (2015) set into dialogue are two perspectives that modern thought has kept separate: those of the craftsman making things and the theorist thinking them with 'critical rigor'. The differences in these perspective, as the Anthropologist Tim Ingold in his treatise on *Making* argues, is not that the former only makes and the later only thinks, but that the craftsman 'thinks through making' and the theorist 'makes through thinking' (2013, 6). This is a theme that Frayling also identified in his foundational essay on *Research in Art and Design* (1993). Namely that research should not be considered the exclusive provenance of professional scientists, what Frayling (1993) calls *Research* with a big 'R'. It should instead re-establish its historical connections with *art* and *design*, what Frayling (1993) calls *research* with a small 'r'. Research through Design, Humanistic HCI, and to an extent our material perspective place an emphasis on the relationship between thinking (or theorising) and making that corresponds more with the perspective of the craftsman, who allow their "knowledge to grow from the crucible of our practical and observational engagements with the beings and things around us"; this is what Ingold (2013, 6) calls practising the *art of inquiry*:

> In the art of inquiry, the conduct of thought goes along with, and continually answers to, the fluxes and flows of the materials with which we work. These materials think in us, as we think through them. Here, every work is an experiment: not in the natural scientific sense of testing a preconceived hypothesis, [. . .] , but in the sense of prising an opening and following where it leads.

With these two quotes in mind and your narrator as well as the subject and broad approach of our research undertaking introduced, we set off to make, study, and critique the contemporary mobile smartphone.

1.1 THE ARGUMENT OUTLINED

Chapter 2 launches us straight into this research undertaking by asking a deceptively simple question. *Does Windows Phone 8 support gifting?* – that is, can we share an object in such a way that it is removed through the act of giving? The purpose of this question is to setup the contemporary smartphone as our 'field site' and to open up a line of inquiry that connects a material perspective of information with the mobile and cloud architectures that increasingly mediate age-old human practices such as gifting. On Windows Phone 8, we explore the mobile media objects that are commonly encountered on mobiles – photos and videos – and the mechanisms, interfaces, and security principles through which these objects are made available to users and app developers. We discover that the Windows Phone 8 sharing interface does not support gifting, because the mobile media accessed through it is only ever encountered, rendered, and shared as a stream of data and not as an entity that can be removed, for instance as a file. We ask the same question of the Android Operating System (OS) and uncover a similar phenomenon, albeit as a more open platform we were able to engineer a workaround and implement an app that explores the above gifting fiction. However, this gifting fiction would only work with photos kept in public storage (where the OS has direct control) and not those hidden inside app sandboxes (where the OS relinquishes control). The gifting fiction illustrates this battle in the sandbox that forces apps to recreate files in sandboxes where they fall outside of the OS's and the users direct control.

In Chapter 3 we explore this deeper issue and outline a wider research (through design) agenda that rearticulates research objectives: to implicate and coordinate mobile system architectures, datastores, the Cloud, and data exchange protocols in the act of sharing with the goal of giving everyday people better awareness and control over the stuff that matters to them and ultimately allow them to express the nuances of how that stuff is shared. We consider foundational books and early scholarship on the important role of the file metaphor/abstraction in system architecture and desktop interface design. The foundational grammar of action (e.g. move, copy, delete) and desktop metaphor of the PC, as more recent scholarship argues has over the past 30 years bound together the concerns of system architects and users alike, but its waning prevalence as a primary and visible metaphor of mobile system design illustrates that a new *grammar of action* and reinvigorated *file abstractions* are needed rather than abandoning and replacing these altogether. The gifting fiction of the previous chapter exposed precisely these shortcomings and demonstrates that we need a social view of files.

In Chapter 4 we interrogate current, and start to imagine what new grammars of action, might look like by attending to the rich social practices that surround and give meaning to the use of mobile phones. We synthesise diverse accounts and theories of how people interact when co-present through our research through design process and develop a simple prototype on Android OS that uses the built-in sharing interface to support *showing* photos. We consider pressing resource constraints of communities in South Africa to motivate the implementation of a more comprehensive photo gallery tailored for collecting and subsequently presenting photos face-to-face.

In Chapter 5 we pause and reflect over the prototypes we designed and developed in Chapters 2 & 4 and consider the major architectural changes introduced by the converged Windows 8.1 and Windows Phone 8.1 platforms that sees the file re-introduced as a major component of the Windows Phone 8.1 platform, its datastore, and its sharing interface. We analyse these early attempts to re-introduce the file and interrogate the resulting muddle when sharing is equated to copying. Finally, we develop the first iteration of the *My Stuff* datastore prototype to tidy up the muddle and deepen our investigation.

In Chapter 6 we integrate our material perspective with a nascent field of research that positions the stuff that people store and share on their personal devices and in the Cloud as a form of digital possession. We conduct a critical analysis of the possibilities and constraints of contemporary cloud architectures to better understand how we can support notions of possession when files are no longer just created and stored on individual devices, but also shared with others and travel through the Cloud. We extend the *My Stuff* datastore to interface and transact with a Cloud service that we developed using common technologies and established design patterns. Finally, we develop new file abstractions that encompass and represent the social life a file gains when it is shared. This work was conducted at Microsoft Research in Cambridge, UK.

In Chapter 7 we relocate our research back to Cape Town only to discover that the *My Stuff* prototype and Cloud service did not work in the same way it did in Cambridge. We deepen our investigation and extend research into the materialities of information by interrogating how physical location affects our use of the Cloud. We show how good infrastructure can mask problems with computer architecture, and document how we re-architected the *My Stuff* datastore and Cloud service to better cope with the material realities of marginal internet connectivity and find inspiration in local mobile media materialities and sharing practices that articulate a need for mobile media to travel freely between local media ecologies and the Cloud.

2 ENGINEERING TO GIFT

2.1 INTRODUCTION

The work presented in this chapter is a fragment of a more complete study, or rather it can be seen as an introduction to this study. It starts with a deceptively simple question:

> Does Windows Phone 8 support gifting?

The purpose of asking this question lies not such much in its answer but in its ability to present an engineering challenge that provokes reflection and discussion and ultimately opens up a line of inquiry that seeks to connect a material perspective of information with everyday sharing and communication practices as well as with the mobile and cloud architectures that increasingly mediate such practices.

2.2 MOTIVATION

Before we discuss how we answer this questions and the implications it suggest, we first motivate why we choose to ask this question specifically.

2.2.1 *Gifting as an interface question*

In his seminal work *The Language of New Media*, Lev Manovich (2001) develops a rigorous and widely influential theory of new media. In that book Manovich introduces the database as the key form of cultural expression of the computer age (Manovich 2001, 218). While we touch on databases later on in this chapter and in depth in chapter 6, it is the second category/form that is very much unique to new media that we focus on here: *the interface*.

Manovich (2001) explores the human-computer interface from a media theory perspective. Starting with early desktop interfaces of the 80s such as the Xerox Star and the Apple Macintosh, Manovich retraces the history of the term human-computer interface. Back then the interface turned on the use of file and folder metaphors arranged on a desktop, but it also included what Manovich calls 'grammars of meaningful actions' (2001, 69) that still pervade contemporary desktop interfaces: copy, rename, delete, to name a few. In this chapter, we seek to first explore the 'grammars of meaningful actions' in more detail and then expand on them by engineering the grammar of 'to gift'. Our reason for doing so, can be found in Manovich's argument that the human-computer interface is now better thought of as a human-computer-culture interface, because "we are no longer interfacing to a computer but to culture encoded in digital form" (2001, 69-70). Following Manovich, we argue that contemporary mobile interfaces concern themselves with the distribution of cultural objects and would thus benefit from being studied from a sociological, *material cultural* perspective. Engineering 'to gift' makes such a perspective relevant.

2.2.2 *Gifting as a practice*

Gifting is an age-old practice that is found the world over. Its continued relevance shows that "everything is still not wholly categorised in terms of buying and selling", because some "things still have sentimental [...] value" (Mauss [1950]

2000, 65). A gift might take the form of an object, but far from being innate they are imbued with a force that somehow "causes its recipient to pay it back" (Mauss [1950] 2000, 3), for instance when we feel obliged to return an invitation or to reciprocate by buying the next round of drinks. It is in this sense that gifts transform an innate object into something that enhances intimacy and ultimately binds people together. Theses bindings give gifting practices their prevalence and continued importance, for they are constitutive of social relations. This has lead Marcel Mauss in his comparative essay on *The Gift* to characterise gift exchanges as a *total social system*. It can thus help explain other types of exchanges, even those mediated by mobile phones. For instance, Taylor & Harper (2003) draw parallels between gift exchanges as articulated by Mauss ([1950] 2000) and how young people create and exchange content on their mobile phones (at the time, mostly text messages). The salient property of text messages that allows them to draw such parallels is that they have value, and "this value is connected with the giver, the recipient and the context in which the exchange takes place, and is embodied and retained in material form" (Taylor & Harper 2003, 268). Kindberg et al. (2005b) extend that argument to people's use of camera phone photos for social and affective purposes.

I could go on to give many more examples or dive deeper into the sociology of gifting, but at this point the evidence should be adequate to claim that the question of whether Windows Phone 8 supports gifting is at the very least a question that deserves to be posed. I worry too that further theoretical treatments of gifting, risks losing sight of how people already understand and *do* gifting. For this we don't need to draw upon essay and manuscripts. We – our ordinary, everyday selves – already know what gifting is all about: we understand its significance, its ritual, its obligations because gifting practices are carried out "in those contexts in which we claim to be 'at home' " (Ingold 2000, 330) and, in fact, contribute to "our being at home in the world" (Ingold 2000, 333). It is this reflexive, common-sense understanding of gifting that I want to emphasise and carry forward as we translate and formalise what is meant when we talk about gifting, especially in the context of mobile phones and mobile media.

2.3 FORMALISING 'TO GIFT'

We should, however, begin our process of formalising gifting by recognising that it is not possible to formalise gifting and *all* that it entails, for when we speak about gifting we are also implying all those many things that abhor specification: its intentionality, mindfulness, and generosity to name just a few. One approach of moving forward is to set aside for a moment all those things that we might call the social context of gift exchanges (we will return to those later) and focus instead on the mechanics of the exchange. In which case, we arrive at the following ostensible formalisation of gifting:

> To gift an object, the giver first presents and then gives (transfers) the object to the recipient.

But such a formalisation does not quite share the 'logical form', to borrow a phrase from Wittgenstein, of our everyday understanding of gifting. For instance this formalisation would allow me to gift the reader his or her watch, an object that was never mine to begin with and therefore not mine to gift. Likewise, the formalisation doesn't quite capture what the recipient can do with the gift after they have received it. What makes the gifted object different from the one which has only been borrowed? To answer to these concerns and recover at least some of the social context of gifting that we set aside earlier, we should label gifts not as objects, but as possessions.

> To gift a possession, the giver first presents it and then gives (transfers) the possession to the recipient.

This more nuanced formalisation would not enable me to gift the reader his or her watch, but would still allow me to gift a photo that I took, or a box of chocolates that I bought. In later chapters we will unpack the concept of possession and how it relates to the digital content that pervades our lives and that is increasingly also becoming the 'stuff' of sociality. At this early stage we drill down into the objective reality of gifting as a transfer of objects, keeping in mind that at this scale we could never fully account for its social and cultural reality. Nevertheless it feels like a truism to say that in order to gift some*thing*, you must first encounter that thing as an object. It is in this sense that we see the relation between the terms possession and object, at least in the context of gifting.

With this formalisation provisionally in place, we next turn to the Windows Phone 8 platform.

2.4 DOES WINDOWS PHONE 8 SUPPORT GIFTING?

We can speak of mobile message and media sharing practices as a form of gift giving, because the value of such practices "is embodied and retained in material form" (TAYLOR & HARPER 2003, 268). They thus invite a material perspective. Studies of the material realities of the digital are, as Dourish reminds us, best achieved comparatively (DOURISH 2015a) and at times when the very materialities of a technology are in flux (DOURISH 2014). Despite our previous experience developing Android Applications[1] we deliberately choose to explore the Windows Phone 8 platform to render the mobile strange[2] again and provide such a comparative perspective. As a comparative study, we will also consider our observations in relation to more popular mobile platforms such as Android. Although not commercially successful with Windows Phone 8, Microsoft developed an Operating System (OS) that stands as a significant point of departure from every other mobile OS on the market: it focuses on the user's content and not app chrome (WHITECHAPEL 2013, 3) and provides a fresh approach (CHIMERO 2013). The Metro design language, which was first introduced with Windows Phone 7, has been expanded and, with the Windows 8 operating system, an attempt has been made to unify it across desktop and mobile experiences. This has further interesting implications for the materialities of information that underwrite and are underwritten by this design language. It is these that we want to unpack and that motivate our choice of studying the Windows Phone platform.

2.4.1 *The objects one encounters on the phone*

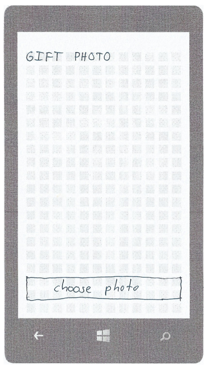
One type of object that we encounter often on the mobile is the photo. It is an object that users can create at a push of a button. But it is also an object that is both deeply personal and social. We leverage photos to communicate (KINDBERG et al. 2005a), tell stories (REITMAIER et al. 2011), and produce and convey identity (VAN HOUSE 2009) and remember (DIJCK 2008).

2.4.2 *Sketching a Windows Phone App*

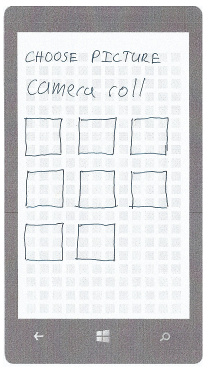
On the mobile, there are four ways through which apps can access photos stored on the device:

1) By using the `PhotoChooserTask` API an app can acquire a single photo from the user.

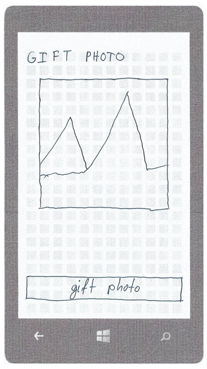
1. see for instance, Reitmaier et al. (2012), Bidwell et al. (2014) & (2014)
2. on this point see Bell et al. (2005).



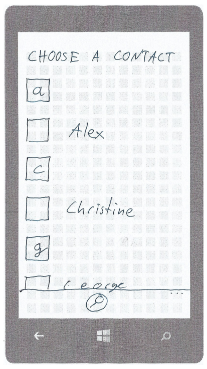
(a) Main screen.



(b) User selects photo.



(c) Display selected photo.



(d) User selects contact.

Figure 1: Gifting Screen Flow using the PhotoChooserTask.

2) By registering for the `Photos_Extra_Share` extension in an app's manifest, that app can become a 'share target' and will be listed alongside built-in apps such as email or messaging in the WP8 share picker.
3) By using the `MediaLibrary` API an app can access all of the user's photo albums and photos at once.
4) By using the `CameraCaputerTask` or `PhotoCamera` API an app can prompt the user to take a photo or build photo taking functionality straight into the app.

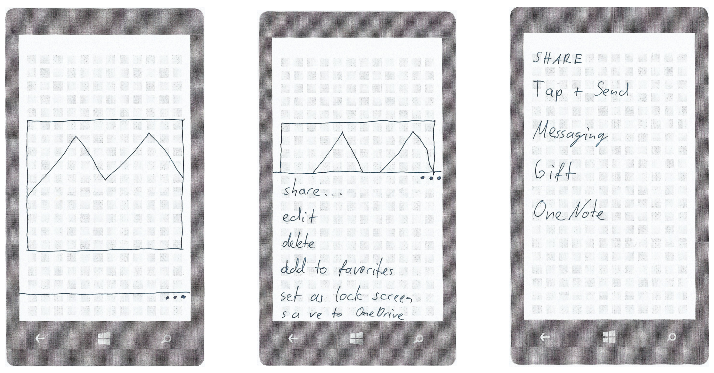
The first two options are by far the most common and provide a standard way for users to select a photo to gift. In figs. 1, 2 we have sketched out a screen flow for choosing and sharing a photo, respectively.

2.4.3 *Encountering Photos using the PhotoChooserTask*

We start with implementing the first three steps of the screen flow outlined in fig. 1. This is a straightforward process that involves creating an app with a single UI page: `MainPage`. This page contains a title `TextBlock`, an `Image`, and a `Button`. In the code-behind of that UI page we initialise the `PhotoChooserTask`, set the button to launch the `PhotoChooserTask`, and process the result of the `PhotoChooserTask`: creating a BitmapImage from the chosen photo and then displaying that image (see lst. 2.1).

2.4.4 *Encountering Photos using the share picker invocation point*

At this point we can also expand our app to provide the user with a second invocation point – the share picker (Whitechapel 2013, 661) – as outlined in



(a) User browses gallery.

(b) User 'shares' photo.

(c) User chooses gift; flow continues in fig. 1c

Figure 2: Gifting Screen Flow using the `Photos_Extra_Share` extension.

Listing 2.1 The main UI page's code-behind file: 'MainPage.xaml.cs'

```
public partial class MainPage : PhoneApplicationPage
{
   PhotoChooserTask photoChooser;

   public MainPage()
   {
     InitializeComponent();
     photoChooser = new PhotoChooserTask();
     photoChooser.Completed += photoChooser_Completed;
   }

   private void photoChooserButton_Click(object sender, RoutedEventArgs e)
   {
     photoChooser.Show();
   }

   void photoChooser_Completed(object sender, PhotoResult e)
   {
     if (e.TaskResult != TaskResult.OK)
        return;
     var image = new BitmapImage();
     image.SetSource(e.ChosenPhoto);
     chosenPhoto.Source = image;
   }
}
```

fig. 2. This is done in a two-step process. First we need to register our gifting app to be invoked from the share picker. This is done declaratively in the app's `WMAppManifest.xml` file using the `Photo_Extra_Share` extension point (see lst. 2.2).

Listing 2.2 Registering for the share picker invocation point in the app manifest.

```
<Extensions>
  <Extension
    ExtensionName="Photos_Extra_Share"
    ConsumerID="{5B04B775-356B-4AA0-AAF8-6491FFEA5632}"
    TaskID="_default" />
</Extensions>
```

When the app is launched through the share picker invocation point the Operating System passes a `FileId` parameter to the app as it is launched. By overriding the `OnNavigatedTo(NavigationEventArgs e)` method, the app can subsequently read out and act on this parameter, as lst. 2.3 demonstrates.

Listing 2.3 Extending the app to display a 'shared' photo.

```
public partial class MainPage : PhoneApplicationPage
{
// ...
  protected override void OnNavigatedTo(NavigationEventArgs e)
  {
    string fileId;

    if (NavigationContext.QueryString.TryGetValue("FileId", out fileId))
    {
      using (MediaLibrary mediaLibrary = new MediaLibrary())
      {
        Picture sharedPicture
          = mediaLibrary.GetPictureFromToken(fileId);


        var image = new BitmapImage();
        image.SetSource(sharedPicture.GetImage());
        chosenPhoto.Source = sharedBitmapImage;
      }
    }
  }
}
```

Here we first try and extract the `FileId` parameter from the `Navitgation-EventArgs`. If successful it means that our app has launched from the share picker invocation point, which in turn means that the user wants to share a photo with/through the gifting app. We handle the `FileId` as a string variable that stores a globally unique id, which might look similar to `66707067-c50d-475e-b465-704479c71a7a`. Next we make use of the MediaLibrary API to 'get' the Picture that is associated with that FileId token. We then create a `BitmapImage` from the Picture and can then finally display it as our chosen photo.

In sec. 2.4.1 we illustrated how photos are typical objects that users create, consume, and encounter on their mobile devices. Before we continue developing our gifting app, we pause to unpack how we specifically encounter photos – what are their materialities? – in our gifting app thus far.

2.4.5 *The materiality of a photo on Windows Phone 8*

In our gifting app, we encounter 'photos' in different ways. Once as a `PhotoResult` returned by a `PhotoChooserTask` and once as a `FileId` extracted from the `NavigationEventArgs` passed into the `OnNavigatedTo` method when our app is launched from the share picker invocation point. But as the astute reader will surely notice, neither one of these is what we would commonly refer to as a photo. So lets inspect them closer. Of the two 'photos', the `FileId` parameter is surely the least photo-like; we will tackle it last, and begin instead with the `PhotoResult`.

The `PhotoResult` (MSDN 2016a) class only has two properties that are unique to that class[3]:

- `ChosenPhoto` – Gets the stream containing the data for the photo.
- `OriginalFilename` - Gets the file name of the photo.

In our app, we used the `ChosenPhoto` property to first create and then display a bitmap of the photo's data. It would, however, be impossible to gift such a `ChosenPhoto` for we only encounter it as a "stream containing the data for the photo" (MSDN 2016a). The best we can do is to display it, or perhaps, save a copy of it.

Looking at the `OriginalFilename` property, which contains the full path to the chosen photo, it appears that we might be able to encounter the photo as an object that could be gifted: an object that can be removed once it has been transferred and successfully received by the recipient and the act of giving is complete. But first we'd need to get access to the file that is stored at the path of the `OriginalFilename` property, such as:
`C:\Data\Users\Public\Pictures\Camera Roll\WP_20160308_001.jpg`
On Windows this is done using the `Windows.Storage` API. But on Windows Phone this approach leads up a blind alley. As Whitechapel explains:

> In Windows Phone 8, the only portions of the `Windows.Storage` namespace that are supported are those with which developers can manage files and directories in their local storage folder and read files from their app package. Windows Phone 8 does not support notions of roaming or temporary data that are present in Windows, nor does it provide access to user content such as photos or music as an `IStorageFolder`. —Whitechapel (2013, 402).

As we are hindered in our quest to encounter, what we might ostensibly call, a 'giftable photo' using a returned `PhotoResult`, we next turn to the `FileId` parameter of our second approach. In lst. 2.3, we can see that it is only after we make use of the `MediaLibrary` API that we can transform the `FileId` into something that is at least more photo-like: a `Picture`. A `Picture` exposes a number of interesting properties (MSDN 2016b):

- `Name` – Gets the name of the Picture,
- `Date` – Gets the picture's date,
- `Height` – Gets the picture's height,
- `Width` – Gets the picture's width,
- `Album` – Gets the picture album that contains the picture.

In addition it exposes two public methods that are unique[4] to the `Picture` class:

3. by this we mean that they aren't inherited from superclasses.
4. Methods that aren't inherited from abstract super classes.

- `GetImage()` – Returns the stream that contains the image data,
- `GetThumbnail()` – Returns the stream that contains the picture's thumbnail image data.

In our app we used this first method to display the photo, but we again find ourselves in the same blind ally we got stuck in earlier, as we can't gift that which we only encounter as a stream. If we turn our attention to the five properties of the `Picture` class, we can quickly tell that the first four won't help us either, as they are simple, descriptive objects. The `Album` property, on the other hand, is more interesting. It is represented by the `PictureAlbum` class (MSDN 2016c) and has the following properties unique to that class:

- `Albums` – Gets the collection of picture albums that are contained within the picture album (that is, picture albums that are children of the picture album),
- `Name` – Gets the name of the PictureAlbum,
- `Parent` – Gets the parent picture album,
- `Pictures` – Gets the collection of pictures in this picture album.

Looking at these members, we can see how a `PictureAlbum` can contain any number of further, nested picture albums and subsequently each `PictureAlbum` has a parent `PictureAlbum` all the way up to the root picture album that is called 'Pictures' and which contains all the albums we also see in the built-in Gallery app: Camera Roll, Saved Pictures, Sample Pictures, and Screenshots. A `PictureAlbum` can also contain any number of pictures. Thus we can conclude that picture albums and the pictures and albums they contain are implemented in similar ways to a nested hierarchy of folders (albums) and files (pictures). To be sure, if we remove the SD card from a Windows Phone 8 device and read it from the computer, we can see the same hierarchy enfolding from the 'Pictures' folder: the root picture album, we mentioned earlier. And the pictures contained within those folders are regular `.jpg` files.

Exploring the `Picture`, `PictureAlbum`, and `PhotoResult` classes helped us to shed light on some of the material properties of photos and how they are encountered on Windows Phone 8. However when it comes to gifting, the first three of the four approaches we outlined in sec. 2.4.2 lead up a blind alley, where we only ever encounter a photo as a stream of data and not as an object that can be gifted. Strictly speaking the fourth approach, using the `PhotoCamera` class, allows us to save a newly captured photo in the app's sandboxed storage. Lst. 2.4 illustrates how this is achieved.

Listing 2.4 Saving a captured image in a local storage sandbox.

```
private void photoCamera_CaptureImageAvailable(object sender,
  ContentReadyEventArgs e)
{
  IsolatedStorageFile isoStore =
    IsolatedStorageFile.GetUserStoreForApplication();
  string filename = "WP_" + DateTime.Now.Ticks.ToString() + ".jpg";

  using (IsolatedStorageFileStream fileStream =
    isoStore.OpenFile(filename, FileMode.CreateNew))
  {
    e.ImageStream.CopyTo(fileStream, 1024);
    fileStream.Close();
  }
}
```

Listing 2.5 Deleting the previously captured image.

```
isoStore.DeleteFile(filename);
```

Within this storage sandbox, it would then be possible to delete the photo using the API call shown in lst. 2.5.

While we have finally discovered a scenario through which we would be able to engineer 'to gift' on WP8, we must concede that such a scenario, of gifting a just taken photo, is hardly compelling. We thus switched to Android, to explore if we might encounter photos in a 'giftable form' on that platform. This choice was driven by the popularity of the Android platform in South Africa, and by the fact that I am deeply familiar with the platform. However, we also note that Apple's iPhone is prohibitively expensive in South Africa[5] so a limitation of this work is that we were not able to experiment with the iOS platform.

2.5 SWITCHING TO ANDROID

At the time, the most recent version of Android was 4.4 (KitKat). Lst. 2.6 shows how we can prompt the user to select a photo from the phone's datastore on Android.

Listing 2.6 Prompting user to select a photo from the gallery on Android.

```
// set action to open an 'openable' document
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
intent.addCategory(Intent.CATEGORY_OPENABLE);

// of type image
intent.setType("image/*");

// execute action
startActivityForResult(intent, PROMPT_FOR_PHOTO_REQUEST);
```

Similar to lst. 2.1 on WP8 the Android app then needs to wait for and then process the results of this request. Lst. 2.7 shows how this is achieved.

It is again worth noting the similarities between the WP8 and Android platforms. On Android the selected photo is returned to the app by a document provider in the form of a `Uri`, which might look like this:

`content://com.android.providers.media.documents/document/image:29`

And just like on WP8, we can read the `Uri` into a `Bitmap` which we can then display in a UI `ImageView` element. But unlike WP8 we can – in certain instances – also tease out the file path from the selected photo's Uri. How this teasing can be done is the topic of two popular StackOverflow[6] questions (MIKEGR 2013; ÁLVARO 2016). For both questions the most helpful answer was provided by Paul Burke (2013), the author of the open source 'aFileChooser' library, which lets users select files on external storage. The solution that Burke proposes[7] is essentially a long list of if-then-else clauses that check, for instance if the `Uri` was returned by a DocumentProvider in which case it needs to be checked if the document represented by the `Uri` is stored on the phone's internal storage or on an external memory card, if it was downloaded and delivered by the `DownloadsProvider`, or if it was returned by the Gallery and its `MediaProvider`, or

5. The entry level iPhone 5s costs approximately $992 compared to $649 in the USA.
6. StackOverflow is an online community of almost 5 million programmers that help each other by asking and answering questions
7. this solution can be viewed at BURKE (2013)

Listing 2.7 Handing the results of the photo prompt request.

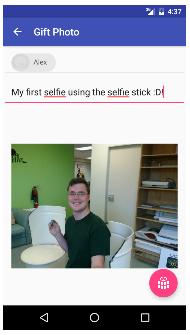
```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
 super.onActivityResult(requestCode, resultCode, data);

  if (requestCode == PROMPT_FOR_PHOTO_REQUEST && resultCode == RESULT_OK
    && data != null && data.getData() != null) {

    Uri photoUri = data.getData();

    try {
      Bitmap bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(),
        photoUri);

      ImageView imageView = (ImageView) findViewById(R.id.imageView);
      imageView.setImageBitmap(bitmap);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

perhaps the phone is running an older version of Android that doesn't support the `DocumentProvider` model in which case we fall back on the `MediaStore` or see if the `Uri` adheres to a file scheme. In each of these cases the file path is obtained in different ways. For instance in the case of the `MediaProvider` we first look at the last segment of the Uri: `image:29`. It can be broken down into two components: the document's type (`image`) and id (29). Ultimately we want to query the MediaStore, which is done using a `ContentResolver`, using the document id. As we are dealing with an image type, and not say a video or audio type, we set the `Uri` of query to the one stored in the `MediaStore.Images.Media.EXTERNAL_CONTENT_URI` variable. We then set the selection to `"_id=?"` and the arguments of that selection to the document id (29) we obtained earlier. Finally we set the projection of the query to the `"_data"` column, as the path is the only part of the query we are interested in. We can then query the database and return the string stored at the `"_data"` column, as it contains the file path of the image. But before we can read from the file or delete it, we need to request that the Android OS grant two additional permissions to our app that we declare in the app's manifest file (see lst. 2.8).

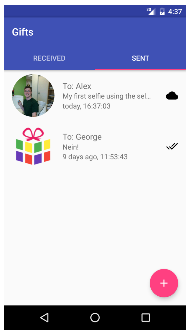
Listing 2.8 Requesting permission to access storage.

```
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

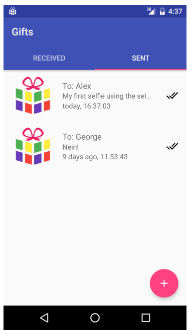
With all this in place, we can now finally access the file path of the photo `/storage/emulated/0/DCIM/Camera/IMG_20141106_082517.jpg` to read from, write to, as well as delete the file stored at that location. With this critical aspect of the app implemented we proceeded to implement a full gifting app prototype and accompanying web service.



(a) User selects photo and recipient, writes a message, and sends the gift.



(b) Gift is uploaded to server and notification is sent to recipient.



(c) Photo is deleted and status updated once recipient accepts the gift.

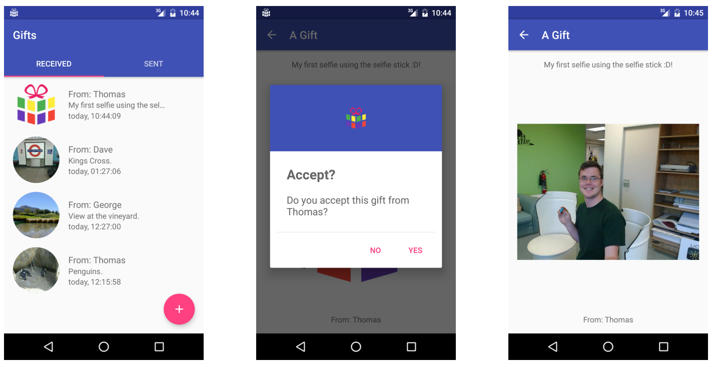
Figure 3: Sending a gift on the Android App.

2.5.1 *Prototyping a Gifting App on Android*

We started by creating a simple web-service using node.js[8], an event driven JavaScript runtime designed to build scalable network applications, and MongoDB[9], an open source document database. We used standard components to create a simple user account model and standard approaches to provide registration and access control. On the client side, upon launching the app users are prompted to register using their name, phone number, and password. The phone number and password tuple are used to login on the server side. If that tuple matches server records, the server returns an access token that is used to authenticate further requests. Fig. 5 outlines this interplay during registration between client app and API server. Later on we expanded the login method to further include the client app's push message token. This token is used by the API server to send push message to the client app using the Google Cloud Messaging Platform[10], for instance to alert users that they have received a gift.

Once registered and logged in, a user can gift a photo by first selecting a photo[11], and then selecting a recipient, either by selecting someone from the phone's address book or by manually typing in the recipients phone number. Finally users can compose an optional message to accompany their gift before 'sending' it off. Fig. 3a shows the UI that facilitates these steps. Once the send button is pressed the client app makes a server side request to 'create' a gift. That request contains the recipient's phone number and the access token that was returned by the server upon successful login. The server first checks if the request is authenticated and the recipient has an account. If these criteria are satisfied, the server creates a record of that request by storing giver and recipient phone numbers and assigning the request an ID. This ID is returned to the client app. The client app makes a second request which contains the ID and access_token and also uploads the photo's contents. On the server side, the photo is stored and a push message is sent to the recipient's app, which contains the gift ID and giver phone number, and the optional message. A record of this gift is stored by the client app in a list of 'sent' gifts (see Fig. 3b), which now shows this latest gift as uploaded but still pending. Fig. 5 shows how the API server mediates between the client apps of giver and receiver.

8. http://nodejs.org
9. http://mongodb.org
10. `https://developers.google.com/cloud-messaging/gcm`
11. This can be done from within the gifting app or by 'sharing' a photo with the gifting app, for instance from the phone's gallery



From: Thomas
My first selfie using the sel...
today, 10:44:09

From: Dave
Kings Cross.
today, 01:27:06

From: George
View at the vineyard.
today, 12:27:00

From: Thomas
Penguins.
today, 12:15:58

My first selfie using the selfie stick :D!

Do you accept this gift from Thomas?

From: Thomas

(a) User is notified of gift

(b) Upon clicking the notification, user is prompted to accept the gift

(c) Gift is downloaded, saved, and displayed

Figure 4: Receiving and accepting a gift on the Android App

Upon receipt of the push message, the client app of the gift's recipient processes the message and using Android's notification framework alerts the user that they have just received a gift. The app also stores and displays a record of that gift in a list of received gifts see 4a. Upon clicking on that record or on the notification in the notification drawer, the user is asked if they accept the gift (see Fig. 4b). Should the user accept, the client app makes a request to the API server containing the gift ID and the recipient's access token. In the response the server 'streams' the file's content, which is stored and displayed by the client app (see Fig. 4c). The interplay between client app and server API while 'receiving' a gift is outlined in Fig. 5c.

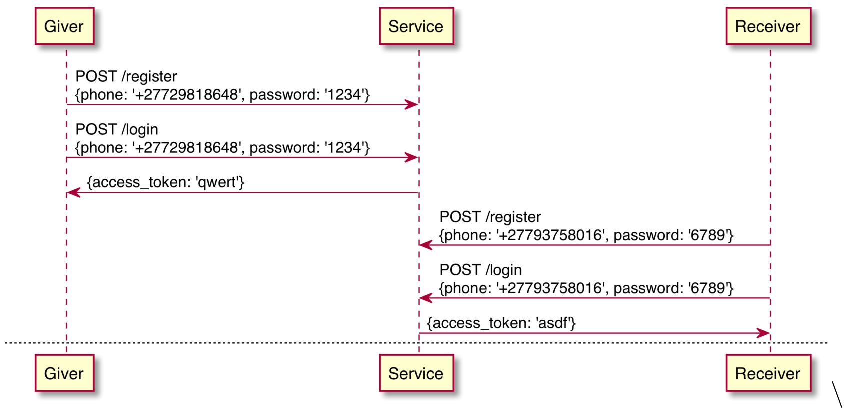
The file, however, now exists in three places: once on the giver's phone; once on the server of the gifting API; and once on the recipients phone. Thus a bit of housekeeping is still required to firstly remove the copy that was created on the server to facilitate the transfer and finally remove the original file on the giver's phone. Only then can the transfer of the gift be regarded as complete. So upon successfully downloading the photo, the client app makes one last request to the API server to indicate that they have 'accepted' (or received) the gift. This request again contains the gift ID as well as the recipient's access token. The server responds by deleting its copy of the photo and sending a push message to the giver, which contains the gift ID and a status that has been marked as accepted. On the giver's side the client app processes this message by deleting the original file and updating its records to show that the photo has been successfully gifted to the recipient (see fig. 3). This housekeeping phase between the respective client apps and the mediating API server is illustrated in fig. 5d.

2.6 DISCUSSION & REFLECTIONS

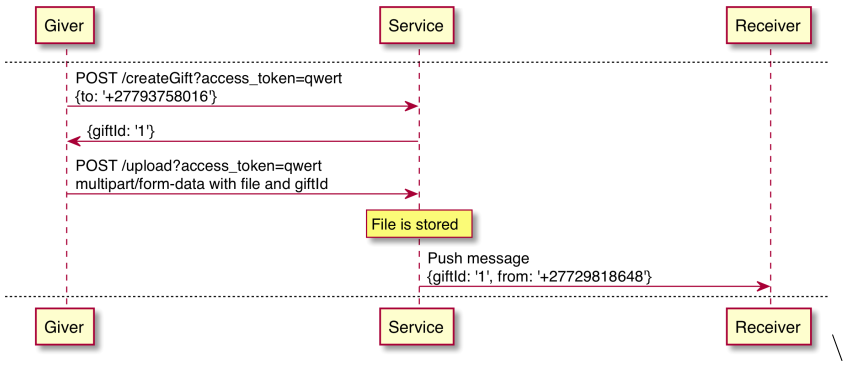
We started this chapter with a deceptively simple question:

> Does Windows Phone 8 support gifting?

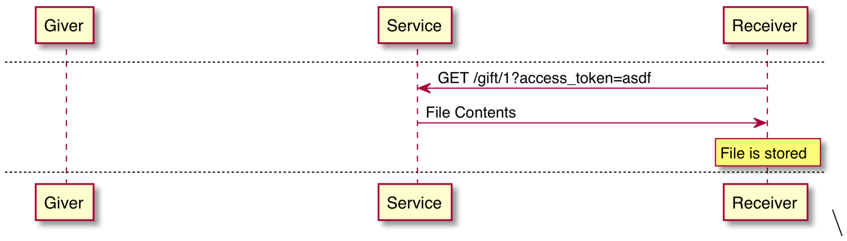
While the answer to that question is *no*, the purpose of asking this question was precisely to provoke reflection and discussion. It is this discussion that we pick up in this section.


Giver
Service
Receiver
POST /register
{phone: '+27729818648', password: '1234'}
POST /login
{phone: '+27729818648', password: '1234'}
{access_token: 'qwert'}
POST /register
{phone: '+27793758016', password: '6789'}
POST /login
{phone: '+27793758016', password: '6789'}
{access_token: 'asdf'}
Giver
Service
Receiver

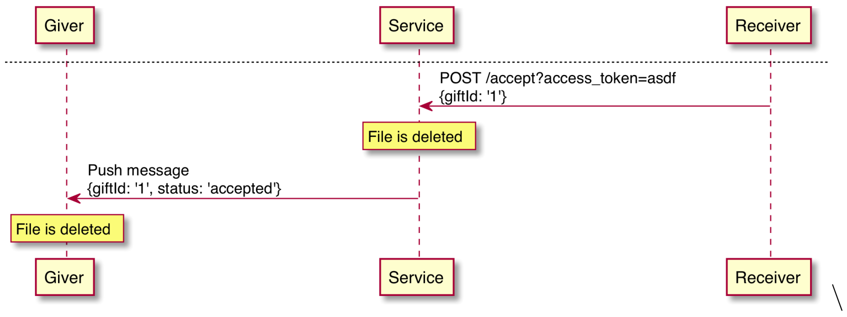
(a) Authenticating users.


Giver
Service
Receiver
POST /createGift?access_token=qwert
{to: '+27793758016'}
{giftId: '1'}
POST /upload?access_token=qwert
multipart/form-data with file and giftId
File is stored
Push message
{giftId: '1', from: '+27729818648'}
Giver
Service
Receiver

(b) Sending a gift.



(c) Receiving a gift.



(d) Deleting transient copy and the original file.

Figure 5: Sequence diagram of the gifting exchange protocol

2.6.1 *Changing architecture*

The computing architecture of contemporary mobile phones, such as those running Android or WP8, is strictly technically speaking no different PCs running Windows or Linux. After all it is possible to run Android on a PC and to run Ubuntu on certain Android phones whose boot-loaders have been unlocked[12]. But if we move away from a strictly technical understanding of the term architecture to think of the term, as Dourish (2014) argues, as a shorthand for "recurrent patterns of arrangements of software, hardware, and representational practice", we can then detect subtle, yet consequential ways in which mobile architectures are shifting. The last part of this definition – representational practices – stands out, for it is the part – as Dourish & Mazmanian (2013) argue – that enables the study of software as a cultural form.

In this chapter we have seen that on WP8 and by and large Android too, we encounter photos as streams of data that can be rendered, shared, or otherwise transformed (e.g. through filters, cropping, or resizing) before rendering or sharing. Even though we mostly encounter them as data streams, we have also caught glimpses of the fact that photos are still stored as files on the phone's internal storage or on a memory card. For instance, the WP8 `PhotoResult` class contains a `OriginalFileName` property or on Android where in certain instances we were able to obtain the file path from a `Uri`. This means that the method of accessing files – file-streaming – hasn't changed compared to the traditional PC architecture. However, its emphasis is shifting. On the PC, we encounter locally stored photos – or any other files for that matter – first as a file that we can subsequently read from or write to using data streams. We might say that the method of accessing files on a PC is *File*-Streaming. But on the mobile, as we've seen in this chapter the emphasis shifts to the latter term: File-*Streaming*. In many cases on both WP8 and Android the file that anchors a data stream has been abstracted away entirely on WP8, as we saw in sec. 2.4.5, and on Android the file could only be recovered using a clever workaround and powerful permissions.

Thus this shift in emphasis has significant consequences to the representational practices that underpin computing architectures. On the mobile this shift means that WP8 does not support gifting! On Android it means that substantial workarounds are required.

2.6.2 *Workarounds*

It is these workarounds that we turn our attention to next. For on the surface, one might be tempted to read them as a triumph of Android over WP8. After all Android supports gifting, while WP8 doesn't. However, before we make such claims, we should unpack the nature of the workarounds we used to make gifting 'work' on Android. Let's begin with mikegr's question on the StackOverflow forum (2013). This time instead of just implementing the proposed solution for getting the file path of a photo Uri, lets look at some of the discussion the question as well as some of the proposed answers provoked. Off the cuff, we can see the reputable and influential user CommonsWare express concern: "I have long been nervous about apps that assume that a [...] `Uri` that represents a file can always be converted into a `File`." This concern was further underlined by the user j__m, this time commenting on the most popular proposed solution to that question – the workaround that we adopted in our gifting app:

> this is absolutely horrible! you should not continue to propagate code that "cheats" like this. it only supports the source apps that you know the pattern for, and the whole point of the document provider model is to support arbitrary sources — j__m on MIKEGR (2013)

12. see https://wiki.ubuntu.com/Touch/Devices

Both j__m and CommonsWare are, of course, correct. They recognise that the Uri handle is opaque by design, and that any apps that attempt to work around this – for instance by querying the _DATA column of the Android `MediaProvider` API – are treading on thin ice. By and large newer versions of Android are backwards compatible and can thus run apps that were compiled against an older version of the Android API. When we ran our app on a newer version of Android 6.0, the UI and navigation from Activity to Activity still worked perfectly. However after successfully gifting a file, the app crashed while trying to delete the original file. Google had changed the permission model, and it was no longer possible to request blanket permissions to read, write, or delete files. So even though we were able to derive the file path 'behind' the `Uri` we could not access it[13].

In another instance, we found out that we were not able to access and thus gift or delete a file if it was stored on a SD card plugged into the phone instead of the phone's internal storage. Likewise a number of third-party apps are making use of the `DocumentsProvider` API, which may or may not adopt the `_DATA` convention for deriving a file path from a `Uri`. This convention, which was adopted before the `MediaProvider` API was superseded by the `DocumentsProvider`, was never officially solidified and is being followed less by the `DocumentsProvider`. The difference in these APIs can be attributed to the influence the Cloud, another dominant computing paradigm of our time.

2.6.3 *Influence of the Cloud*

The point of Android's Storage Access Framework is to unify cloud and local storage services under the `DocumentsProvider` API that encapsulate their services[14]. With the introduction of the `DocumentsProvider` API, files stored on Cloud (storage) services, such as Dropbox, Google Drive, or OneDrive, can be displayed and interacted with alongside files stored on a local datastore. So if we wanted to, for instance, attach an image to an email or message, we could access the files stored on our Dropbox just as easily as a file stored locally. If we did choose a file on Dropbox, it would first need to be downloaded, only to be subsequently uploaded again once the message or email is sent. The `DocumentsProvider` API handles this first step transparently. But it assumes that the user has a sufficiently stable internet connection and can afford the extra data charges[15].

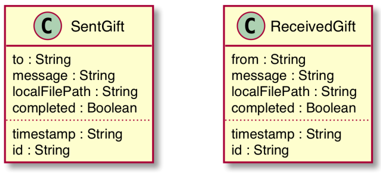
On the surface one might be tempted to see the mobile and the Cloud as separate domains of computing. However, the Android `DocumentsProvider` API illustrates how the cloud is influencing mobile architectures in tangible ways. We already spoke at lengths on how this influence required us to code workarounds. But it also changes how we speak of and refer to data. Whereas before we would speak of files, we now speak of 'openable' content and deal with opaque URIs. This is no matter of mere semantics, as Blanchette alerts us:

> While programmers mostly operate within strictly positivists conceptions of language, computer code creates relationships among multiple symbolic systems, those necessary to move the cogs of the machine, and those necessary for those operations of the machine to be situated within language, and thus, social order — Blanchette (2011, 1045).

13. Eventually we found out that we had to request permission to access that specific file at runtime. However, this might not always be the case and in this instance required us to update the app.

14. on this point see, Developers (2016)

15. This is a big assumption, as in many countries Wifi is not ubiquitous and data packages are not only capped but also expensive.



(a) Database schema for a sent gift.

(b) Database schema for a received gift.

Figure 6: Database schema for gifting meta-data.

Thus any investigation of mobile architectures needs to not only take account of the cloud but also how the cloud changes how we represent and talk about data.

2.6.4 *Files are insufficient*

In this chapter it may appear that we, like many developers on forums such as StackOverflow (e.g. (MIKEGR 2013; ÁLVARO 2016)), lament the loss of the file. To some extent and especially in the context of our gifting app this may be true, but we should also also try to understand why contemporary mobile platforms are moving away from files and local datastores to a more unified, cloud-centric datastores.

Driven in part by the rise of mobile phones, computing has become more social over the past decade. To put it bluntly, files – a fundamental abstraction and unit of engagement in computer systems for users and developers alike (SILBERSCHATZ et al. 2013) – haven't kept pace with this changing landscape. In the context of our gifting app these shortcomings manifest themselves when, in the course of developing the app, we were confronted with the following two, related questions:

1. Where do we store a gift we have just received?
2. Where do we store its meta-data?

Dealing with meta-data

Once a user accepts a gift, our gifting app downloads and displays it, but it also includes two bits of meta-data surrounding the exchange that need to be stored : who is the gift from and what (optional) message did they compose to go along with it (see fig. 4c)?

Especially when dealing with structured and repeating data such as gifting meta-data the standard practice for storing and accessing such data on a mobile app is to use an SQLite embedded database. In our case we defined a simple data schema[16] for which we also created equivalent Java Classes that can be seen in fig. 6.

The main benefit of storing gifting meta-data in such a structured fashion is that we can easily query gifting meta-data that stand as a proxy for the gift itself. To show a specific gift – for instance from George – or to list pending, completed, or all gifts our app can simply query the database. Since the local file path of both pending sent gifts and completed received gifts are stored alongside other gifting meta-data, we can easily display the results of a database query using the established master/detail UI navigation pattern that can be seen in figs. 4a, 4b.

16. a formal specification of how the database should be organised.

Storing received files

The question then becomes where do these local file paths, which we have stored in a SQLite database, point to? Here there are two approaches. We can store the photo in the gifting app's private storage directory or in a public external storage directory, such as the `Pictures/` folder. In this context the terms private and public refer solely to apps. So stuff that is private to an app can only be accessed and acted on by that app, whereas stuff that is public can be accessed and to some extent also acted on by any app, for instance the by the phone's Gallery app.

We chose to store received photos in the public `Pictures/` folder in a sub-folder that we created and called `Gifts/`. We chose this option because of how we understand and came to articulate the act of gifting as a transfer of a possession and we thought that users should be able to act on the gifts they received in any ways that they deem appropriate. For instance, the user might choose to store the file on their computer instead of their phone or to move the file into a different album or to delete the file altogether. But this leaves our gifting app vulnerable to inconsistencies, which is why we can understand why other apps might choose to store the user's stuff in a private storage directory. Once files are placed in public storage, however, an app might lose track of them. In our case this might happen if a user moves a file to a different album or chooses to delete it entirely. The local file path that we store as part of a gift's meta data, would no longer be valid. Keeping a file in public storage also means that when a user looks at the photo using a different app, for instance the photo's built-in gallery app, the meta data surrounding the exchange would no be available to the user.

Using private storage neither users nor other apps can 'meddle' with data stored there. So the situations we encounter above can be avoided, as file paths remain stable in private storage our app can therefore ensure that whenever a user views a gift that the meta data surrounding that gift is always displayed alongside it. It is thus understandable why to avoid inconsistencies, which are hard to manage programmatically and can lead to crashes, and to show users rich views that may include meta data or other content over raw data, many apps would choose the app's private storage over public storage directories. But that control and consistency comes at a cost. That the user's stuff – in our case the gifts they received – would be hidden behind and can thus only be accessed through an app. This phenomena is a manifestation of the inadequacy of both files and the datastores they are stored in.

2.7 CONCLUSION

The goal of this chapter was to look more deeply into the materialities of information as it is manifested on contemporary mobile architectures. We uncovered a shifting emphasis of how data stored on the device is made available to apps through sharing interfaces that prevent apps from obtaining a proper handle of, in our case, a photo stored on the device that the user wants to gift. On the receiving end of this social act we also see the shame shifting materialities of information in effect force apps to re-create photo files in private storage sandboxes in order to ensure continued access to them and to present and bundle these files together with their associated social meta-data. Such meta-data, such as who gifted this photo, is often just as important as the file itself.

This interplay between mobile datastores and their sharing interfaces is problematic and worthy of more sustained investigating – the topic we turn to next as we formalise our research approach.

3 RESEARCH AGENDA

3.1 INTRODUCTION

In the previous chapters we have come to know the subjects of our study more deeply: the mobile phone, the stuff we store on it, and the social practices that surround its use. But the gifting fiction and prototypes we developed have served mostly instrumental purposes so far. That is to show that the mobile and its sharing interface are problematic objects. These prototypes, however, didn't adequately address the deeper problems they helped us to identify. In fact, it is typical of Research through Design (RtD) projects to co-evolve research problems and solutions. At this point in our inquiry it seems as if our understanding of the problem has outpaced our ability to address it. In this chapter we pause and reflect on the work we've done so far with the goal of developing and re-articulating our research objectives.

3.2 SHARE: SOLUTION OR PROBLEM?

We have a whole range of verbs that describe how people relate to other people through their actions. We show and tell, give and gift, take and trade, borrow and steal. In the previous chapters we have seen that if we look at any mobile interface the only word we seem to know is 'share'.

Looking at the history of 'share' in mobile interface design it is not hard to see that it has become an entrenched concept that affects all aspects of system design and use, ranging from how we specify feature requirements (see for instance, Reitmaier et al. 2012) all the way to how we market resulting products. Consider how Snapchat, a relatively new social network that has taken the world by storm, advertises its service:

> Experience a totally new way to share today. Snap a photo or a video, add a caption, and send it to a friend (or maybe a few). They'll view it, laugh, and then the snap disappears from the screen.
>
> –"Snapchat" (2014)

While we can certainly say that Snapchat has come up with a novel way of sharing within the context of current mobile platforms. But if we use a verb other than 'share' to describe the service, it suddenly doesn't seem novel at all.

> Snapchat is a service that allows its users *to show* photos or videos to their friends.

By casting all communication acts that mobile phone support or could support in terms of a 'share' operation, we restrict ourselves and create mismatches between computing architectures and human practices. This is more than quibbling over semantics. Language, especially the language we use within design, has extraordinary powers, and "is one of our primary sources to knowledge and insights" (Hansen 2014, p. 22). While we all already know it, I will repeat it here nevertheless: words matter. We can see that they matter by how we care about which words we choose and how we use them in our everyday use of language. So why do we restrict ourselves to one word in the mobile sharing interface, instead of drawing on a richer spectrum?

Fuelled by 'share' and a combination of ubiquitous content, content creation devices, and connectivity, we are living in a world furnished with digital objects.

We stand on familiar ground when we consider such objects – be they digital or physical – as useful or aesthetic, however as Turkle alerts us, we stand on less familiar ground when we consider them as "companions to our emotional lives" (2007, 5). As such companions the digital object we access, keep, and share through our mobile devices are in many ways becoming the 'stuff' of sociality. In the next chapter we will also see how mobile phones are no longer 'just' telephones – technical means for synchronous voice communication at a distance – but a resource to make and engage with digital objects with people near and far. In a nutshell 'share' – the verb, button, programming interface, infrastructure or whatever you want to call it – has enabled millions of different apps to access content stored on the mobile and share it with other people, apps, services, or devices. But does share actually mean what we intend it to? How does 'share' relate to the other verbs we outlined above? If it is operationalised as a copy operation on contemporary mobile platforms, how well does this systems architecture perspective match people's practices, and perhaps more importantly, how does it match people's values as articulated through their practices?

3.3 WHAT CONSTITUTES AN ANSWER?

We began our inquiry with a simple question "does Windows Phone 8 support gifting?". Especially in the context of our gifting fiction we have seen this deceptively simple question unravel into a myriad of not only technical questions but moral ones as well. Thus a critical aspect of re-articulating our research agenda is to consider what constitutes an answer to our original questions. For instance, what does it take for 'share' to mean 'gift' or 'show' and what happens to the object in the process?

3.3.1 *Implicating mobile system architecture*

If we return to the Snapchat example, we can see that they have included a disclaimer on their website:

> Please note: even though snaps are deleted from our servers after they are viewed, we cannot prevent the recipient(s) from capturing and saving the message by taking a screenshot or using an image capture device. —"Snapchat" (2014)

This disclaimer illustrates that for some acts of sharing, such as when showing a photo, the system architecture should be implicated in the act. For instance, on the recipient's phone the screenshot functionality could be disabled.

3.3.2 *Implicating the app model & data store*

In the previous chapters we have already seen the muddle that ensues when we equate communication *acts* with sandboxed communication *apps*. Behind an icon (on iOS & Android) or tile (on Windows Phone) is not only an app but also a sandbox that contains the user's data. Within such a sandboxed app model, data is hidden behind or inside the app, and data abstractions become unclear. New abstractions are therefore needed that foreground the user's stuff and respond to the fact sharing is a fundamental operation on that stuff, which may even transform it in the process.

3.3.3 *Implicating the cloud*

It is almost impossible to talk about the mobile, especially in the context of digital photography, without also talking about the cloud. To be sure, we might explicitly use cloud services to build or backup a collection of photos or show them off to our friends on social networking sites. But at other times we might consider the photos we take and share as ephemera, for instance a silly selfie we send to a friend to make them laugh. Despite these intentions, such ephemera have a tendency to live on. As the following quote illustrates, we need to carefully consider how the cloud is implicated in an act of sharing.

> In the networked reality of people's everyday life, the default mode of personal photography becomes 'sharing'. Few people realise that sharing experience by means of exchanging digital images almost by definition implies *distributed storage*: personal 'live' pictures distributed via the internet may remain there for life, turning up in unforeseen contexts, reframed and repurposed. –van Dijck (2008, 68, original emphasis)

The salient question then becomes: if storage is de-facto distributed, how do we retain *awareness* and *control* over our stuff?

3.3.4 *Implicating data exchange protocols & access points*

Keeping ephemera as ephemera is a complicated task when our stuff moves through the cloud. Part of this task involves engineering data exchange protocols that better respond to the intentions and motivations behind an act of sharing so that a showing operation does not transfer ownership or when gifting, as we saw in chapter 2, the object is removed after transfer. In a way this point goes hand in hand with how the cloud is implicated in acts of sharing.

The paradox is that the reach of the cloud computing paradigm is such that it also affects the functionality of wireless access points, whose purpose is no longer to interconnect devices but to connect them to the internet and the cloud. For instance, it is not possible to create a wireless access point to share files locally on current mobile platforms without also sharing your 3G internet connection. Many wifi hotspots prohibit peer-to-peer connections between devices connected to the same access point.

The mobile is more than a mere portal into the cloud and its sharing interface as well as how we engage socially with content stored on the device shouldn't have to depend on cloud services. We should, therefore, be mindful of how exchange protocols and access points function for people unwilling or unable to afford the cost of using cloud services or in situations when these are unavailable. Such efforts would, in effect, invert the metaphor of the cloud by creating hyper-localised, ad-hoc instantiations of the cloud that still provide similar, social services: a *cloudlet*, if you will.

3.4 A WAY FORWARD

The challenge for us becomes to design a system that gives people better *awareness* and *control* over the stuff that matters to them. Given the various components we have implicated in this undertaking, such a system needs to first carefully consider current relationships between the mobile OS, its datastore, its sharing interface, and the cloud with the subsequent goal of re-imagining that relationship. First and foremost, this is a substantial technical undertaking and will require us to continue to dive into the technical realities of mobile architectures and cloud infrastructures. From personal experience, for instance when engineering to gift, I can say that these realities are best understood when working with the materials at hand: that is, through programming – an activity that deserves some unpacking.

Downey (1998), a self-proclaimed anthropologist sitting among computer engineers, describes this activity as 'boundary blurring'. From an objective point of view, we might say that the programmer and the machine they are programming are two distinct and separate entities. However if we reflect on the *act* of programming, and this is the point that Downey (1998) is making, we also extend our agency into the machine. While we certainly never articulated it so eloquently, this boundary blurring resonates with my direct experience of programming. However as Phil Agre (1995, 73) points out for programmers immersed in the closed worlds of system development, "it becomes difficult to imagine the perspective of somebody who does not view a computer system as a logical anatomy, an ontology made of datastructures, a set of formal relationships and constraints, and a network of paths for data to move along. Since the programmer is imaginatively inside the system, the very concept of a user interface can be difficult to grasp or take seriously" (cited in Suchman 2007, 188).

Taking these perspectives into account technical expertise cannot be both necessary and sufficient condition. We therefore also need to develop our understandings of how users interact with and come to understand the stuff that they keep and share on their mobile phones and in the cloud and let that understanding guide our inquiry. This is the pattern of inquiry – to bring technically and socially complex, multi-dimensional and changing ideas into *correspondence*[1] – that we have implicitly been following thus far and that we formalise at this point. It is an approach, as we saw in the introduction to this dissertation that fits with a material perspective and the Research through Design approach.

3.4.1 *Research through Design Process*

A recurring theme within Research through Design is that documenting the process is a crucial activity and yet, judging by the repeated clarion calls advocating for that practice, is probably also one that gets neglected. Documenting not only the Research through Design process as a whole but also keeping an account of the ever changing properties of a design-in-the-making is a key method for elucidating some more epistemic aspects of designs (Dalsgaard 2016). Others aspects and forms of knowledge are encoded into the object itself. Here Bardzell et al. (2015) note that Research through Design should be seen as "a *thing-making practice* whose objects can offer a critique of the present and reveal alternative futures, while remaining grounded in empirical science, behavioral theory, contemporary technological possibility, and socio-cultural practices" (Bardzell et al. 2015, 2095). The active role of the artefact is emphasised by Stolterman & Wiberg, who note that artefacts 'co-produce' knowledge and "manifests desired theoretical ideas as a compositional whole" (2010, 109). A concept that is also expressed by Zimmerman et al, who extend this line of reasoning: "the prototypes themselves were experiments with material and technology, codifications of understanding about users and contexts, and sketches of potential futures" (2010, 315). This resonates with the perspectives that a prototypes functions as a "reflexive probe into the practical materializations that configure new technological objects" (Suchman et al. 2002, 175).Thus, Research through Design sees the act of making as a route to discovery in itself, where prototyping becomes a site of inquiry. Other design research programs, such as Koskinen et al's *Constructive Design Research*, likewise centralise the thing that is constructed, which in turn "becomes the key means in constructing knowledge" (Koskinen et al. 2011, 5).

As computer software is the material of our Research through Design approach, it is the encounters we have with that software that we document and present in this dissertation: namely, the interfaces (both UIs and APIs) between our ubiquitous computing infrastructures and architectures. Here Dourish & Bell (2011) remind us that when we try to understand ubicomp we can't just do it technically but also socially, culturally, and historically. A view that Ingold

1. On this point see Tim Ingold's (2013) metaphysics of creativity and design

extends to any type of material, for "the properties of materials, in short, are not attributes but histories" (Ingold 2011, 32). What better place to start then with one of the most mundane aspects of computer system design: the humble file.

3.5 WHAT IS A FILE & WHAT SHOULD IT BE?

Files are, to put it bluntly, archaic and unremarkable. In fact, we only really noticed the important role they play in system architecture when they go missing, for instance on Windows Phone 8, or rendered predominantly invisible, as they are on Android. But what exactly are we missing? To understand what a file *is* technically and the role it plays in computer architecture, we might turn to a foundational book on the topic, Silberschatz et al's 'Operating Systems Concept', where we are confronted with multiple definitions and characterisations:

1. A file is a collection of related information defined by its creator (Silberschatz et al. 2013, 453).
2. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the *file* (Silberschatz et al. 2013, 455)
3. For most users, the file system is the most visible aspect of an operating system (Silberschatz et al. 2011, 455).

Implicit in these characterisations is the mediating role the operating system plays between file 'creators' (computer scientists/engineers) and its 'users'. This is a topic that Harper et al. (2013) explore in great depth in pursuing a line of reasoning that – contrary to the title of their paper "What *is* a File?" – explores what a 'file' *does* in practice, namely to bind together the concerns of users and computer scientists/engineers. A file then is more than just a "collection of related information" but also "a label for a space of organised collaboration" between users and engineers (Harper et al. 2013, 1126), what Star (2010) refers to as a 'boundary object'. Through this organised collaboration engineers can worry about, say, the efficiency and reliability of a storage system and users can edit the newsletter for their football club or look at the photo of the sunset they took last week. This is a remarkable achievement and illustrates "the talent of the designers of early systems [which] 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" (Harper et al. 2013, 1129).

3.5.1 *Icons & Grammars of Action*

The cunningness[2] of early designers is further shown in how files are re-presented on the PC namely using icons and how they can be acted on. Icons are representations of things – files or computer programs – that give that thing a sense of place – on the desktop or in the documents folder. When a file is moved to another folder, its icon moves with it. When it gets deleted, its icon disappears. In this design there is a functional coherence that is built on deceit. The file icon reinforces the notion that a digital file is like a physical file – a bounded entity of sorts – but obscures the fact that the file might be fragmented across the hard disk it is stored on. Dragging a file icon into a different parent folder reinforces the notion that its the file that is moved, when in fact it is the full path of that file that is renamed, which has the desired effect of changing its position in the directory structure (see Silberschatz et al. 2013, 518). Actions such as moving and deleting files are as old as the PC itself and form part of the definitional

2. Flusser argues that all design requires an element of deceit and trickery to "seduce people people into perceiving distorted ideas" (1999, 26).

grammar of action the landmark Xerox Star, the first desktop computer with a graphical user interface.

> The [Xerox] Star has a few commands that can be used throughout ... They strip away extraneous application-specific semantics to get at the underlying principles. Star's generic commands are derived from fundamental computer science concepts" – Smith et al. (1982, 525)

Lets look at the move action again to better understand how these generic commands were intended to work.

> MOVE also reinforces Star's physical metaphor: a moved object can be in only one place at one time. Most computer file transfer programs only make copies; they leave the originals behind. Although this is an admirable attempt to keep information from accidentally getting lost, an unfortunate side effect is that sometimes you lose track of where the most recent information is, since there are multiple copies floating around. MOVE lets you model the way you manipulate information in the real world, should you wish to. We expect that during the creation of information, people will primarily use MOVE; during the dissemination of information, people will make extensive use of COPY. (SMITH et al. 1982, 526)

The goal of commands like move, copy, and delete, which Harper et al. (2013) refer to as the grammar of action, is to let users excerpt direct control over their stuff, as illustrated in the above discussion of the move command, while just as importantly retaining awareness of where that stuff is.

3.6 OUTLOOK – MOVING TO THE MOBILE AND NETWORKED COMPUTING LANDSCAPE OF TODAY

At least so it was on the PC. But, the tools and interfaces that people use today to manage and share their digital stuff has not kept pace with the changing nature of that stuff, the practices of users, and the exponential advances in devices, storage, and networking technologies that shape the digital landscape we inhabit. In fact, the grammar of action encapsulated in our contemporary toolsets and interfaces are built upon the definitional grammar of the first PCs of over 30 years ago. This limited toolset of the past no longer encompass new possibilities of the mobile and the cloud and ultimately contribute to mismatches between what people can do and what they want to do with their digital stuff.

Despite these shortcomings, Harper et al. (2013) argue that we shouldn't abandon the concept of the file altogether, precisely because for over 30 years it has been such an effective tool to bind together the concerns of both users and engineers and through the ways it gives users awareness and direct control over the stuff that matters to them.

What is instead needed is to redefine what a file is by closely examining the constraints and possibilities of its current grammar of action with an eye on defining "a new *grammar of action* for an HCI of the 21st century" (HARPER et al. 2013, 1126). We ally ourselves to, and extend this line of research, by leveraging the mobile phone in particular and the cloud services with which mobiles interface more generally to interrogate current and reimagine future grammars of action by attending to the rich social practices that surround and give meaning to the use of mobile phones and cloud services.

The mobile highlights that we need to have a social view of files. It are precisely these social practices that we turn to next.

4 COLLECTING & SHOWING

4.1 INTRODUCTION

The word telephone is comprised of two parts, both derived from Ancient Greek: *tele* (long distance) and *phone* (speaking). The popularity of messaging apps on contemporary smart phones amply demonstrate that nowadays we do much more than speaking on our phones. So perhaps we should refer to them instead as telecommunication devices, because the legacy of the first part of the term *tele*phone – implying interaction at a distance – remains an intrinsic assumption of mobile interface design. In this chapter we study and challenge this assumption by giving an interwoven account of the theoretical and practical work we undertook in pursuit of deriving and designing a grammar of action for co-present interactions.

Especially when making communication or sharing apps, as we did in chapter 2, it is easy to focus too much on the mechanics of the exchange, and therefore to "overlook the humans who are doing the communicating" (Harper 2010, 49). When we focus on the mechanics, we think of communication acts as messages that need to be transferred from one person (or device) to another. To be sure, this model has proven generative potential for mobile design. Sometimes communication is about the message, such as when we send a text to a loved one to tell them: "I'm on my way home." But the deeper problem, as Harper alerts us, is that "the vision used to orient design is of a world that is not the same as the one real people populate" (Harper 2010, 240). Instead, we need to think beyond simple messages and figure "communication between people [as] a performance that ties people together (or throws them apart) in various ways" (Harper 2010, 247). Seeing communication as a performance highlights that it is not just about *what* we say, but *how*, and *where* we say it. These different characterisations of communication, as simple messages or as encompassing performances respectively, help us understand why mobile design (research) tend to focus on *tele*communication with absent others. But before we examine existing and build new systems that explore co-present interactions, we should engage with the deeper issue that Harper (2010) identifies. Namely, that we need better ways to think about co-present interactions in the first place.

To advance this goal we first show how we sensitised ourselves to theories on proxemics, context, identity, and embodiment to obtain a more nuanced understanding of performative aspects of communication and co-present interactions. We explain how we used these theories as a critical lens to examine and critique current systems that support co-present interactions. But far from just critiquing, we also show how we applied and refined this lens to generate and map out a new set of co-present interactions on mobile devices. We then explain how we integrated insights, sensitivities, and critical stances, derived from theory, to explore these through the process of design, culminating in the design and development of two prototypes that express and explore key concepts. We present these aspects of our work in four sections, sensitising, critiquing, integrating & generating, and exploring, before we reflect process and draw some implications for design. In choosing this structure, we depart from conventional practice, where related work, usually discussing theory and related systems, is presented as an upfront baseline that is often bracketed out from the 'actual' research. Here we show how theory, related systems and even our own previous work were central throughout: a constant, yet productive, site of struggle. Our theoretical understandings, drawn from a variety of different intellectual disciplines, emerged as we engaged with them at the different stages

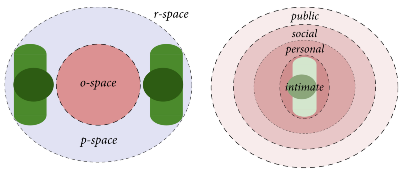


Figure 7: Kendon's F-formations & Hall's proxemic zones.

of our research. So beyond generating and exploring co-present interactions, it is this story that we tell.

4.2 SENSITISING

Focusing on performative aspects places the importance of communication at the heart of the human condition, and destabilises the assumption that the message of a communication act is somehow more important than how it is bound to context, time, and identity. This broader view opens up a path for research that looks at how people might want to produce and engage with the media stored on their phones when they are co-located. Here, we turn to theory to sensitise ourselves to salient concepts such as context, time, and identity to understand how we might design for more meaningful co-present interactions, as well as how we shouldn't.

4.2.1 *Understanding co-present practices*

A major conceptual lens used to describe and analyse co-present situations is Edward Hall's theory of proxemics (1966). Hall has coined the term proxemics for people's use and perception – through eyes, ears, nose, and skin – of space. Many of Hall's observations focus on how fixed-feature spaces (e.g. architecture) interact with semifixed-feature spaces (e.g. furniture), and how this affects people, and how they interact with one another. While our experience of space certainly depends on the interplay of fixed- and semifixed-feature spaces, for Hall the most significant category of spatial experience is a mostly unstated, informal space: the distances maintained in encounters with others. He proposes that a person does not end at his or her skin, but is surrounded by a series of expanding and contracting fields: a space, whose size and properties vary on account of culture (more generally) and personal relationships. These can be classified into four, discrete proxemic zones of Fig. 7: intimate, personal, social, and public. While many of his observations seem common-sense, they relate to a behaviour – our use and perception of interpersonal space – that largely lies outside of our awareness; we take it for granted. Hall uncovers these through his descriptions of causes and effects, highlighting how these are interpreted. For instance, people who are angry will move in close to make their point, just as people who are amorous will move in close to express affection. And it's not just touch, but also sensing the heat of another person that combine as we feel intimacy. But the very same sensations (touch & heat) can make you feel claustrophobic in a packed train, so we keep our muscles taught to maintain our space.

Researchers, such as Greenberg et al. (2011), who have grounded their work in Hall's proxemic theory and terminology acknowledge – as we also suspected in our readings – that Hall's theory is at best suggestive to design and that "we just don't understand the HCI of proxemics" (Greenberg et al. 2011, 50). But we can find inspiration in Hall's approach that reflexively integrates observations. Harper advocates for a similarly reflexive approach to understand everyday actions, such as a son brewing his mother a cup of tea to be ready for her when she gets home. We don't need theories to explain the significance of such acts:

"we [need] to use the expertise about the world that we [gain] by living in that world" (Harper 2010, 194).

4.2.2 *Understanding boundaries of people & time*

If we only focus on certain aspects of communication, such significances can easily be lost. Much research to date has focused on mediating the 'bodily mechanics' of communication through computers, such as capturing and conveying gestures or glances (Harper 2010, 10). While, bodily mechanics are important, focusing solely on them creates a boundary around individuals. Hutchins argues that it is precisely these boundaries that need to be softened and advocates that "the proper unit of analysis is [...] not bounded by the skin or the skull. It includes the socio-material environment of the person" (1995, 292). Similar arguments apply to time. Experience, as Hutchins notes in a later article, "is not only multi-modal, but also multitemporal or temporally extended in the sense that it is shaped both by memories of the past (on a variety of time scales ranging from milliseconds to years) and by anticipation of the future (over a similar set of time scales) (2010, 432). We integrate our memories of the past, experience of the present, and anticipation of the future when we engage with others (Suchman 2007) and with artefacts of our world (Hutchins 2010). Rigid boundaries of time or bodies cut through lines of interaction and obscure relevant phenomena, such as the significance of a cup of tea brewed in anticipation of mother's arrival.

4.2.3 *Understanding identity*

If we soften boundaries of bodies and time, then important implications follow for identity that contrast with how identity is approached by common ontologies of computer science. Often these start with an instance or individual – an anatomical, and bounded unit – and associate a unique identity to it, to discriminate one instance from another. They further group instances together to form sets, classes, collections, etc. based on common attributes. To be sure, this is a powerful model at the heart of many computing systems and databases in particular. It is, however, also one that views identity rigidly. Is this show we should be thinking about the identity of a person? While identity is something unique to us, it also "implies a relationship with a broader collective or social group of some kind" (Buckingham 2008, 1). In softening borders, the question then becomes how does identity interact with the broader collective over time? Buckingham argues against rigid views because: "who I am (or who I think I am) varies according to who I am with, the social situations in which I find myself, and the motivations I may have at the time" (2008, 1). In discussing different disciplinary orientations towards identity, he alerts us that identity is not something we posses, or something we are, but is something we do. It is something that comes into being in dialogue between self and other.

One of the more prominent theorists on identity is Erwin Goffman, who sees identity as something that is performed (1959). Goffman approached identity through the metaphor of theatre to describe how individuals use their physical and social surroundings to present themselves to the world and, in turn, how the 'world' interprets their 'performance'. He noted that the relationship between the performer and the audience is one of impression management; through their (inter)actions performers project themselves, whether intentionally or unintentionally, to an audience that interprets their (inter)actions. So both parties are involved in negotiating a 'performance'. Goffman also classified two types of performance 'regions' that a performer has access to and that are used to maintain their impressions, namely the front-stage and the back-stage. The front stage is an attempt by the performer to give the appearance that their performance is their de facto standard. Here, clothing, posture, speech, gesture, and expressions, can also affect performances. In turn, the backstage is the region where the performer can openly contradict front-stage performances, drop their front and "step out of character" (Goffman 1959, 70). While Goffman is not without critics (e.g., Buckingham 2008), he, like Hall, was a perceptive observer of social interactions. He elevated the world of social interactions from the obscurity of plain sight, by giving us a vocabulary to talk about and observe it.

4.2.4 *Understanding context*

The discussions so far show that co-present interactions, like communication acts in general, are highly contextual. But as Dourish alerts us, it is a context of a particular nature (2004). Especially face-to-face, context is an interactional, dynamic, occasional property that arises from activity. This characterisation departs from how we usually delineate context, as something that is stable and can be measured and encoded without reference to the activity at hand. Within any dialogue what is and what isn't contextually relevant cannot be established a-priori; a comment can trigger a memory and lead the dialogue down a different path. So something that wasn't contextually relevant before, now is. In fact, negotiating context is a very ordinary achievement (Dourish 2004); we do it almost automatically, often without noticing. This interactional view of context suggests inquiries and designs that focus not on how to re-present context, but rather on flexibly accommodating changing contexts.

Harper argues that to sensitise ourselves to the performative nature of communication acts, then we must see these along three dimensions: "the how of the act itself (the bodily skills involved), the who of the act (where one needs to be alert to the intentions of the actor themselves and how the undertaking of some act conveys a sense of identity or self for that person and to the audience or recipient of the act), and third, the where of the act (the location of its performance)" (Harper 2010, 245). Or, in short, context, communication, and identity are enmeshed in performance.

4.2.5 *Understanding photographic (co-present) practices*

To show how this enmeshing works in practice, we identified (mobile) photography as a salient practice in which the theories we have discussed take hold in a way that also relates them to (digital) media. Photos usually depict unique, memorable, happy, events and rarely the routine, sad, or ordinary (Shove 2007). In a sense they re-present an idealised self (Goffman 1959). Photos with family or friends contribute to a sense of belonging and of self. Related practices of photo viewing and sharing re-produce and mediate deeply rooted social practices such as gift giving (Mauss [1950] 2000) and storytelling (Van House 2009).

Kindberg et al examined how 34 people in the US and UK use their camera phone, their intentions at the time of capture, and subsequent usage patterns (2005a). The authors distinguish these intentions along two dimensions: affective-functional; and social-individual, where social use is further broken down into co-present viewing and absent sending/viewing. Their study shows affective photos outnumbering functional, and social outnumbering individual photos. Most image sharing happened in the moment and on the phone's screen, and rarely via Bluetooth or MMS. Results also show that post hoc sharing didn't happen nearly as often, because "the time and effort one must put into sending these 'gifts' [is] difficult to achieve in the moment" (Kindberg et al. 2005a). Some participants simply 'hadn't gotten around to it yet,' forgot or lost the impulse to share. When post hoc sharing did happen, it often involved storytelling.

Chalfen further analyses how people communicate and tell stories with photos (1987). He contends that photographers are reluctant to create self-containing visual narratives. The narrative remains in the head of the photographer: "a picture may be 'worth a 1000 words' … [but] pictures don't literally 'say' anything – people do the talking" (1987, 70). This does not mean that photos always need to be accompanied by a narrative especially if shared with close-knit, yet absent friends (Kindberg et al. 2005a). But it does illustrate why post hoc, co-present sharing is a practice of enduring importance (Van House 2009). Photos, especially those we cherish, re-present memories, usually of "moments of positively valued change, marked by parties, official recognition, or public celebration" (Chalfen 1987, 96). In this sense, photos link to identity, group belonging, and presentation of self. The stories we tell around photos do more than contextualise a photo but also contribute to our biographical 'narratives' – the stories that explain ourselves to ourselves (Buckingham 2008). Face-to-face sharing lets people adapt their presentation of photos, and with it their presentation of self, to their social surroundings. Such face-to-face sharing is a "dynamic, improvisational construction of a contingent situated interaction between story-teller and audience" (Van House 2009, 1073). This explains why applications, explicitly designed for telecommunication of photos (e.g. Naaman et al. 2008) still find their use in co-present situations.

4.3 CRITIQUING

In the last section we looked at co-present practices and developed a performative lens of co-present interactions by sensitising ourselves to concepts of identity, context, and time. This lens destabilises simpler, yet prevalent, views of interacting humans. Here, we show how we applied and refined our sensitivities by critiquing related work through this lens. We identify two examples of related work that are representative of how co-present or proxemic interactions are commonly conceptualised. We intend for discussions to be productive, a chance for us to contextualise and interrelate what we learned and to identify opportunities for design. But we also recognise that criticism, an embraced part of more mature design disciplines, can be perceived as negative and unhelpful, so to set the tone for a constructive discourse, we start by critiquing our own previous work.

4.3.1 *Mobile Digital Stories*

In previous, collaborative work we designed a mobile digital storytelling system to suit the needs and functions of rural African communities. Informed by ethnography and technology experiments involving storytelling, implemented a design workshop to involve users in a rural community in South Africa's Eastern Cape in the design of a mobile digital storytelling system. Using this method, we created a mobile digital storytelling prototype to suit the needs of rural users. Details of our design process and the resulting prototype, that can flexibly accommodate different digital storytelling techniques, have been published elsewhere (Reitmaier et al. 2011). For our critique we focus on how I[1] field-tested the prototype in a rural community in Kenya. During this formative evaluation, I didn't consider performative aspects of storytelling and focused instead on how people created stories – how they took photos, recorded audio, and stitched them together into digital stories – on our prototype. While I also gained more diverse impressions of the system-in-use and recorded these in my field notes and pictures, my unit of analysis during storytelling activities was centred largely on the prototype and the person interacting with it. It was only after sensitising myself to theories of distributed cognition (Hutchins 1995) and situated action (Suchman 2007) that I understood the lines of interaction that were

1. I switch to the first person voice here to take ownership of this critique.

being cut. People did not merely create, nor tell, digital stories, they performed them. One storyteller paid little attention to the prototype when recording her story and instead looked deep into my eyes. Users tailored performances to specific (co-present) audiences; they engaged with and drew on their physical and social surroundings, often in co-present and collaborative creativity. While I was able to reflect on these activities with more appropriate analytical lenses, I only begun to understand the meanings users created well after the fact (Reitmaier 2011). This illustrates how narrow disciplinary orientations – in my case on usability – obscure important aspects of co-present interactions on mobile devices/applications, especially how these are constituted in and inseparable from physical and social contexts of use.

4.3.2 *Mobiphos*

Mobiphos is a novel interface that supports photo capture and automatic co-present, synchronous sharing within a predefined group (Clawson et al. 2008). Reading that paper, we suspected that privacy would be a major issue; but to our and the researchers surprise it was not. Because photo-capture is automatically linked to co-present sharing, the consequences of people's actions are apparent – all members of the group will be able to see all the photos you take. So, people adapted their photo taking behaviour to take this into account. Users created meanings by matching the possibilities of the technology to their ongoing goals, on the fly; instead of worrying about privacy they adapted their photo taking behaviours and at different times collaborated or took mischievous and funny photos for other to see.

While Mobiphos can provide an enchanting experience, it works better for certain genres of photography, such as tourist photography (Chalfen 1987). This is also the scenario in which Mobiphos was deployed. Likewise, the tourist is an established social role (Goffman 1959), that carries with it conventions and norms that also relate to photo taking, for instance that photos tend to be of places and objects. The design and deployment of Mobiphos also sets an explicit timescale that is limited to the duration of the co-present 'camera recreation.' We are left to wonder what happens before and what might happen after?

4.3.3 *Pass-them-around*

We also find inspiration in Lucero et al's collaborative, co-present, and multi-screen photo sharing experience called *pass-them-around* (Lucero et al. 2011). By pairing multiple phones together, Lucero et al are able to leverage the metaphor of passing around paper photos, whereby a photo is passed from one phone to the next. In their prototype users can either share a collection of photos sequentially or share individual photos one at a time. However, the emphasis of that particular paper is on the multi-screen interaction technique and the spatial arrangements of these. It does not explicitly consider which photos, out of potentially thousands, users might want to share.

4.3.4 *Proxemic Interactions*

Greenberg et. al recognise the importance of spatial relationship in proposing and formulating an admittedly speculative vision of ubicomp, namely proxemic interactions: "just as people expect increasing engagement and intimacy as they approach others, so should they naturally expect increasing connectivity and interaction possibilities as they bring their devices in close proximity to one another and to other things in the ecology" (Greenberg et al. 2011, 44). Proxemic interactions are triggered by sensing relationships between people and digital as well as non-digital objects. These relationships are characterised and measured along five dimensions: distance, orientation, movement, identity, and location. In developing different scenarios and applications, they show how Hall's (1966) proxemic zones (see Figure 7) can: "regulate implicit and explicit interaction; trigger such interactions by continuous movement or by movement of people and devices in and out of discrete proxemic regions[2]; mediate simultaneous interaction of multiple people; and interpret and exploit people's directed attention to other people and objects" (Ballendat et al. 2010, 121). The researchers have implemented a number of prototypes, making use of 'fine-grained knowledge' of these dimensions by tracking them through a motion capture system. The resulting prototypes are rule-based systems. The proxemic presenter, for instance, augments traditional presentation tools, and shows presenter notes when the person presenting turns towards the display. The proxemic media player pauses when a person starts to read a book. It can interpret pointing gestures to allow users to select different media items, and displays the movie title when another person enters the room. These applications are intuitively appealing, and show that Greenberg et al. are correct that devices should react to proxemics. But we are concerned that such information is encoded in a representational view of context (Dourish 2004). Likewise, Goffman teaches us that people behave differently in relation to different contexts. We are concerned that such proxemic interactions are operationalised along too ridged and narrow a definition of identity that "uniquely describes the entity" (Greenberg et al. 2011, 44). What might proxemic or co-present interactions look like, that are built upon a performative account of identity and an interactional understanding of context? And what would they look like, not in the conference or living room of the future, but on the presently ubiquitous mobile phone.

4.4 INTEGRATING & GENERATING

The theories we considered so far have largely been formulated prior to, or independent of, a device that is shaping the communication landscape around us: the mobile phone. In this section we revisit the theory we outlined earlier to integrate and relate these to how people (might want to) engage with each other and their mobile devices when co-present. In the process, we generate new design spaces and suggest under explored phenomena and avenues for research.

A key aspect of our performative lens is Goffman's theory on the presentation of self. It has been widely applied to analysing how we present ourselves on social media outlets, such as Facebook and MySpace, where we write ourselves into being (boyd 2008), using text, but also media. In everyday co-present interactions, however, our bodies, not our profiles, are the focal point of that performance. We use gesture, speech, and facial expressions; augment them with clothing styles, in order to project who we are (Goffman 1959). Combining these perspectives, leads us to question what role media plays in our co-present 'performances'? To do so, we first need to understand how media links to performative aspects of identity? Perhaps a better way of understanding identity is when we are stripped of it. In Asylums, Goffman describes how this can happen when mental patients are institutionalised (Goffman 1961). Names are replaced by numbers; our clothing with a uniform; our hair gets shaved off; our pockets are emptied; and the small paraphernalia that we carry on our person – wallet, often containing pictures, briefcase, purse, handbag, book, etc. – are also removed. These items form a person's 'identity kit', crucial items for the management of a personal front (Goffman 1961). For some, the mobile phone, as a physical object, is an intimate part of their identity kit: the brand, the colour, and how it is accessorised. But the mobile is more than a physical device; it is a repository of information and histories (Ling 2008). So beyond physical devices, how do mobiles figure into our identity kits? What does the stuff on the device – the playlists, gallery, call

2. These correspond to Hall's (1966) zones.

and SMS log, etc – say about us? What role do these personal repositories play in how we present ourselves face-to-face?

To answer these questions we revisit the front and backstage regions of Goffman's concepts of impression management. Looking at mobile phones and the applications we use on them, we might say that they are front stage devices, as they provide "insights into our tastes, our style of consumption, and perhaps our allegiance to certain groups" (Ling 2008, 96). But we might also say the mobile is a back stage device, as it has "evolved into a significant repository of personal information" (Ling 2008, 97), often containing sensitive information which might, to use Goffman's terminology, contradict the performer's front. But it isn't a purely back stage device either, as we draw upon this repository on applications, like Facebook, Flickr, messaging, and email, to present aspects of ourselves to the people we communicate with, enabling and sustaining a front stage performance. This dual characterisation of the device also explains, why when co-present, people are reluctant to let go of their mobiles, opting instead to show a photo to their friends while holding onto the device. We argue that a better way of figuring the mobile is as a back stage device that interfaces with the front stage, both asynchronously using Facebook and Email or synchronously when co-present.

When thinking about co-present interactions, the coming together of people (as well as their devices), it is easy to see these as isolated events. But the bonds between people remain intact, as they move between periods of absence and presence. We can think of these as rhythms that punctuate life (Lefebvre 2004). Or as Suchman reminds us, interactivity, or engaged participation with others, does not just require a presence, but also an autobiography, and a projected future (Suchman 2007). Far from being isolated events, we look forward to our get-togethers. To be sure, some get-togethers are spontaneous, but rarely completely unexpectedly as a substantial degree of ordinariness characterises our lives (Dourish 2004). During get-togethers, we use our mobiles to share a past experience, a piece of our autobiography so to speak (Van House 2009). So why can't we use our mobiles to project into the future to better support such practices? Can a system be designed for people to draw upon the media they produce and consume while being mindful of absent others, but rather than sharing in the moment or forgetting altogether (Kindberg et al. 2005a), project into the future? That is, to anticipate future presence and the joy of sharing photos face-to-face might bring, where intersubjectivity is richer (Hollan & Stornetta 1992), and where we can co-orient (McLeod & Chaffee 1973) towards and make sense of the media together?

4.5 EXPLORING 'SHARE FACE2FACE'

All of these questions provoke us to think about what meaningful co-present interactions on mobile devices might look like. Shaped by our performative lens we identify a design space that we call Share Face2Face. This space assumes that the natural co-present sharing pattern is not necessarily a pattern of sending files, but a pattern of co-consumption and co-orientation. Within this class the natural sharing gesture is show-and-tell, in which a small group of people co-orient themselves towards the mobile's screen to look at a photo or listen to a song, together. Share Face2Face does not see these events as isolated, but as something that people might look forward to. As such it has a softened boundary of time, and explores how to enhance these encounters, by allowing people to draw upon the media that they produce and consume while apart and being mindful of others at a different time and place; and then to bring such acts of mindfulness into a co-present situation, at a later time or in a different place. This extended timeframe renders Share Face2Face into a design space that is more deliberate, slow, and curatorial; that leaves room for pausing and pondering; and allows people to anticipate and look forward to future presence. It is not about sharing across distance in the right-here-and-now, but sometime in the future, when the time is right, when we re-connect face-to-face.

If we look at this design space closely – how it interweaves with social practices; and how it is informed by a theoretical scaffolding of a variety of intellectual disciplines – we see a wicked problem. Basically this means that our formulation of the situation is an integral part to addressing it (Gaver 2012). As such, it warrants a research through design (RtD) approach (Zimmerman et al. 2010). While crossing disciplinary boundaries during our ongoing theoretical ponderings are an invaluable resources that help us to observe, talk, and think about co-present interactions, in choosing a RtD approach we should also remain true to our disciplinary orientations, namely interaction design and computer science. In crossing boundaries, it is easy to become dismissive of our own skill/et set, which in comparison to the perspectives and insights that others bring to the situation appear unremarkable and ordinary. As reflective and reflexive design researchers, we embrace the unique perspectives and skill sets that we bring to the situation and recognise how we generate knowledge. As interaction designers, we have developed a deep understanding of what software, interfaces, and sensors can and cannot do. These understandings and skill-sets colour our interpretations of theories. We agree with Gaver when he says, "the practice of making is a route to discovery" (Gaver 2012, 942). Thus, the prototype (Suchman et al. 2002) presents a chance to express, and through the process of prototyping, further our understandings of these co-present interaction spaces. We experiment with technological possibilities – new arrangements of people, contexts, and technologies – to build the right thing (Zimmerman et al. 2007), to change and possibly disrupt behaviour (Rogers 2012) and see what reality could become (Löwgren & Stolterman 2004). We are also inspired by Hutchinson et al's argument that there is value in creating very simple prototypes, or *technology probes*, that offer a single function and are often left strategically incomplete and flexible. Such probes are important tools in "challenging pre-existing ideas and influencing future design" (Hutchinson et al. 2003, 19).

In the following two sections we introduce one probe and one prototype and demonstrate how they express our understandings of the design space we identified.

4.5.1 *Sketching Share Face2Face*

In looking for pragmatic design solutions (Marsden et al. 2008) – ones that don't require adding infrastructure – we explored ways to convey our operative image of Share Face2Face on existing camera phones. Researchers studying camera phones have identified that the impulse to share is strongest in the moment (Kindberg et al. 2005a). A notion that they first identified in an earlier paper, appropriately titled: *I saw this and thought of you* (Kindberg et al. 2005b). Looking at how media is shared, posted, or 'Bluetoothed' on current devices we identified that all camera phones have built in sharing mechanisms. In most cases, these can be accessed from the contextual menu of a particular media item, revealing a list of sharing options: Bluetooth, Email, Facebook, or Messaging. To encapsulate our operative image, we wanted to present a similar option to users called Face2Face and created a suitable icon (see Figure fig. 8). To keep our probe simple and interpretatively flexible (Sengers & Gaver 2006), we created a dummy application on an Android phone that hooks into the Android's built in sharing mechanism. As a consequence, the Share Face2Face probe was displayed next to media sharing technologies such as Email, Bluetooth, and Facebook, which we hoped would make users reflect on how they currently share media (Sengers et al. 2005). If indeed a piece of media needs to be shared in the right-here-and-now, or if sharing it face-to-face later on might bring about a different experience, for instance by seeing the reaction of a friend's face when

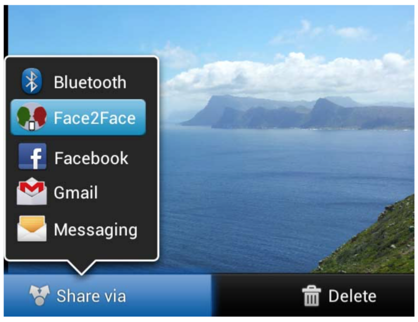


Figure 8: The *Share Face2Face* probe.

they look at a photo or listening to a song together. While we intend to use this probe on a larger scale, to first test out its feasibility we first went around the university campus during lunchtime to informally interview 10 students.

We first showed them the concept, which we accessed through the phone's contextual share menu from an image we took earlier. The Share Face2Face concept immediately caught the imagination of the students we interviewed. They talked about how face-to-face sharing of a photo is a different experience from Facebook: "facial expressions say more than likes." But when queried about how they might like to use such an application, their answers were characterised by vagueness. In their view such an application would be useful in general, but we only elicited a few specific scenarios, such as showing a picture of a flower that they saw on a recent hike to his friend , who is studying botany and might have something interesting to say about it. Reflecting on these results we realised that we failed to consider how the probe only takes on specific meanings in relation to specific images, evoking specific feelings. Clicking on the Face2Face sharing option can be seen as lodging an intent to share this media item face-to-face. This reminded us of Suchman's theory of Plans & Situated Action – commonly misinterpreted as a theory of only situated action, neglecting the important imaginary and discursive practice of planning (Suchman 2007). It is through planning that we project into the future and imagine how things might be. Such sharing intents – or plans – are an important resource within the situated action or practice of face-to-face image sharing. Lucy Suchman was also one of the first researchers within HCI to take the machine seriously as a companion to an interaction that goes beyond questioning if a machine is like a human to consider specific sociomaterial assemblages, the forms of invocation available to them, and their evocative effects (2007, 282). This suggests a discourse exploring the relationship between plans and situated action; how do these sharing intents relate to, and might be useful within, face-to-face encounters?

4.6 INTERROGATING THE SHARING INTERFACE

It is, however, difficult to implement a system that explores such a temporally extended discourse using the sharing interface of current mobile platforms. As we saw in the chapter 2, the digital content that such interfaces make available are ephemeral; intended to be handled as soon as possible. Such interfaces are incongruent with practices that emphasise collecting and face-to-face showing. But before we dive back down into the world of system design to, for instance, engineer a workaround to this incongruence, let us first (re-)consider some existing practices surrounding mobile media sharing in general, and face-to-face sharing specifically.

4.6.1 *Re-considering existing practices*

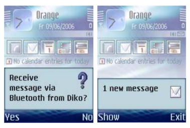
Up to now, we have mostly been looking at studies of mobile media use in Europe & North America. Walton et al's (2012) study of proximate media sharing practices by young people in Khayelitsha[3] presents us with an interesting point of departure. In this context and particularly among young users, "phones are often semi-pubic shared resources" (Walton et al. 2012, 403). This statement deserves some unpacking. Imagine if all the stuff that is stored on your phone is shared by default; always accessible to friends and family members, even if technically the devices is yours. Social display, relationship-building, and deference to authority, all play important roles in such proximate forms of sharing and help explain why "collocated phone use trumped online sharing" (Walton et al. 2012, 403), especially in a context where airtime and mobile data are expensive. Walton et al., therefore, characterise the mobile phone as a kind of "public personae, similar to social media 'profiles' " (2012, 403).

How very different these practices seem! They certainly don't resonate with how we think of sharing interfaces and interactions on current mobile devices. However, there are considerable overlaps between the study conducted by Walton et al. (2012) and Taylor & Harper's (2003) study of texting and mobile phone sharing practices of English teenagers. In their study, Taylor & Harper speak not only of localised interactions, whereby "the phone can be passed between those members of a group without the need for formal acts of offering or acceptance" (2003, 275), but also demonstrate how phones "give young people something to talk about amongst themselves" (2003, 280). By showing each other their messages, they are "not only offering each other the concrete, but also an intangible show of trust and loyalty" (Taylor & Harper 2003, 280). Certainly the emergence of the cloud and a proliferation of social networking sites have shifted some sharing practices, as well as enabled a whole range of new ones. Taylor & Harper's study, however, remains relevant and poetically demonstrates overlaps with how young people in Khayelitsha share media, because such acts of sharing "are a manifestation and a reflection of deeply rooted needs in [teenager's] social relationships, needs that have to do with systems of reciprocity and social solidarity" (2003, 268).

The smartphone did not feature in either of these two studies, but the deeply rooted needs and values that are articulated through collocated sharing practices remain salient. Smartphones aren't just technical means for communication, but also a resource to make conversation. We can incorporate images and videos, messages and music stored on, or accessed through, the device to chat with absent others, for instance using WhatsApp[4], just as we can incorporate as well as traffick[5] those media to draw in the people that surround us. It is, however, in this later regard that smartphones offer little improvements over the feature-phones or so-called dumb-phones that came before them. For we continue to speak of and treat the smart-phone as a deeply personal device. When we speak of the smart-phone as a social device, we generally refer to the myriad of social networks it connects to that are enabled by the cloud computing paradigm. In short, what makes the smart-phone 'smart' and social is that it can act as a portal into the cloud.

The Cloud computing paradigm and the range of social networks it enables finds its parallel in the Durkheimian paradigm of sociology, which posits a collective consciousness that is elevated and therefore also divorced from people's mutual involvement in their everyday practical activities[6]. Such a Durkheimian perspective, where ritual expressions become *super*-ordinate modes of existence, has also been applied to the study of mobile communication, most famously by

3. Khayelitsha is the largest township – or informal settlement – in Cape Town.
4. on this point see also O'Hara et al. (2014, 1133)
5. see Harper et al. (2007)
6. on this point see Ingold (2000, 196)


Orange
Fr 09/06/2006
No calendar entries for today
Receive message via Bluetooth from Diko?

Orange
Fr 09/06/2006
No calendar entries for today
1 new message

Figure 9: Screenshots from S60 device receiving a file. Image Source: (Harper et al. 2007)

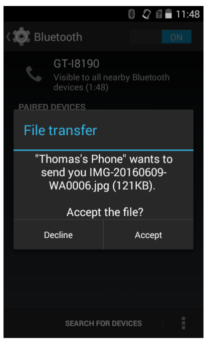
Rich Ling (2008). Coincidentally it was none-other than Marcel Mauss ([1950] 2000), a student of Durkheim's, that "dealt a blow to the entire Durkheimian paradigm from which it never fully recovered" (Ingold 2015, 10). For what Mauss had rediscovered was what our distant predecessors already knew: that "our lives are bound or drawn together as literally as two hands clasping"; and that "through the gift [. . .] – I am with you in your thoughts – and in your counter-gift, you are with me in mine" (Ingold 2015, 10-11). Therefore it does not matter if our everyday practical activities, or the materials we draw on during these activities, are physical or digital, it is these everyday acts that make us into social beings.

Contemporary 'smart'-phones miss precisely this point by assuming a single user whose social interactions are carried over wireless networks and subsequently mediated by social networks. Rainie & Wellman (2012) call this phenomenon 'networked individualism'. Where such networks are inaccessible, unreliable, or unaffordable, as Walton et al.'s (2012) study illustrates, people make due by swapping memory cards or sending media by bluetooth. Harper et al. (2007) have labeled such co-located mobile media sharing practices 'trafficking'.

Compared to feature-phones, contemporary smart-phones offer few improvements to support these practices. In some cases the opposite is true. On Symbian S60, the os that runs on older Nokia phones, bluetooth file transfers are surfaced as part of the messaging interface, as can be seen in fig. 9. By making incoming media accessible and thus also visible, such an interface affords a more conversational engagement with the media. On S60, it is also possible to stay permanently discoverable to make receiving files easier. Further still, by setting a device as permanently discoverable the device name (e.g. 'Nokia 6216') becomes a kind of status message that other people can read as they scan for bluetooth devices. The "Bluetooth Users Against Bush" campaign made use of this fact by encouraging participants to set their devices as discoverable and to customize their bluetooth device names to "Blu2th Against Bush", thereby creating a subtle way for participants to show solidarity and discover nearby people that align with the anti-Bush movement (Bluetooth Users Against Bush 2004).

Such 'trafficking' practices are no longer as well supported on more modern Operating Systems, such as Android or Windows Phone. For instance, these devices need to be explicitly set as discoverable and only remain so for a short duration (on Android) or while the user is on the bluetooth screen of the settings app (on WP). And while a user is presented with a dialog to accept incoming photo file transfer requests, the incoming file isn't surfaced to the user once it has been received, but is instead filed away in the 'bluetooth' folder on Android and in the 'Saved Pictures' album on Windows Phone. The received media is not at hand, and thus, doesn't afford the same types of engagements as those that are surfaced through messaging interfaces.

The above discussions on the technical means and obstacles to trafficking content are, of course, only one side of the coin. Whether users are showing or Bluetoothing a photo, as Harper et al's study alerts us, they need to do so with finesse: sending or showing "something dull or in poor taste would indicate the dullness or perversity of the sender" (2007, 254). We might say that choosing what to share is crucially important. However for the adolescents in Khayelitsha, where mobile devices are often semi-public, figure and ground are sometimes


11:48

Bluetooth ON

GT-I8190
Visible to all nearby Bluetooth devices (1:48)

PAIRED DEVICES

File transfer

"Thomas's Phone" wants to send you IMG-20160609-WA0006.jpg (121KB).

Accept the file?

Figure 10: Screenshot of Android device receiving a file

reversed. As Walton et al. reveal, the adolescents would delete, try to hide, or password protect content that they perceived as too private to share, and would at the same time try to display "certain content more prominently if they felt it might enhance their status" (2012, 409). Absence is "the necessary Other to presence" (Law 2004, 157) – a truism that has important implications for the design of systems to support co-located media sharing practices.

4.7 DESIGNING & IMPLEMENTING A CO-PRESENT PHOTO GALLERY

With Share Face2Face, we tried to sketch a system to provoke users to reflect on how they share media and to facilitate face-to-face sharing of such media. While testing the feasibility of the concept, we found that the sharing interface does not adequately support the temporally extended nature, or asynchronicity, of such an interaction. We also found that the usage scenarios envisioned for Share Face2Face were generally vague, while the sharing interface is generally used to share specific content with specific apps or people. Co-located sharing practices, as we have seen above, are generally the opposite, namely messy. In our designs and sketches thus far ('Gift' and 'Share Face2Face') our questions have largely revolved around understanding and extending the sharing interface. However, the studies we considered above demonstrate how people use the phone itself and the content stored on it to make conversation. Often the sharing interface plays no role during such encounters. It is all done through the screen and speakers.

As I look over the photos I took during the last couple of weeks on my phone I see: some photos I took during a workshop I attended; some photos of affection that I sent to my girlfriend who is traveling; and some I took with friends at a pub while we celebrated my birthday; and one photo I took of an empty bag of coffee to remind myself which roast I liked. Thumbnails of all of these photos fit within one screen on my phone. While I would be happy to show and share the photos I took during the workshop and while celebrating my birthday, the ones I sent to my girlfriend are too private to share, and the one of the coffee bag seems too mundane to share.

To return to the terminology we adopted earlier, one of the main implications of a device that functions in both front and back stage regions, like the mobile, is that maintaining the device in co-present situations can be problematic. I would quickly run into trouble if I wanted to show the birthday and workshop photos to a co-present friend. Since photos are displayed chronologically on the gallery, the act of going through these photos together or passing the phone around to people in a group could easily become uncomfortable. Swiping too far and I would no longer be pictured as being jovial with friends, but affectionate with a loved-one or as a rather mundane coffee snob. Alternatively, I could show the photos one-by-one, but such a presentation is hardly compelling.

Drawing primarily on the studies of existing co-located sharing practices and relating these to the above scenario, we sketched a proposal for a gallery app, where facilities for co-located sharing are built into the app. The vagueness that characterised the responses during preliminary Share Face2Face interviews, lead us towards interfaces that support tagging of photos into three general categories:

1. to share and make public
2. to keep private
3. to ignore-for-now and let drift into the background

When co-present users should then quickly be able to frontstage and display photos from any category. To create such a system, we first implemented an app that mimics the built-in gallery. As the app we envision doesn't require make use of the Operating System's sharing interface, we decided to return to Windows Phone 8 and implement an app that mimics the built-in gallery. Once implemented, we extended this app with functionality and accompanying interactions to quickly tag and frontstage tagged photos.

4.7.1 *Mimicking the built-in gallery*

On Windows Phone 8 the `MediaLibrary` API, as we saw in section 2.4.5, allows us to access photos and photo albums. In their recipe-driven approach towards Windows Phone 8 programming, Lalonde & Totzke (2013) make use of precisely this API to develop a gallery app. We based our app on their 'recipe', because it shows how to integrate the `MediaLibrary` API into an app that follow a common architectural design patterns for event-driven applications, namely MVVM (see MSDN 2014)

Lalonde & Totzke's app (2013, 163) consists of three types of pages:

- `MainPage.xaml` – to display the root picture albums
- `AlbumGalleryPage.xaml` – to display grid of thumbnail images
- `ViewPicturePage.xaml` – to display single image

Throughout these three page types, the app makes use of a single ViewModel, appropriately called `MainViewModel.cs`, which is stored as static singleton by the `App.xaml.cs` class and accessed by the above UI pages. This ViewModel interacts with the Models of the app, namely the `Picture.cs` and `PictureAlbum.cs` classes of the `MediaLibrary` API. The MainViewModel has one method – `LoadData`– and exposes four main properties:

- `PhotoAlbums` – An `ObservableCollection` of `PictureAlbum` that is populated by the `LoadData` method and which contains the main albums of the phone.
- `CurrentAlbum` – A `PictureAlbum` that keeps track of the album that a user selects and also updates the `CurrentAlbumPictures` property when it is changed.
- `CurrentAlbumPictures` – An `ObservableCollection` of `Picture` that reflects the pictures in the currently selected album; it is updated anytime a user selects a new album.
- `CurrentPicture` – A `Picture` that is updated anytime a user selects a photo to view.

MainPage.xaml is displayed when the app first launches. In the OnNavigatedTo method of its code-behind file[7], it simply instantiates the MainViewModel and calls its LoadData method. Keeping the code that lies directly behind a UI page to a minimum is a typical practice when following the MVVM pattern. The UI of the page consist primarily of a LongListSelector. It is used to display the list of albums that are stored on the phone, such as the Camera Roll, Saved Pictures, and Sample Pictures albums. It does so by *binding* its ItemsSource to the PhotoAlbums property of the MainViewModel, which has been set to the list of albums stored on the phone by the LoadData method that was called above. Each item – in our case a PictureAlbum – is rendered by a DataTemplate, which consists of an Image and two TextBlocks. The Image is bound to the Pictures[0] property of the PictureAlbum and therefore displays a thumbnail of the album's first picture[8]. The TextBlocks are bound to the Name and Pictures.Count properties of the PictureAlbum and therefore display the album name and picture count of that particular album – e.g. Camera Roll (Picture Count: 73) – next to a thumbnail of its first picture. The user can then scroll through and select an album. Upon selection the LongListSelector_SelectionChanged method of the page's code-behind is called. That method first checks if the selection is valid and then sets the CurrentAlbum property of the MainViewModel to the selected album and navigates to the AlbumGalleryPage.

AlbumGalleryPage.xaml consists of two main UI components, a TextBlock and a LongListSelector. The TextBlock is bound to the CurrentAlbum.Name property of MainViewModel, . The LongListSelector is used to display the list of pictures in the currently selected album. It does so by binding its ItemsSource to MainViewModel's CurrentAlbumPictures property and uses a DataTemplate to render each item – in our case a Picture – in a Grid layout. This template consists of a single Image that is bound to the item itself – in our case a Picture – and thus displays the picture's thumbnail[9]. On this page users can see and scroll through the photos contained in a particular album, e.g. Camera Roll. From here they can either navigate back to the previous page or select an individual photo. Upon selection the LongListSelector_SelectionChanged() method of the page's code-behind is called. That method first checks if the selection is valid and then sets the CurrentPicture property of the MainViewModel to the selected album and navigates to the ViewPicturePage.

ViewPicturePage.xaml consists of two main UI components, a TextBlock and a Image. The TextBlock is bound to the CurrentPicture.Name property of the MainViewModel, while the Image is bound to the CurrentPicture property[10]. On this page users can look at the picture they just selected and navigate back to the previous page.

In their gallery app, Lalonde & Totzke (2013) have made heavy use of data bindings. By leveraging these bindings the resulting code is both elegant and concise[11]. With a Model consists of two classes, a single ViewModel with four main properties, and three UI pages with minimal code-behind, we have created an app that replicates most of the main functionality of the built-in gallery. In

7. WP8 UI pages consist of two components: the XAML markup of the UI and a code-behind file.

8. This binding is achieved with the help of a Converter, which takes an Picture and a String as parameters, and returns a BitmapImage. The string is used to control the BitmapImage's source is set to the Picture's GetThumbnail() or GetImage() methods, respectively.

9. This binding is achieved with the help of a Converter, which takes an Picture and a String as parameters, and returns a BitmapImage. The string is used to control the BitmapImage's source is set to the Picture's GetThumbnail() or GetImage() methods, respectively.

10. This binding is achieved with the help of a Converter, which takes an Picture and a String as parameters, and returns a BitmapImage. The string is used to control the BitmapImage's source is set to the Picture's GetThumbnail() or GetImage() methods, respectively.

11. if we disregard the boilerplate code that is part of any mobile MVVM app

world of programming such brevity is virtue (Oram & Wilson 2007, 488), mostly because it is easier for programmers to scan and reason about the code.

4.7.2 *Extending the built-in gallery app*

Compared to the built-in gallery, the app that Lalonde & Totzke (2013) develop in their 'recipe' is still rather rudimentary. It does not support common gestures of the built-in gallery, namely:

- dragging or swiping to the next image.
- double tapping to zoom in and out of an image.
- pinching & dragging to zoom and pan an image.

Users have learned how to use these gestures to navigate and view their photos with ease and finesse. We therefore extended Lalonde & Totzke's (2013) app, following similar recipes, to support these gestures and so we can familiarise ourselves with how such gesture interactions are implemented on Windows Phone.

The changes we had to make mostly focused on the `ViewPicturePage` which we renamed `PictureInAlbumPage`. In the `MainViewModel` we deleted the `CurrentPicture` property and instead created a `CurrentPictureIndex` property. With this property it is still possible for a page to access what used to be the `CurrentPicture` by using its index to access it from the `CurrentAlbumPictures` collection. On the page itself we changed the `Image` component to a `Pivot` and bound its `ItemsSource` to the `CurrentAlbumPictures` and set its `DataTemplate` to an `Image`, which in turn is bound to and displays an individual image of that collection. The Pivot control then listens for swiping gestures and handles them appropriately by advancing to the next or previous image. We also expanded the `PictureInAlbumPage` to listen and react to pinching gestures using the Windows Phone Toolkit Library[12]. Each gesture is reported back to the page in its code-behind first as start and stop events (e.g. `PinchStarted`) and then as a stream of delta values (e.g. `PinchDelta`). These delta values, in turn, are processed and fed into a composite transformation that has been attached to the image. This transformation then scales and translates the image. We used the same principles to listen for and react to double tap events that zoom in to and out of an image.

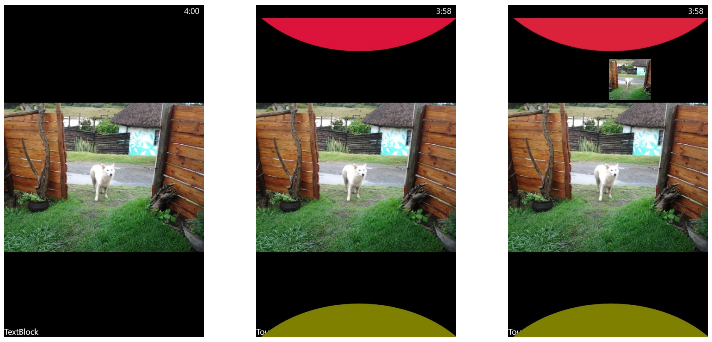
Strictly speaking the gestures we added above don't change the functionality of the original system. Even on that rudimentary app, users could navigate through their collection of and view individual photos. However, supporting these gestures allows for users to navigate between and view their photos more quickly and fluidly. With a simple swipe users can move from one photo to the next, whereas on the rudimentary app users would need to go back to the previous (album) page, find the thumbnail of the photo they just viewed, and tap on the adjacent thumbnail.

Implementing and repeatably testing these touch gestures allowed us to gain familiarity with how gesture support is implemented on Windows Phone 8, but equally importantly we learned how certain gesture work together as systems whereas others compete with one another. This familiarity became an invaluable resource as we sketched out how to extend this gallery with features and interactions to support co-located sharing practices.

4.7.3 *Designing the co-present gallery*

To design interactions to support co-present engagements with our gallery app, we were inspired by how common gestures enable quick and fluid experiences.

12. https://github.com/MicrosoftArchive/WindowsPhoneToolkit



(a) Viewing a photo.

(b) Zones appear when beginning to drag a photo vertically, or double tapping the photo.

(c) Dragging the photo towards the upper zone.

Figure 11: Co-present photo Gallery interaction.

Lucero et al likewise levered such gestures to support collaborative collocated photosharing (Lucero et al. 2011). Tbl. 1 gives an overview of the types of gestures Windows Phone supports.

Table 1: Common gestures used in Windows Phone

Gesture	**Description**
Tap	A finger touches the screen and releases.
Double Tap	A finger taps the screen twice and then releases.
Hold	A finger touches the screen and briefly holds in place.
Drag	A finger touches the screen and moves in any direction.
Flick	A finger drags across the screen and lifts up without stopping.
Pinch	Two fingers press on the screen and move around.

When viewing an individual photo, our extended gallery app already makes use of the Drag and Flick gestures for navigating between photos as well as the Double Tap and Pinch gestures for zooming. This only leaves us with two remaining touch gestures: Tap and Hold. The pervasive Tap gesture already has a well established usage pattern, namely to active UI elements such as Button or thumbnail images. The Hold gesture, on the other hand, is less established; sometimes it used to bring up a context menu when users long-press on a UI element. Drawing on our experience of adding gesture support to our gallery app, we looked for opportunities to combine individual gestures into systems of gestures. For instance, when a user zooms into an image, by using the Pinch or Double Tap gesture, the Drag gesture no longer navigates to the next image, but is used for panning the zoomed-in image. Only when the user zooms out of the image again is the Drag gesture used for navigating to the next image. Looking for similar opportunities to overload and combine individual gestures, we noticed that the gesture used for navigating between images is a *horizontal* Drag, meaning *vertical* Drag gestures wouldn't interfere with navigation. On paper we experimented with various sketches that use vertical dragging before settling on the one shown in Fig 11.

When a user starts to drag the photo vertically, two semi-circular zones – one green and one red – start to appear on the screen. The green zone is used to tag photos for sharing, while the red zone is used to tag them for keeping (private). As the user drags the photo, a thumbnail of the current photo tracks the users finger and when that thumbnail enters the green (or red) zone, it is tinted green (or red). Dropping the thumbnail in a zone tags the photo; the zones then fade out again. Long-pressing on the photo also makes the zones visible. Only this time the user can tap on them. This filters the collection and makes all photos that haven't been tagged through that zone invisible. For instance, when the user taps on the bottom, red zone only photos that are marked for keeping (private) are displayed.

4.7.4 *Implementing the co-present Gallery*

When we tried to implement the above system, we noticed that the *horizontal* Drag gestures (for navigation) and the *vertical* Drag gestures (for tagging) couldn't be neatly differentiated from one another. On paper, we failed to consider that users don't swipe along perfectly straight horizontal (or vertical) lines! Every horizontal drag, always has a residual vertical component and vice versa. We tried to implement a threshold to account for this residual, but the built-in `Pivot` component wouldn't let us reconfigure how it responds to Drag events beyond enabling or disabling them entirely. Instead we adapted the tagging interaction to also make use of the Hold gesture, so the user first long-presses on the image – this fades in the zones and tells the `Pivot` to ignore Drag events. Just as before, the user can then drag the photo's thumbnail into a zone to tag it. Implementing this revised system, allowed us to better discriminate between the horizontal and vertical Drag gestures, but in practice the tagging interaction became cumbersome and slow. The Hold gesture, used to activate the tagging system, takes a whole second to fire. In contrast, swiping to the next or previous image only takes a fraction of a second. So if a user wants to tag 15 images, they need to long-press on 15 different images for a second each. We experimented with other gestures for activating the tagging system. The quickest and most reliable of these is an interaction whereby users double taps on the image, but upon the second tap they don't lift their finger again. Instead they continue and drag the image towards one of the zones, which are already fading in. By combining the Double Tap and Drag gestures in this way the resulting systems for tagging photos in almost as quick and fluid as the gestures used for swiping between images.

4.8 DISCUSSION & OUTLOOK

Design, as Ingold notes, extends into making (2013, 70). This certainly resonates with our experience of designing and implementing/making our co-present photo gallery. By replicating how gestures are used in built-in apps, and continually testing the resulting app on a real device, we could understand the use of gestures more deeply and draw on those understandings to design the tagging system of our co-present gallery. While the resulting system was certainly influenced by how we sketched out our idea on paper, it didn't correspond with that idea exactly. This is why Ingold argues that we should look towards cooking and gardening, instead of architecture, for our models of design, because both gardeners and chefs don't just see the ways things currently are, but also where they are going (2013, 70). By also responding to where things are going, the relationship between design and making can no longer be found in connecting a pre-conceived idea to an object. Instead the relationship *goes between* idea and object, "following and reconciling the inclinations of alternately pliable and recalcitrant materials" (INGOLD 2013, 70). Even thought the gesture system on Windows Phone is documented as an API, in practice it also becomes a design material. We learn about such materials – or any material for that matter – more deeply by engaging with them to discover and respond to the ways in which they are pliable and in which they are recalcitrant.

Just as design materials participate in a design process, so too do the theories and studies of existing practices we consider and integrate. In this chapter we have learned how co-located or any type of social interactions are characterised by their ordinariness (Dourish 2004); we take them for granted. These interactions are enmeshed with unwritten rules, feelings, expectations, anxieties, and experiences (Goffman 1959; Hall 1966). So, we drew upon theory to develop analytical lenses and sensitise ourselves to key practices, sometimes rendering these visible to us in the first place. We have characterised this approach as a constant, yet productive and generative site of struggle that depends on one pervasively misunderstood activity: *reading* (Ingold 2007). When we speak of reading we see it as an act of habitation rather than consumption. Just like a renter furnishing her apartment with objects, acts, and memories, by continually engaging with (or reading) theoretical texts at different stages of our research, we have made these texts our own. We used analytical lenses derived and refined through reading to understand and critique approaches and conceptual foundations of related work including our own. These understandings showed us that first and foremost we needed a better way to think about co-located interactions on mobile devices. By interrelating theories we uncovered that such interactions are situated primarily in a social ecology with devices, not in a device ecology with people. Our dialogic engagement with theory allowed us to develop such grounding principles. We have identified two design spaces Share Face2Face and the Co-present Gallery, and have begun to explore them through the process of design, which so far has differed from more traditional user-centred design (UCD) processes.

In his widely cited paper on context, Dourish notes that if we are to sensitise ourselves to different disciplinary orientations on profound concepts such as context, identity, and communication, this not only implies "a change to the ways in which we go about designing technologies, but also a change to the technologies that we design" (Dourish 2004, 28). This is more than a change in process, because the results of a PD or UCD process can still be stable, static, and closed (Dourish 2004). It was after all "a circular move of writing a cognitivist rationality onto machines and then claiming their status as models for the humans" (Suchman 2007, p.259) – a view that is not only prevalent in academic but also in everyday discourses. While we firmly believe in the philosophies of PD and UCD, in practice we have so far not adhered to them. The discussions presented thus far characterise a different, first step that doesn't fit into traditional models of UCD. We call it *trying to understand*. In our account, we have articulated – sometimes explicitly and other times deliberately implicitly – this *theory on design* (Zimmerman et al. 2010) in the process of uncovering implications for the design of co-located interactions. If we take design seriously as a method of research, this should happen in parallel. We aren't saying that all research should engage with theory in such a manner, but it was a fruitful approach for us, and is an emerging trend in HCI (Rogers 2012). We also believe that it is an approach that could benefit other research that latches onto more tacit aspects of our everyday lives. This is why we report and interrelate them here, because *theory on design* makes more sense in relation to *specific* design problems (Gaver 2012). To develop research through design as a methodology and to hold ourselves accountable to the decisions we, while struggling with, nevertheless made, we respond to Zimmerman et al.'s call to action by documenting the whole process, showing "how theories from other disciplines were integrated" and beginning with the crucial first step: *problem framing* (Zimmerman et al. 2010, p.316). But further than theory on design, we hope that our discussions, interpretations of theory in relation to co-located interactions, probes, and technology experiments surrounding the probes, will serve as placeholders that open a new and fruitful design space: the space of co-located interactions that fit primarily into our social ecologies and not just our device ecologies.

4.9 LOOKING BACK & MOVING FORWARD

Looking back over the prototypes and probes we explored and implemented to this point, we have gained an understanding of the material realities of information that is stored on the mobile phone and discovered that human practices of gifting and co-present sharing are incongruent with the specific ways in which information is passed through the mobile's sharing interface. While it may be possible to engineer apps that workaround this incongruence, we are hesitant to do so, because such workarounds often result in fragile apps. If the woodworker fails to account for the direction of the grain of the wood they are working with, the chair they build will not be strong. A similar conclusion lead us to abandon expanding the Share Face2Face probe. So we shifted the problem from sharing to presentation, from the sharing interface to the Gallery app. While our co-present Gallery might provide better mechanisms for us to manage and present our stuff (and ourselves in the process) to those around us, the impasse we reached earlier still concerns us.

As chance would have it just as we shifted our inquiry from the mobile sharing interface to its Gallery, Microsoft announced major architectural changes to the Windows Phone platform that re-introduced the file as a major component of the platform, its datastore, and its sharing interface. This presented a challenging moment for our inquiry: a point of departure, so to speak. Do we keep the object of our study – Windows Phone 8 – consistent and ignore these changes to the underlying platform? Or do we study these changes and integrate them into our inquiry? We certainly felt validated: however mundane or anachronistic the file may seem in our contemporary landscape of computing that is dominated by the cloud and the mobile, it still serves many important purposes, as we saw in chapter 2. After much deliberation we decided that it would be reckless to ignore these developments for the sake of consistency. But we also felt that we would be doing our inquiry a disservice to not engage with our deeper concerns. Asking a simple question – does Windows Phone 8 support gifting? – got us to this point. Before we ask the same of Windows Phone 8.1, we need to pause and reflect if this is still the most pertinent question we should be asking. Perhaps our concerns and dissatisfactions we developed through our practical and critical engagement with the platform show that we are not putting our question marks deep down enough.

5 INTERLUDE

5.1 INTRODUCTION

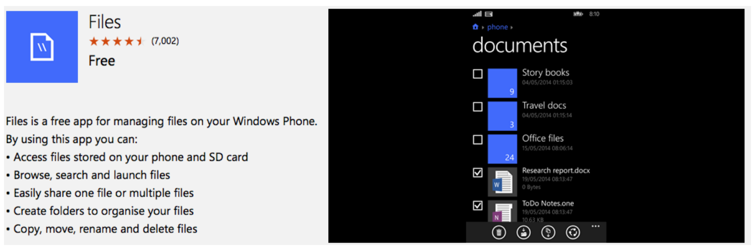
It was Paul Dourish (2014) who alerted us to the fact that inquiries into the materialities of information are best conducted at moments when such materialities are in flux. In the previous chapters we have seen how contemporary mobile platforms are moving away from the notion of a file: almost entirely on WP8 and on Android by homogenising cloud storage with local storage. In this brief interlude, we consider the convergence of PC and mobile operating systems by looking at the changes introduced by the Windows 8.1 (W8.1) and Windows Phone 8.1 (WP8.1) operating systems.

A release with a minor version number typically indicates that only small changes have been made. For the WP8.1 UI this is certainly true. From the users perspective the release is almost indistinguishable from WP8, save for a few new features that were added. But under the hood WP8.1 is a major architectural landmark, because it incorporates the Windows Runtime (WinRT). This means that for the first time Windows and Windows Phone Operating Systems share a runtime, which enables a number of other shared features. For instance, W8 and WP8.1 share the same .NET subset and have an identical app model and app lifecycle (starting, pausing, resuming, stopping), and there remain only few UI widgets and APIs that are specific to either WP8.1 or W8.1. When people speak about the convergence of devices (from phones, phablets, tablets, and ultra-book laptops), this shared runtime and code base is what we might refer to as material evidence of such convergence. The W8.1 and WP8.1 releases are also the product of a restructuring within Microsoft. Whereas before there were separate teams responsible for developing Windows for every device it runs on – PCs, Servers, tablets, gaming consoles and phones – the newly created OS division within Microsoft is responsible for all operating systems: from Windows Phone all the way to Windows Server. Looking back, the technology journalist Peter Bright, sees Windows and Windows Phone 8.1 as a critical component to realising its vision of a shared core operating system that "can span hardware from little embedded Internet of Things devices to games consoles to PCs to cloud-scale server farms" (2016, 3). Because the material properties of this core operating affects potentially millions of users and devices, it is worth investigating it in more detail.

At Microsoft's BUILD2014 conference, where WP8.1 and the shared APIs it supports were introduced in detail, a few 'new' features of the WP8.1 platform garnered applause from developers: the re-emergence of the file as a central component of the new sharing contract specifically and of system design more generally. These architectural changes also enabled Microsoft to launch a file manger app called *Files*. Before we dive into the technical details of the architectural changes introduced with WP8.1, let us first examine just how users experience these changes by considering, as a case study, the *Files* app.

5.2 EXAMINING THE WINDOWS PHONE 8.1 FILE MANAGER

As can be seen in fig. 12 the app is one of the more popular ones on the Windows Phone app store and has received overwhelmingly positive reviews from which we can gather that its not just developers who applauded the re-emergence of



Files
★★★★½ (7,002)
Free

Files is a free app for managing files on your Windows Phone.
By using this app you can:
• Access files stored on your phone and SD card
• Browse, search and launch files
• Easily share one file or multiple files
• Create folders to organise your files
• Copy, move, rename and delete files

8:10
phone ›
documents
☐ Story books 04/05/2014 01:15:03 9
☐ Travel docs 04/05/2014 01:15:14 3
☐ Office files 15/05/2014 08:06:14 24
☑ Research report.docx 19/05/2014 08:13:47 0 Bytes
☑ ToDo Notes.one 19/05/2014 08:13:47 10.63 KB

Figure 12: The Windows Phone File Manager: *Files*

the file, but that for users too being able to manage their stuff remains of central concern.

The purpose of the *Files* app is to give users direct access and control over the files stored in a few common folders on both their phones internal and external storage, namely:

- Documents
- Downloads
- Music
- Pictures
- Ringtones
- Videos

One way users can act on the files stored in those folders is by long-pressing on them. Fig. 13a shows the menu that pops up after long-pressing on a photo file stored in the user's camera roll folder, a subfolder of the pictures folder of the SD card. Looking at the options in this menu we can see that the *Files* app, by virtue of the APIs it calls on, reproduces the familiar grammars of actions from the PC – delete, rename, move to, and copy to. But we also see that *Files* has expanded on this traditional grammar with a new action: *share*. This action, along with a few others, has been deemed so important that when a user presses, instead of long-presses, on a photo file, not only are the traditional actions no longer accessible, but *share* is displayed most prominently along side a range of other, new actions. When a user presses this share icon (fig. 13b), for instance to email a file, they are presented with a list of apps, and after they choose Email (fig. 13c) the photo is attached to an email draft (fig. 13d).

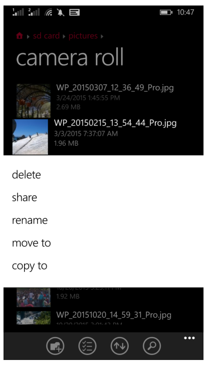
This interaction is similar to what is often called the 'sharing experience' on Android and WP8. But if we look more closely at the screenshots and dig a little deeper into the way 'share' is operationalised on WP8.1 we notice a few subtleties. For instance once the user presses the 'share' button, *Files* first saves a copy of the file to the 'Saved Pictures' folder, as can be seen in the top left corner of fig. 13b. Only after this copy operation completes, is the user prompted to choose an app they wish to share the file with. Although, strictly speaking they aren't sharing the file they were looking at, in our case:

`D:\Pictures\Camera Roll\WP_2015_02_15_13_54_44_Pro.jpg`

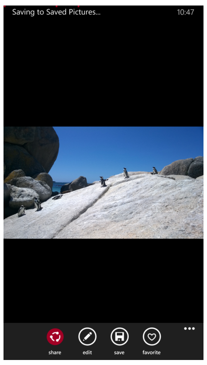
Instead they are sharing a copy of that file, which has been stored in the 'Saved Pictures' folder, in our case:

`D:\Pictures\Saved Pictures\WP_2015_02_15_13_54_44_Pro(1).jpg`

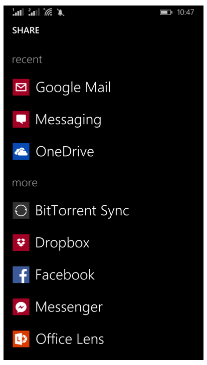
And if we compare the file sizes between the original file stored in the camera roll folder of fig. 13a (1.96 MB) with the photo that is ultimately attached to the email of fig. 13d (485 KB), we can tell that the email app has further resized and probably also compressed the photo. This is not a bad thing per se, especially in South Africa, where the photo was taken and subsequently shared, mobile data costs are high. So I was happy to only have to pay for roughly a quarter of the data costs to email the photo to my family to show them some penguins I saw.



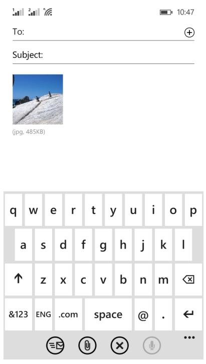
(a) Standard grammar of action.



(b) Expanded grammar of action – share.

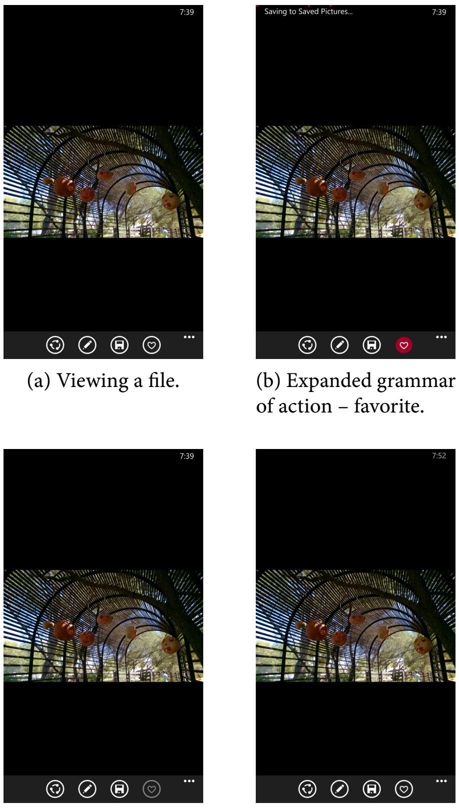


(c) List of "share targets."



(d) Emailing the file.

Figure 13: Emailing a file.



(a) Viewing a file. (b) Expanded grammar of action – favorite.

(c) A favorited file. (d) Revisiting a previously favorited file.

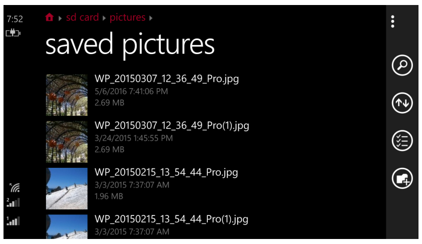
Figure 14: Favoriting a file.

To summarise, it might seem to the user that they are emailing a photo file, but they are instead emailing a resized and compressed copy of a copy of a photo file.

These resizing and compressing operations can also lead to problems, for instance when I emailed the photo a second time, not only did this result in a second copy stored in the ’Saved Pictures folder, but my intention was also different this time. I was responding to a request of a friend. Because the camera on my phone was better than his, he wanted to *have* the photo to include in an album he intended to make to commemorate his visit. I was unable to email him the original uncompressed version that would be better suited to printing.

Unpacking this simple scenario we can conclude that the grammar of ‘share’ is predicated on the grammar of ‘copy’ and often results in the creation of multiple copies, one in the ‘Saved Pictures’ folder and any subsequent copies apps might create of the file. In the case of the email app, it saves a resized and compressed copy of the photo file in its sent emails folder. The copy stored in the ‘Saved Pictures’ folder is especially problematic if we look at the grammar of favouriting.

Favorite is part of the expanded grammar of action that the *Files* app has brought to WP8.1. It is an action that has been imported from the web, where it is used to designate one or a couple of items as favourites from a collection that might encompass many items. Items that have been marked as favourites are often made more visible and can be accessed quicker than other items. In fig. 14 we break down the interaction of favouriting a file. We begin with (fig. 14a) viewing a photo file in the camera roll. Next (fig. 14b) we press the favourite button. This results in the button being greyed-out (fig. 14c), which indicates to the user that the photo file has been successfully favourited. This marking, however, is ephemeral and only lasts as long as we are viewing the photo file.



saved pictures

WP_20150307_12_36_49_Pro.jpg
5/6/2016 7:41:06 PM
2.69 MB

WP_20150307_12_36_49_Pro(1).jpg
3/24/2015 1:45:55 PM
2.69 MB

WP_20150215_13_54_44_Pro.jpg
3/3/2015 7:37:07 AM
1.96 MB

WP_20150215_13_54_44_Pro(1).jpg
3/3/2015 7:37:07 AM

Figure 15: The resulting muddle in the 'Saved Pictures' folder.

For should we revisit the photo file at some later point, as we do in fig. 14d, the button is no longer greyed-out. If we look more closely at fig. 14b we can see that while we are marking a file as favourite, the *Files* app is creating a copy of the file and saving it in the 'Saved Pictures' folder. Thus, favouriting does not attach a tag to the photo, nor can it be considered an attribute of an object. In the *Files* app a favourited photo file is marked as thus by virtue of its location, when a copy gets placed in the 'Saved Pictures' folder. A choice that might seem understandable on its own, but quickly becomes inexplicable if we consider that the 'Saved Pictures' folder, as can be seen in fig. 15, already contains copies of multiple other files, for instance the ones we emailed earlier and that we might very well not consider a favorite.

Despite the proliferation of copies and resulting muddle in the saved pictures folder, the *Files* app empowers users by granting them direct access and control over their stuff. But users still don't have direct access to *all* of their stuff, rather a subset of their stuff that is stored in a few common folders, such as Pictures and Videos. This is precisely the point that the user 'J' points out in her review of the *Files* app (see, Microsoft 2014), a review that 12 other users found helpful:

> It works fine, but it isn't really a file explorer. It's one place to find pictures, videos, downloads, and documents but it doesn't show the folder structure in the phone nor any apps or app created files like my recordings. It just isn't what is says.

J is, of course, referring to the stuff – *her* stuff – that is stored in app sandboxes and thus is not directly accessible. It is this topic, and the roles that files play in it, that we turn our attention to next.

5.3 THE BATTLE IN THE SANDBOX

What precisely are these "files like my recordings" that J mentions and why can't she access them? Developers typically refer to data that users orient to in such as way as 'user data'. As the name already implies, it is data that somehow *belongs* to the user. The Windows Dev Center (2016) characterises *user data* as data that users create and manage using an app, but that is "useful or meaningful to more than one app" and that "the user wants to manipulate or transmit as an entity independent of the app itself". Further, user data is differentiated from app data, which is data that an app creates and manages itself: user preferences, reference content, and runtime state. Using the above nomenclature and applying it to the gifting fiction we developed in the chapter 2, we would consider received gifts as a typical form of user data.

However it was only after much debate that we tentatively decided to store received gifts in the `Pictures\` public storage directory. And this decision – to store user data in a public folder – is far from standard practice, as illustrated by J's above comment. In fact, these debates and difficult decisions are symptomatic of the battle for control between apps and contemporary mobile operating systems.

The WP8 and Android Operating Systems only reluctantly grant apps with limited access to the user's files, typically in the form of a `DataStream` or opaque

`Uri`. In return apps are promised isolated storage, a sandboxed form of storage where they are in total control. Due to these vastly different forms of storage and access models, if apps need access to a file beyond the initial moment of sharing they are forced to recreate it in their sandbox. WP8.1, however, allows some files to break free from their respective sandboxes. This is achieved using the new share contract and file picker implementations. At the BUILD2014 conference developers applauded the announcement of these new features (GALLARDO & SINGH 2014).

5.3.1 *Examining the share contract & file picker*

A central aspect of the WP8.1 share contract is the `DataTransferManger`, an OS component that brokers data from one application sandbox to another. Using the `DataTransferManager` an app that wants to, for instance, share one or more photos stored in its sandbox can bundle that data – principally consisting of a list of `StorageFile` items – into a `DataPackage`, pass it to the `DataTransferManager`, and request that the share UI be shown to the user. The OS then prompts the user to select an app they want to share that content with. This app is referred to as the share target. Next the OS launches[1] the share target app and passes it a `ShareTargetActivatedEventArgs` parameter that contains a `DataPackageView`. From here the share target can access the file references the sharing app passed into the corresponding `DataPackage` that it created.

The share contract

To examine the properties of this share contract and the brokering role the OS plays within it more closely, we can create an app – let's call it 'Share Inspector' – and register it as a share target for `*.jpg` and `*.png` image files and share with it an image from the phone's camera roll. In the context of this example the camera roll folder, and all common folders for that matter, are effectively sandboxed, because if an app doesn't request in its manifest special permission to access common folders those folders and the files they contain are inaccessible to that app, and thus, de-facto sandboxed from it.

If we inspect the properties and attributes of the `StorageFile` that arrives at the share target, we notice that it has been designated `Read-Only | Archive`. And if we try and re-access the file during a future launch of the app an exception is thrown to indicate that the app is no longer allowed to access that file. So we would not be able to implement our gifting scenario of chapter 2 using the share contract. Further still, the share target is obliged to report the share operation as completed and during the BUILD 2014 conference GALLARDO & SINGH (2014) strongly encouraged developers to do so promptly. Upon reporting the operation as completed, the share target app – 'Share Inspector' – is closed and the user is returned to the sharing app. While it is possible to pass a, so-called `QuickLink` parameter[2] to the `ReportCompleted()` method, this parameter is ignored on WP8.1. Taking these insights together we can conclude that the purpose of the share contract is not to enable a discourse between apps and stuff that is stored within their sandboxes, but to quickly post stuff online or to send it as part of a message or email.

The file picker

Through the share contract, as we have seen above, it is possible for one app to temporarily pass a file reference to another. The file picker, on the other

1. or resumes if it is already running
2. A `QuickLink` is designed to facilitate sharing to familiar destinations within an app on W8. For instance, when a `QuickLink` to 'Email Mom' is returned after sharing an subsequently emailing a photo to Mom, the 'Email Mom' `QuickLink` is listed at the top of the share target list, above the more general Email app.

hand, allows one app to reach into another's sandbox to obtain access to one or more of its files. The former action is one of receiving, the later one of choosing. By choosing a file, rather than having it shared with it, the app is freed from the constraint of having to complete the share operation as soon as possible. However, the file reference it obtains is again restricted to read-only operations and expires once the app closes.

5.3.2 *Discussion*

Since in both of the above approaches towards sharing data the receiving app is only getting a file reference, data does not have to be duplicated and the share operation is not restricted to any size limits. This is at least the conclusion that Gallardo & Singh (2014) come to at BUILD 2014. However, the nature of the file references that an app ends up with are such that they either enable a limited, but common set of scenarios – to post, message, and email the file – or they force apps to recreate files in their own sandboxes.

With the well received file manager *Files* that Microsoft introduced with WP8.1 users are empowered with direct access and more control over their stuff. However, examining this file manager and the new sharing contract in more detail, we must note that users are at the same time disempowered through a proliferation of copies. Sharing a photo from the *Files* app creates at least one and often further file copies: one that is always stored in the 'Saved Pictures' folder and another that often resides in the receiving apps' sandbox.

These copies are a symptom of the battle for control over user data between apps and the OS. While the OS plays a brokering role to grant access to files outside an apps' sandbox, including those stored in common folders such as `Pictures` or `Videos`, the limited and temporary nature of that access is of consequence. For many apps, such access is too limited. They opt instead to keep user data, or shared and subsequently re-created copies of that data, in their sandboxes instead. The OS, however, has promised apps isolated, sandboxed storage in return for limited access outside of such sandboxes. Therefore much user data remains opaque to the user – hidden behind apps and outside of their, as well as the OS' direct control.

5.4 TAKING BACK CONTROL

Fortunately WP8.1 introduces a third, less pervasive, mechanism to excerpt control over user data: the `KnownFolders` storage library. This is the same API that the Windows Phone 8.1 File Manager itself uses to gain direct, programmatic access to "common locations that contain user content"[3]. As this is a powerful API with big privacy implications a corresponding capability must first be enabled in the app manifest file in order to use the API. The snippet in lst. 5.1 illustrates a basic usage scenario.

An important difference between the `StorageFile` references we obtain in lst. 5.1 and the one we obtain through the `Filepicker` of chapter 2, is that the references obtained through the `KnownFolders` library can be reobtained even if the app is restarted and that full access is granted. That is, it is possible to move, rename, or delete any file or subfolder contained within the pictures folder rather than just read from the file or to create a copy.

5.4.1 *The MyStuff file manager & datastore*

We will not bore the reader with the details at this stage, but started to implement our own file manager, which we call *My Stuff*, using the MVVM – Model-View-

3. https://docs.microsoft.com/en-us/uwp/api/windows.storage.storagefolder

Listing 5.1 Enumerating the files in the user's Picture Folder

```
using Windows.Storage;
using Windows.Storage.Search;
using System.Threading.Tasks;
using System.Diagnostics;

// Get the user's Pictures folder.
StorageFolder picturesFolder = KnownFolders.PicturesLibrary;

// Get the list of files in the current folder
IReadOnlyList<StorageFile> files = await picturesFolder.GetFilesAsync();

// Iterate over the results and print the list of files
// to the Visual Studio Output window.
foreach (StorageFile file in files)
    Debug.WriteLine(file.Name);
```

ViewModel – software design pattern that facilitates a clean, testable separation of concern between user interface and the application logic (MSDN 2014). Much like the *Files* app we discussed above, we began by replicating its basic functionality through the foundational grammar of action – move, copy, delete, rename – exposed through the `KnownFolders` API.

While we turn towards expanding this foundational grammar in chapter 6, it is worth pausing and noting here that for the first time in our research undertaking we felt we have arrived at a point where we were tackling the problem at a sufficient scale. Rather than concerning ourselves with the design of better user interfaces, or fighting with recalcitrant mobile materialities of information, we could now prototype a mobile datastore to reimagine the *relationship* between the OS, its datastore, and interfaces for acting on and sharing the data stored within it.

5.5 CONCLUSION

Even talking about files and mobiles, as we have done in this interluding chapter, and in lieu of the rich social practices we learned about in previous chapter creates a strange juxtaposition. Files seem outdated in our increasingly networked and mobile lives. But the file, however archaic and mundane it may appear in that juxtaposition, remains of central concern to system design. Attempts to resurface files on WP8.1 are, in a word, muddled. In this chapter we have attributed this muddle to a new button and underlying grammar of action that pervades our mobile lives: *share*. A second, major contributing factor is that the OS has by and large relinquished control over user data, creating proliferation of copies that designers of earlier systems sought to avoid (Smith et al. 1982). But where there is muddle, there is also opportunity for tidying. The *My Stuff* prototype datastore provides us with precisely the platform to further our investigation.

6 INTERFACING WITH THE CLOUD

6.1 INTRODUCTION

When you put contemporary smartphones into airplane mode, the device continues to work, albeit in a somewhat limited fashion. Thanks to its local datastore, you can still flick through your photos, check your calendar, listen to music, and catch up on older email threads. When you deactivate airplane mode, perhaps after a flight, the device inevitably starts buzzing and beeping: new emails start tumbling into your inbox, messages – *have u landed?* – start appearing, and you are notified that, for instance, eight people liked and three people commented on the photo you posted before you took off.

From an architectural standpoint this makes the mobile phone such a fascinating device. In airplane mode, we can see that the mobile phone is more than a portal to the Cloud. After all, we can still access the photos, music, videos, and other files that are important to us and that we keep on its local datastore. When connected to the internet, on the other hand, the mobile phone tethers[1] itself to the Cloud and all the services and connections it offers: to friends, family, or even strangers. It is as if the *wind of communication* is blowing through the device that manifest itself in the form of messages, notifications, beeps, and badges.

It is this 'tethering' that we turn our attention to in this chapter, for when people say that the mobile is a personal device, they are referring to not only the digital stuff that people keep in its local datastore, but also to the way the mobile interfaces with the Cloud to support and re-present interpersonal relations. This dual characterisation of the mobile – as being both personal and interpersonal – is both a driving force behind, and corollary of, the converging worlds of mobile, social, and cloud computing. Consider for instance, the photo that was snapped and posted to the Cloud before take-off. On the mobile the photo is part of the Gallery's `Camera Roll`. In the Cloud, e.g. on Facebook, the photo becomes part of Facebook's social graph, where friends can view, like, share, and comment on it. What are we to make of these photos? Are they still the same, or are they different but related in some other way? Looking at this scenario more closely, we can develop important implications for the design of the *My Stuff* datastore.

Speaking strictly technically, the two photos are *not* the same. While they may have started as copies of one another, on route to Facebook's social graph the photo is resized and compressed and loses around 94% of the attributes (e.g. Date Created/Modified, EXIF metadata, etc) that are associated with the photo-file in the local datastore (see, Thereska et al. 2013, 7). Since these photo-files are technically distinct from one another, we could reason that these two photo-files loosely correspond to the above dual characterisation of the mobile – where one photo is personal and the other is interpersonal/social. If, however, we'd also emailed or messaged the file to a family member, perhaps with an accompanying messages – *I'm coming home* – the situation gets complicated even further. In any case, alarm bells will be going off in the mind of the astute reader, as we are starting to transfer conclusions, derived from a technical[2] comparison of files stored in different places, to the meanings they embody for us personally and socially. While the personal vs. interpersonal distinction helped us to develop

1. For instance push notifications are sent through a persistent socket connection between the mobile and the cloud that is maintained/re-established through a combination of long-polling and keep-alive messages.

2. I could go on to talk about information theory as a branch of mathematics here. After all, as Dourish & Bell reminds us, information in the context of ubiquitous computing is a cultural category (2011, 193)

an initial understanding of the multifaceted and fascinating nature of the mobile, its usefulness is now waning.

6.1.1 *From Files to Possessions*

There is a nascent field of research within HCI that studies how people experience the stuff they keep on their personal devices[3] and in the Cloud (Odom et al. 2011; Odom et al. 2012; Harper et al. 2013; Lindley et al. 2013; Zhao & Lindley 2014; Odom et al. 2014). In our above example we have sometimes been using the term File[4] to refer to the photo. Odom et al., on the other hand, would refer to such a photo as a digital possession (2012). Both terms are, of course, referring to the same thing – the photo – but they differ in the way they orient us to that photo. The former term – *File* – orients us to technical aspects of the photo, allowing us to talk about where the photo is stored, how it is encoded, and its name, size, and extended attributes. The term *digital possession*, on the other hand, orients us to the social and moral dimension of the photo, where we might consider how people draw on their photos to communicate with their friends and ultimately craft their identities. Moving forward, we would do well to integrate such orientations, so that we can approach our problem from a perspective that balances technical with social and moral concerns. After all the files that users encounter on their mobiles[5] are precisely those that people would identify as their own: their photos and videos, music and podcasts, recordings and ringtones, documents and contacts.

Developing a functional understanding of Files & Possessions

As disparate as these two orientations seem on the surface, they overlap in the functional ways in which they approach their respective subject matters: Files and Possessions. If we revisit their foundational book on *Operating System Concepts*, Silberschatz et al. point out that "to define a file properly, we need to consider the types of operations that can be performed on files" (2013, 506).

In typical Computer Science fashion, the operations that can be performed on files are distilled to an essential and minimal set – the 'general case' – of file operations that an operating system needs to provide (Silberschatz et al. 2013, 506):

(1) **Creating a file**.

(2) **Writing a file**.

(3) **Reading a file**.

(4) **Repositioning within a file**, also known as a file **seek**, enables a program to read and write to different parts of a file.

(5) **Deleting a file**, which releases all file space as well as the directory entry.

(6) **Truncating a file**, which resets the file length to zero and releases all file space but keeps the directory entry.

Missing from this list are common actions that users might initiate on a file, such as copying and renaming (moving) a file. However, such actions can be re-written as combinations of the 6 primitive operations[6]. For instance to copy a file `Source` to the folder `Destination`, the Operating System first creates an

3. PCs and mobiles
4. In computer science a file is a "named collection of related information that is recorded on secondary storage" [...] and "is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file" (Silberschatz et al. 2013, 504).
5. Or that resurface as files in different locations, when for instance emailed.
6. Both the general case that consists of primitive operations and composing other operations from these primitives are typical of functional approaches.

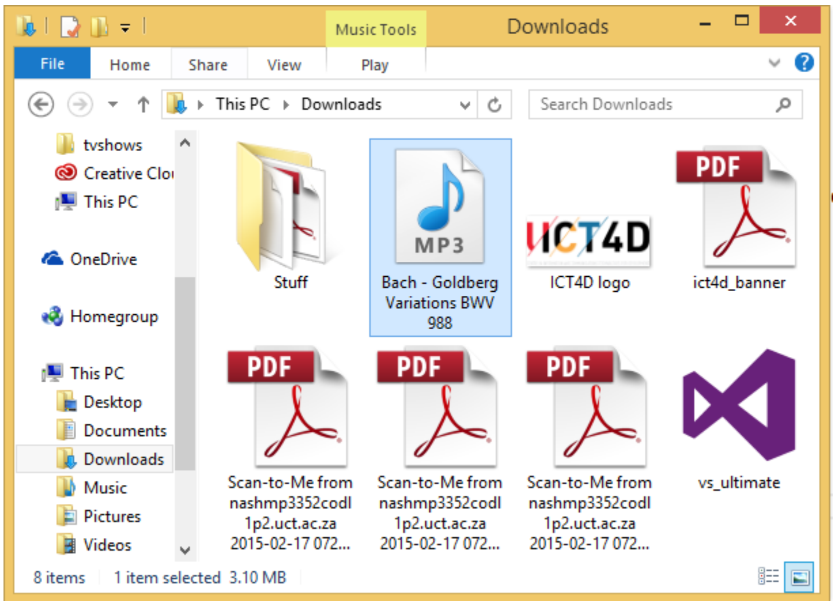


Figure 16: The Downloads Folder

empty file with the same name in the folder `Destination` and then reads from the file `Source` and writes to the newly created file.

A similar functional understanding in the area of digital possessions is put forward by Odom et al., who argue that "possession is at once a noun for a type of object (physical or virtual) and a verb that labels ways of treating things" (2012, 782). Researchers are unpacking these 'ways of treating things', but concede that possessing something digital: not only emerges through use (Zhao & Lindley 2014) and is often grounded in our understandings of physical possessions (see, Odom et al. 2012, 788), but is ultimately complicated substantially or even undermined when our digital possessions are stored in the Cloud (Odom et al. 2014). We ally ourselves to researchers who have pursued such lines of inquiry, and extend it by expanding its empirical focus to the mobile and how it interfaces with the cloud to explore how mobile/cloud architectures can support such notions of digital possessions.

Files & Possessions on the Desktop

Before we consider these complications in relation to the mobile, it is worth pausing for a moment to look at the place where files seem 'at home'— on the Desktop, where "the file system is the most visible aspect of an operating system" (Silberschatz et al. 2013, 503). Ever since Xerox designed the user interface to its revolutionary Star computer around the Desktop metaphor, digital objects have not only gained pictorial representations – *icons* – but users have also learned to think of their digital objects in physical terms – that they have a location (the Desktop or a Folder) and can be selected, dragged, and dropped (Smith et al. 1982). File icons draw upon the important role that place plays in our experience of material possessions and extend it to digital objects. For instance, just as I know that I can find my favourite pen in the front compartment of my backpack, I can find the piece of music I just downloaded in my Downloads folder (see fig. 16). Thanks to the file's graphical representation and almost physical properties that convey a 'strong sense of place' (see Banks 2014, 257), users can reason about their files in ways that are similar to how they reason about their material possession, whereas Odom et al. remind us, *knowing what you have* is entwined with *knowing where to look* (2012, 788). In short, file icons and folders have come to represent our 'stuff' – our possessions – on Desktop

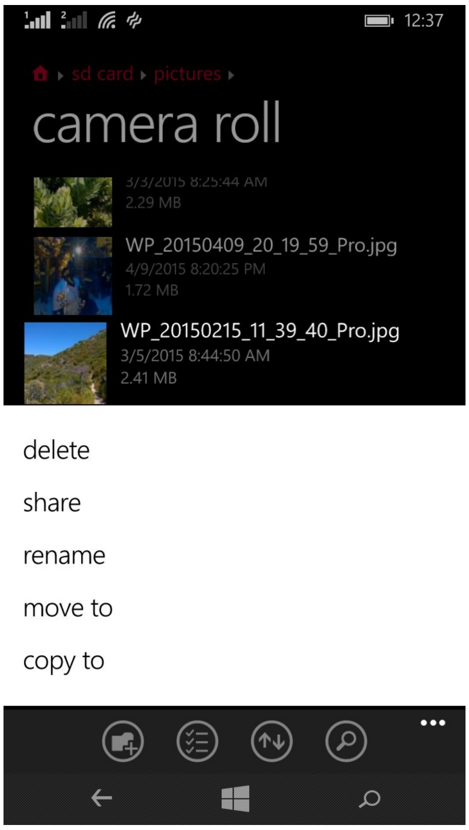


Figure 17: The standard set of actions on a mobile file manager.

computers. But as the following quote from one of the participants of study conducted by Odom et al. (2012) highlights, ownership of a file is complicated once it moves online:

> "*the more I talk about it, the more the idea of owning something online seems lost in translation*" (cited in Harper et al. 2013, 1135).

6.1.2 *Outlining a way forward*

Is our diminished or lost sense of ownership and control over our stuff an inevitable outcome of our increasingly interconnected digital lives? Or put another way, if we take seriously the insight that people think of their files, including those kept on their mobiles, as digital possessions, to what extent do current mobile and cloud architectures actually support, or undermine, such a treatment of Files?

On the mobile, we have shown that the file manager – My Stuff – that we developed in chapter 5 can support common actions of Desktop Shells, such as Windows Explorer or OSX Finder. These actions include, listing collections of or displaying individual files and folders on the phone & SD card, moving and copying files between folders, as well as deleting and renaming files & folders (see fig. 17). These standard actions, which are composed of one or more of the minimal file operations outlined above, operate within the boundaries of the phone's file system and behave, barring a system failure, in predictable ways. While these actions fall in line with notions of digital possession, they only do so in a narrow sense; that possessions are something you have. However within the broader sense of the term – where possession is a label for treating things – such actions no longer encompass what users seek to do with their stuff and that are enabled by the Cloud. In a way we are caught between two worlds. We can stay within the confines of the file manager, where digital objects are modelled on the physical things that surround us. Here we can speak of our files as possessions, in the narrow sense. Or, we enter the online world of the Cloud, where our files lose their Gibsonian affordances (Harper & Odom 2014) and become non-things (Flusser 1999) – possessions only in the very broadest of senses. Instead, we start referring to our stuff differently: as Experiences, Information, Data, or Blobs.

Contemporary mobile architectures position themselves as a portal into the cloud and attempt to translate between these two worlds through a single interface – *share* – that mostly moves in one direction: from the mobile to the Cloud. There is no question that the Cloud has enabled a plethora of ways to share and engage with our stuff socially. However once our stuff gets pushed through this interface and into the Cloud, the operating system and its file manager ceases to manage or look after it. The file in effect moves outside of our *awareness* and *control*. It is no wonder then that something gets 'lost in translation' (Odom et al. 2012). We can only conclude that it is impossible to consider how a file manager might support notions of digital possession without supporting social actions of sharing that interface with the Cloud and other devices and data stores. A careful, critical, and creative analysis of the possibilities and constraints of contemporary mobile and cloud architectures is necessary to better understand how we can support notions of possession when files are no longer just created and stored on individual devices, but also shared with others and travel through the Cloud. I conducted this analysis and developed the *My Stuff* prototype during a three month internship at the Human Experience and Design research group at Microsoft Research in Cambridge, UK, who have been leading investigations and developing prototypes that explore issues surrounding what possession means in the landscape of cloud computing (Odom et al. 2012; Harper & Odom 2014), social media (Lindley et al. 2013; Banks 2014), and personal devices (Harper et al. 2013).

6.2 CONTEMPORARY CLOUD ARCHITECTURES

To begin our analysis of how mobile and cloud architectures can better support notions of possession, we start with a simple scenario: giving a copy of a photo to a friend. This social action moves beyond the confines of the individual device – that of the sender – and implicates a second device – the receiver. To develop a solution that supports a treatment of possession within this scenario we will need to extend the grammar of action of the *My Stuff* file manager to support alongside common local actions such as `move to` and `copy to`, a new action `give to`. To orchestrate this act of giving we turn to the Cloud, a key technology of the contemporary computing on the Internet whose architectural style, as we will see in the next section, promise not only great possibilities but are also causing concern by changing the relationship people have with their digital stuff.

6.2.1 *Demystifying the Cloud*

The term 'the Cloud' mystifies as much as it informs. It is, as Reese (2009) points out, a canonical example of a buzzword that conveys an appearance of meaning without much actual meaning. But if we look at the technologies that make up cloud computing – namely, Web Servers, Databases, and the HTTP Protocol – we can see that these are the same familiar technologies of the Web. What distinguishes the Cloud from the Web, however, is its use of *virtualisation*. The fundamental idea behind virtualisation is to abstract away the hardware of a single computer (CPU, disk drives, memory, etc) into several *virtual machines* that run concurrently on a single physical computer (Silberschatz et al. 2013, 711). Each virtual machine is a separate execution environment: that is, it runs its own Operating System, which in turn might run an application such as a Web Server or Database. From the point of view of the virtual machine it appears to be running on its own private computer, even though technically speaking it is the virtual machine manager, or *hypervisor*, that creates and runs many virtual machines by providing a hardware interface that is identical to the physical host computer (Silberschatz et al. 2013, 712). In a nutshell, virtualisation technologies allow entire operating systems – running applications of their own – to run as applications within other operating systems.

Cloud computing applies virtualisation technologies to encapsulate and abstract away from lower level computing concerns and couples it with an attractive pay-as-you-go charging model. Within the cloud computing paradigm, computing resources of various kinds are made available to customers, who are referred to as *tenants* in cloud computing jargon, as a service using internet technologies. This phrase – 'as a service' – is a moniker that is used to differentiate between three general types of cloud services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)[7].

Infrastructure as a Service

As the name implies, *Infrastructure as a Service*, frees tenants from the burden of purchasing, creating, and maintaining their own physical IT infrastructure (Reese 2009, 18). Tenants instead rent virtual machines from Cloud Providers[8] and pay only for the amount of computing resources (CPU cycles, memory, disk storage, bandwidth, etc.) used. What makes the IaaS different from Web hosting companies of the 1990s is the ease and speed at which virtual machines can be requested and disposed of using either a web interface or API as well as the granularity (per minute, instead of per month or year) at which they are billed. If we wanted to develop an e-commerce site MothersDayGifts.com, to stay within the context of gifting, we could start doing so *today* using only a credit card and within minutes of requesting have access to and start developing on a small virtual machine running a full LAMP stack[9] that is connected to a high speed fibre network and costs about $0.05 per hour: no upfront costs and no monthly or yearly commitments required. In this example the Cloud Provider, such as Amazon EC2 or Microsoft Azure, owns and is responsible for reliably operating the physical computing hardware and network, whereas we, as the tenants, are responsible for everything from the guest operating system up (Reese 2009, 18): the database, web server, as well as the application and business logic of our site.

In our fictional example, we might expect demand to grow during the weeks surrounding mothers day. If we were to role out our own IT infrastructure, we would have to anticipate and purchase hardware and operating system licences for peak demand which might only last for 4 weeks; these computing resources would be largely unused for the remaining 48 weeks a year. With IaaS, on the other hand, it becomes possible to dynamically tailor computing capacity to run our site to the demand placed on it at any given time. For instance, during times of low demand our site could run on a single, small virtual machine, whereas during times of high demand – in the weeks surrounding mothers day – we could request more virtual machines and spread our site across them: for instance, one virtual machine or even two or three to run the web server(s) and another, more powerful virtual machine for the database. The major benefit of IaaS is that we would only need to pay for this increased capacity during times of high demand, saving money when demand is less. Since demand is something that web services can measure and log (i.e. hits per minute; or CPU load) and thus also act upon, many services that are deployed in the Cloud are configured in such a way that the services themselves can programmatically

7. see Support (2013)
8. The most popular Cloud Providers are Amazon, Google, Microsoft, & Rackspace.
9. Linux, Apache Web Server, MySQL database, Python/PHP scripting language

request additional or drop superfluous virtual machines depending on its needs. This ability to quickly and automatically scale computing capacity up or down to match demand is referred to as *elasticity* in cloud computing jargon and is often only possible within virtualised environments.

We can see three important properties of IaaS Cloud architecture at work: it eliminates the guesswork and upfront costs of acquiring IT infrastructure to run a service on the web; it allows services to elastically scale up or down to match demand; and a metered, fine billing granularity mean that tenants not only pay for the resources they use at any given time, but also that economies of scale and Moore's Law (1965) mean that prices are low and continually falling[10].

Platform as a Service

Just like with any physical computer connected to the internet, it is crucially important for the systems running within virtual machines to run up-to-date software and to have all the latest security patches installed. *Platform as a Service* (PaaS) frees tenants from dealing with such administrative duties by managing and providing a complete operational and development environment on top of the physical computing infrastructure (Reese 2009, 17). This all sounds good in theory, but let us return to our example e-commerce site, or more precisely the database that powers it, to see how it might benefit from running in a managed PaaS environment. On the above IaaS Cloud architecture the e-commerce database was running in its own virtual machine, where we, as tenants, are responsible for ensuring that all software updates and security patches are installed and that the database is running smoothly. This single instance, however, will neither scale nor replicate[11]. So we would either risk losing data or need to implement scaling and replication procedures ourselves. This, in turn, would require us to provision, maintain, and pay for a second virtual machine. Using PaaS Cloud architecture, we could instead store our database tables[12] on Google's Cloud SQL, Amazon's RDS, or Microsoft Azure's Table Storage. All of these PaaS offerings provide simple mechanisms to setup, develop, and scale a relational database that is replicated across data centres to ensure data integrity. Time consuming administrative and operational duties, such as patching and updating software, ensuring that proper software licences have been acquired, and writing the complex procedures to effectively scale and reliably replicate the database are all relegated to the operations teams at Google, Amazon, or Microsoft. With PaaS tenants can focus more on creating infrastructures and services and less on maintaining them.

Software as a Service

Examples of Software as a Service are platforms such as Google Docs[13], Office 365[14], and Salesforce[15] and are therefore not relevant for our research undertaking.

6.2.2 *Choosing a Cloud Service Model*

Now that we have an overview of the three main cloud service models – IaaS, PaaS, and SaaS – we need to choose service model to develop the 'cloud backend'

10. In their Cloud Platform Keynote, Google announced price cuts that effectively mean that their prices follow Moore's curve, adjusted by a slim operating cost overhead (2014)

11. Data replication across database instances is also used as a strategy to lesson the chances of data loss due to hardware failure

12. In our example these tables would contain information on products, customers, invoices, etc.

13. http://docs.google.com

14. http://office.com

15. http://salesforce.com/

for the My Stuff file manager. The choice really comes down to the former two, as within the SaaS model we would be unable to write our own APIs. Especially, for developers working on their own or within companies small and large the PaaS offerings of the Google Cloud Platform or Microsoft Azure are the most interesting. Consider 'Snapchat'. This novel photo sharing app has millions of users, and yet the company does not have an operations team of their own, because they developed their service on top of Google's PaaS offerings (*Google Cloud Platform Live: Keynote from Urs Hölzle* 2014). And it's not just Snapchat, for many developers PaaS model hits a 'sweet-spot', where they can still develop their own data-models and code their own APIs, but are freed from system administrative duties such as making sure that servers are running smoothly day and night as well as installing software updates and security patches (Simmons 2015). Since the main emphasis within the PaaS model is on creating new services and features and less on maintaining them, PaaS is a popular choice for teams that do rapid prototyping and adopting agile software development methodologies (Support 2013).

With such high praises, it should come as no surprise to the reader that we choose to prototype My Stuff's extended grammar of action of top of the PaaS model. The reader will also notice that in the preceding pages that we have mostly consulted developer oriented conferences, blog posts and videos; and deliberately allowed ourselves to become swept up in the hype and marketing of the Cloud, to experience for ourselves the promises (leaving aside for the moment if these promises are justified) of being able to create cheap, scaleable, easy-to-develop and maintain cloud services. Before we canonise these praises and promises, we should also consider the two main drawbacks of PaaS offerings (Support 2013). Firstly, when developers write their APIs against specific vendors PaaS offerings, their services tends to be locked-in to that vendor. Secondly, PaaS generally adopt standard technologies and work better with specific, albeit popular computer languages such as Python, JavaScript (node.js) and .NET; so, PaaS is generally unsuitable for projects using niche or domain specific languages or requiring a specific coupling between hardware and software.

In some ways these drawbacks are further motivation for choosing the PaaS model in general and helped us choose between the big three Cloud vendors: Amazon, Google, and Microsoft. Remember our motivation for leveraging the Cloud to extend our file managers grammar of action is to understand through our practical engagements – developing data models, APIs, storage services, etc – not only the the architectural style of the Cloud but also the interaction between the Cloud and the Mobile, two major and converging computing environments of our time. So we aren't interested in specialised hardware or niche software, but to develop against all those standard protocols (i.e. HTTP), solidified Cloud technologies (SQL), and developer best practices and principles (e.g. MVC, REST).

6.2.3 *Choosing a PaaS Cloud Vendor*

Although we considered all three big Cloud Vendors – Amazon Web Services, Microsoft Azure, and Google Cloud Platform, – we quickly eliminated Amazon Web Services because their focus is on IaaS offerings. We favoured Microsoft's Azure Mobile Service PaaS offering over Google's because it offers greater interoperability and is tailored to *mobile* app development. Azure Mobile Services (AMS) have positioned themselves as a one-stop for building backend services and datastores that support all major mobile platforms (Android, iOS, and Windows). It focuses on providing cross platform solutions to common mobile app requirements – how to register and authenticate users; how to send push notifications; and since users are increasingly expecting their 'cloud connected' apps to work even when offline, how to store, interact with, and synchronise cloud data with local databases and vice versa. Within all these scenarios, AMS emphasises interoperability, which means that almost by definition AMS adopts standard technologies (SQL Databases), protocols and data-interchange formats (HTTP & JSON), as well as established design patterns and styles (MVC & REST) to make the networked applications that run on top of AMS support common mobile platforms. Ultimately we choose the Azure Mobile Services as it is run on top of a reasonable cross-section of solidified technologies and developer best practices that will help us study and understand the subtle, but important ways these individual components are composed that ultimately affect the design of applications and services that people use everyday to store and share their stuff. These are complex topics that we will discuss in the coming sections, but for the moment we will focus on familiarising ourselves with Azure Mobile Services. But before we dive into these subjects, I again ask to be excused for the tedious and detailed accounts that follow. It is however necessary to describe and document the tedious to get at the subtleties.

6.2.4 *Getting Started with the Azure Mobile Service PaaS*

A testament to how easy it is to use cloud platforms is the fact that the most challenging part of getting started with Azure Mobile Services (AMS is giving a the service a unique name. All tenants using AMS share the same, global namespace, and many names are already in use or reserved. Similarly to how it is difficult for people with common names to find an appropriate and unused email address on popular providers such as gmail. Fortunately for us the name `mystuffservice.azure-mobile.net` was available. The remaining steps to create the service are straightforward:

1. choosing between one of the hosting zones US west, US east, North Europe, East Asia, and Japan West;
2. choosing between two backend languages/platforms, .NET and JavaScript/Node.js;
3. and finally, creating an a new SQL database server instance and setting its administrator account and password (or alternatively, providing the connection string, account, and password to an existing SQL database server).

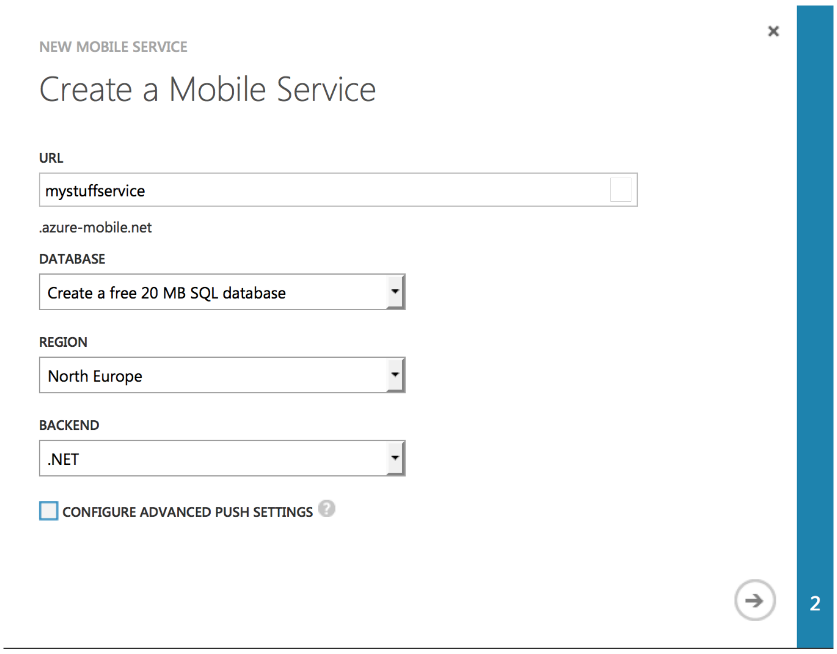
We choose North Europe as our hosting zone since at the time we developed the backend were living in the UK, and since we implemented the My Stuff mobile datastore in C#, we decided to stay within the .NET ecosystem on the server/service side as well (see fig. 18). Finally, we created and configured a new SQL database server instance and placed it in the same hosting region (see fig. 19).

After creating the service it takes a few minutes for it to become 'live'. Existing applications can then interact with the service by installing an AMS client library[16] that connects to the now live service using its web address and application key.

```
public static MobileServiceClient MobileService = new MobileServiceClient(
      "https://mystuffservice.azure-mobile.net/", // URL of mobile service
      "ijrtUPvkMy2iB7eFvuNiNmofZbILSN23" // application key
);
```

This client library exposes an API that can interact with the newly created AMS backend. This API is difficult to get working properly, since it mediates between client-side and server-side concerns. In practice this means that server side changes, especially those that affect the data model, often need to be met with congruent changes to the client side, and vice-versa. Without a properly structured code base such tightly coupled systems can grow in complexity and

16. https://www.nuget.org/packages/WindowsAzure.MobileServices/1.3.0 for Windows clients. However, similar libraries exist for Android and iOS.


NEW MOBILE SERVICE

Create a Mobile Service

URL

mystuffservice

.azure-mobile.net

DATABASE

Create a free 20 MB SQL database

REGION

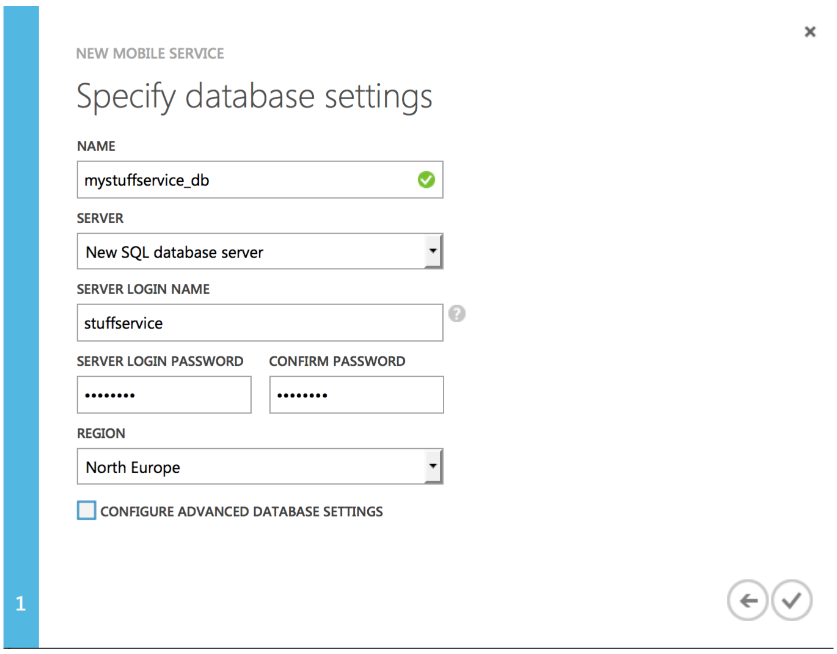
North Europe

BACKEND

.NET

☐ CONFIGURE ADVANCED PUSH SETTINGS

Figure 18: Creating, naming, and placing an AMS service.


NEW MOBILE SERVICE

Specify database settings

NAME

mystuffservice_db

SERVER

New SQL database server

SERVER LOGIN NAME

stuffservice

SERVER LOGIN PASSWORD

••••••••

CONFIRM PASSWORD

••••••••

REGION

North Europe

☐ CONFIGURE ADVANCED DATABASE SETTINGS

Figure 19: Creating, naming, and placing the database server and creating an administrator account.

are often difficult to maintain and debug[17], let alone develop in the first place. So a better starting point – especially for developers who, like myself, are unfamiliar with Microsoft's PaaS offerings – is to download the source code of a sample Todo app that connects a simple Todo list front end running on Windows Phone to a matching AMS backend. Todo list applications have evolved into a de-facto standard scenario to demonstrate the structure and syntax of data models, views, and controllers on the server side[18]. The AMS sample application, which for convenience already have the above server addresses and application keys filled in, can be downloaded from the portal page of the newly created service. Upon inspection of the source code we can see that the overall application – called a *Solution* in the Visual Studio IDE – consists of two separate components – or *Projects* – one for the server-side and another for the front end.

On the *server-side* a single, global Todo list is stored and queried through the `TodoItem` database table, whose scheme is defined by the `TodoItem.cs` entity data model class. Additional, internal fields such as the TodoItem's `Id` as well as `CreatedAt` and `UpdatedAt` timestamps are members of the `EntityData` parent class and thus are also part of the database table.

```
public class TodoItem : EntityData {
       public string Text { get; set; }

       public bool Complete { get; set; }
}
```

Client side applications can interact with the Todo list, defined by the above data model, through the API that the 'TodoItemController.cs' class exposes.

```
public class TodoItemController : TableController<TodoItem> {
    // POST tables/TodoItem
    public async Task<IHttpActionResult> PostTodoItem(TodoItem item);

    // GET tables/TodoItem
    public IQueryable<TodoItem> GetAllTodoItems();

    // GET tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
    public SingleResult<TodoItem> GetTodoItem(string id);

    // PATCH tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
    public Task<TodoItem> PatchTodoItem(string id, Delta<TodoItem> patch);

    // DELETE tables/TodoItem/48D68C86-6EA6-4C25-AA33-223FC9A27959
    public Task DeleteTodoItem(string id);
}
```

Looking only at above method signatures and comments of the controller class we can tell that the API is accessed through HTTP and that the endpoints and actions it exposes are the 'CRUD' (create, read, update, delete) operations (see Silberschatz et al. 2011) on the 'TodoItem' database table.

The *front-end* app of the sample makes use of the Azure Mobile Services client library to connect to and interact with the backend. When complied & deployed to a phone, the front end application fetches and displays Todo items and allows users to create new as well as check off (complete) existing Todo items. The app's source code isn't as neatly structured as the server-side code; it mixes calls to the

17. See for instance http://amy.palamounta.in/blog/2015/03/02/level-up-your-api-with-hypermedia/

18. See for instance the immensely popular TodoMVC (2015) project that implements the same Todo application on 64 different application frameworks, allowing for a more structured comparison between the frameworks. At the time of writing the project has been 'starred' 13,475 and 'forked' 7,403 times on GitHub.

web service with navigation and UI events. In the places where it is used, we can tell that it exposes the above server-side Web API as an `IQueryable` collection. This is a common interface also used by list data structures. So syntactically at least, enumerating (listing), inserting, and updating Todo items on a service and database running in a datacenter somewhere in Northern Europe is no different than operating on a locally stored list. Behind the scenes the library deals with object serialisation, de-serialisation, network calls, and caching so clients can interact with server data, stored on a database, as if it were local data stored in a list.

The server-side and front-end components in the sample application only cover the most basic scenario, where a single user interacts with a single, global Todo list; if we'd deploy the phone app to a second user's device, they'd see the same, global Todo list as the first user.

6.2.5 *Diving Deeper into Azure Mobile Services*

The AMS documentation explains how to extend this simple app with common requirements that people have come to expect from cloud connected mobile apps:

1. User Accounts/Authentication
2. Sync and offline support
3. Cloud Storage
4. Push Notifications

Accounts & Authentication

Account Registration and Authentication is a key component to enable multi-user support. Extending the sample app and service to support this feature is mostly done on the server side. The front end only needs to be expanded to provide a registration/login UI that make the appropriate calls to the client library. Restricting access to only those Todo Items that belong to the user is done in three steps:

1. Adding a `UserId` field inside our `TodoItem` data-model class, on both the server and client-side. On the server side this also adds `UserId` as a column in the `TodoItem` database table.
2. Annotating the `TodoItemController` class with a `[AuthorizeLevel(AuthorizationLevel.User)]` attribute that ensures that all operations against the TodoItem table can only be performed by an authenticated user.
3. restricting CRUD operations so they can only operate on those TodoItems that are owned by the user currently accessing the API.

Offline sync

The Azure Mobile Services client library supports what is loosely termed 'offline sync' (Malayeri 2015). The 'offline' part of the term refers to the ways in which clients can access, insert, modify, or delete data in the app even when there is no network connection by storing data in a local, offline database. This also serves to improve app responsiveness as server data is cached locally. The 'sync' part of the term refers to the ways these offline changes are synchronised with the online database and vice-versa. Because data and changes can be locally cached and synchronised, apps can not only support offline workloads but app environments themselves become portable and can be used and synchronised across multiple devices, each with their own offline, local cache that in the process of synchronisation pushes local changes to, and pulls remote changes from, the online database.

Offline sync support is implemented entirely on the client side, by storing and interacting with data in a local SQLite3 database. On top of this local database the client app defines tables, in our case a `TodoItem` table. The database is then used to initialise a `SyncContext` in the Azure Mobile Services client library. To synchronise changes the client app calls the following method, which first pushes local changes to the API server and finally pulls remote changes.

```
private async Task SyncAsync()
{
    await App.MobileService.SyncContext.PushAsync();
    await todoTable.PullAsync("todoItems", todoTable.CreateQuery());
}
```

Cloud Storage

The Cloud is not just a place to do computing, but increasingly also to store and share data. To cater for this need, Azure provides a storage platform called Azure Blob Storage (ABS) that has comparable features to Amazon and Google's Cloud storage offerings. The Todo list application demonstrates how a mobile app can upload to as well as access data stored in the Cloud. It does so by expanding the app to allow users to attach a photo to a TodoItem, for instance if a TodoItem is 'Buy Coffee' then the user could snap a photo of the particular brand or roast they want to buy. These photos, however, are not stored in the Database, but are uploaded to and then accessed from the Azure Blob Storage system. This simple scenario illustrates how data is managed and represented differently between mobile apps and cloud services. For on the mobile phone, when a photo is taken it is stored as a File with a particular name and in a Folder hierarchy. But once uploaded to Azure Blob Storage, the file ceases to be a File, because technically speaking Azure Blob Storage doesn't store files: it stores file data in the form of a *blob* (Myers 2015). Such quibbling over semantics is often symptomatic of the worst scholastic pedantry, but as we will show in the coming sections this subtle, often imperceptible difference, has enormous consequences.

Within many scenarios the subtle distinction between a file and a blob storing file data has no practical or observable consequences. But in order to understand how such differences can have larger emergent effects, we need to start by unpacking their similarity as well as difference.

Just like files are contained within folders, blobs 'live' in containers. However unlike folders, which can be nested inside other Folders, containers are stored at the same level. So while a storage account might have multiple containers, and each container can fold multiple blobs, a container can't hold another container. In other words, blobs and containers are flat, whereas files and folders are hierarchical. Additionally, containers are not just places that hold blobs but also control access. That is, a container dictates how and by whom blobs can be created, read, written to, and deleted.

We return to the Todo list application to examine the various components that are implicated in the various transformations that a files undergoes as it is uploaded to the Cloud. In order to use the Azure Blob Storage service the tenant must first create a storage account. To reserve the name for our future project we called this storage account `mystuffstore`, but for the moment we will be using it to expand the Todo list app. Storage accounts can contain an unlimited number of containers, where each container can store an unlimited number of blobs. The expanded Todo app/service, however, only uses a single container, named `todoitemimages`, and assigns each blob a `GUID` (globally unique ID) to avoid naming collisions in the global, flat namespace of the `todoitemimages` container. These assignments are carried out on the client side, as can be seen in the first few lines of the following method. The `todoItem` is then 'inserted' into the `todoTable`. This insert operation is carried out as an HTTP POST on the API server.

```
private async Task InsertTodoItem(TodoItem todoItem, StorageFile media)
{
    // Set blob properties of TodoItem.
    todoItem.ContainerName = "todoitemimages";

    // Use a unique GUID to avoid collisions.
    todoItem.ResourceName = Guid.NewGuid().ToString();

    // Send the item to be inserted. When blob properties are set this
    // generates an SAS in the response.
    await todoTable.InsertAsync(todoItem);

    // If we have a returned SAS, then upload the blob.
    if (!string.IsNullOrEmpty(todoItem.SasQueryString))
    {
        // Get the URI generated that contains the SAS
        // and extract the storage credentials.
        StorageCredentials cred = new StorageCredentials(todoItem.SasQueryString);
        var imageUri = new Uri(todoItem.ImageUri);

        // Instantiate a Blob store container based on the info in the returned item.
        CloudBlobContainer container = new CloudBlobContainer(
            new Uri(string.Format("https://{0}/{1}",
                imageUri.Host, todoItem.ContainerName)), cred);

        // Get the new image as a stream.
        using (var inputStream = await media.OpenReadAsync())
        {
            // Upload the new image as a BLOB from the stream.
            CloudBlockBlob blobFromSASCredential =
                container.GetBlockBlobReference(todoItem.ResourceName);
            await blobFromSASCredential.UploadFromStreamAsync(inputStream);
        }
    }
}
```

When the todoItem arrives at the API server, it first checks if the `todoitemimages` container exists. If not, it creates the container and grants public, non-authenticated read-only access to the blobs it contains. Using a Shared-Access Signature (SAS), the API server grants temporary write access to the container for 5 minutes. This signature is stored in the `SasQueryString` field of the `todoItem`, which is finally inserted in the database and returned in the request response to the client. The client app then uses the SAS credential to provision a new blob and upload the file's contents.

In our example, we might start with the file `D:\Pictures\WP_20150311_18_04_27_Pro.jpg`, a photo of the particular coffee brand we want to attach to the todo item 'Buy Coffee'. In transit, the file's content is read from a `DataStream` that is 'transported' through HTTP, which is written to the blob: `http://mystuffstore.blob.core.windows.net/todoitemimages/9b490ceb-8e99-41e4-9091-ee5aa13c47ee`. Here we can already see how a file, when uploaded to the cloud, can lose its name, extension, and place. Further still, we could argue that the local file loses its significance altogether, as the app only accesses the photo using the above URI. This is done in through the following XAML binding.

```
<StackPanel Orientation="Vertical">
    <CheckBox Name="CheckBoxComplete" IsChecked="{Binding Complete, Mode=TwoWay}"
```

```
                Checked="CheckBoxComplete_Checked" Content="{Binding Text}"
                Margin="10,5" VerticalAlignment="Center"/>
    <Image Name="ImageUpload" Source="{Binding ImageUri, Mode=OneWay}"
            MaxHeight="250"/>
</StackPanel>
```

This highlights two salient properties of the Azure Block Storage system. Firstly, ABS is located in a narrative of the scalable and de facto unlimited: whereas storage on the phone is expensive, volatile, and limited, ABS is cheap, virtually unlimited, and easily scalable. Secondly, blobs are optimised for streaming and can be accessed using a trivial UI binding.

Push Messages

Push messages are used to send small amounts of data (<4KB) from cloud servers to client apps. For instance, chat services typically use push messages to deliver messages and UI alerts to the user. Or the push message might alert the app that there is new data to be fetched from the server (Developers 2015). Basically push messages are responsible for most of the beeps, buzzes and badges on a smartphone.

For developers push notifications provide an important mechanism for architecting applications that rely on a two-way communication stream between client apps and API servers, that *both* client and server can initiate. Traditionally it is the client app that initiates communication with the server, for instance by making a HTTP request to an API server end-point; the server, in turn, answers the request with a HTTP response. Beyond responding to such requests, the server has no other way to send messages to the client app. With push messages, however, the server can initiate communication with the client by sending it a small message. The client, in turn, can 'respond' to such a message by making an HTTP request to the API server. Android, iOS, and Windows Phone provide push messaging services that work in approximately the same way by maintaining a single active socket connection open between a phone and push servers running in the cloud. All applications that make use of push messages use the same connection that is established and maintained by a system background service, so overhead and battery drain are minimised.

When an API server wants to send a push message to a client app, running for instance on Windows Phone, it sends a request containing the message and the phone's unique client app ID[19] to the Windows Push Notification Services' (WNS) server, as seen in fig. 20 (1). The WNS server then sends the message to the Notification Client Platform (NCP), a background service that handles push messages for all applications (2). The NCP looks at the message, matches it to the correct client application, starts that application and delegates it to the correct client application and starts that application (3). The client app might create a notification straight way from the data in the message or the app makes a normal HTTP request to the API server (4).

6.2.6 *Tying the components together*

We originally saw the Todo sample application as a quick way to learn about and develop a simple PaaS API server and a matching client app. Extending this sample app with user accounts/authentication, offline and database sync support, cloud storage, and push notification, however, transformed its purpose. The extended sample app, inspired us to think about how, for instance, Azure Blob Storage could be used to temporarily store and distribute files from sender to receiver or how push notifications could be used to notify participants and

19. This ID is obtained when the client app registers for push messages with the push severs and is forwarded to the API server.

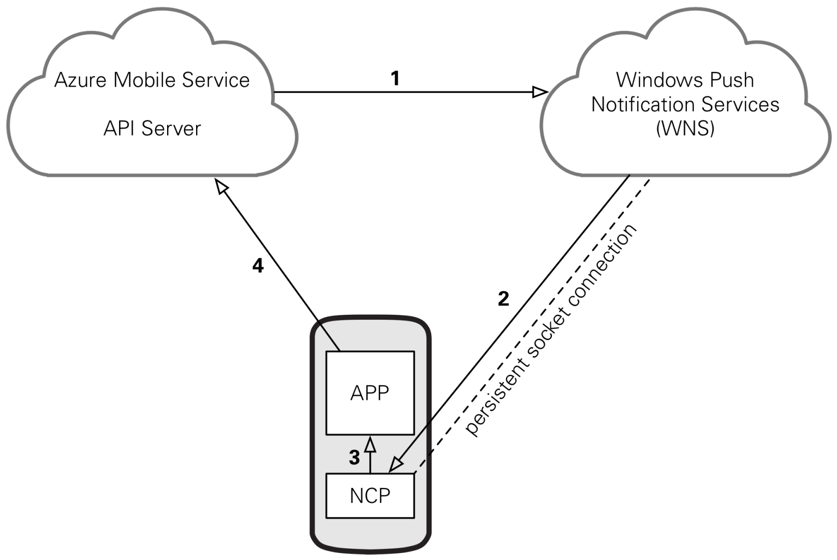

Azure Mobile Service
API Server
1
Windows Push Notification Services (WNS)
4
2
persistent socket connection
APP
3
NCP

Figure 20: Push notification architecture.

coordinate more complex acts of sharing such as gifting, where the file needs to be removed from the givers phone once the recipient receives it. So more than a contrived example, the extended Todo app/service became, for me, a kind of scaffolding for common mobile app requirements.

6.3 ARCHITECTING TO GIVE

We started with a most basic scenario. Giving a photo file (or rather a copy of a file) to a friend. Immediately this simple scenario leads us to consider three high-level questions:

1. How do sender and receiver identify each other?
2. How is the file uploaded to, stored on, and downloaded from Azure Blob Storage?
3. How does the original file change to reflect this act of giving?

In the next three sections, we develop technical solutions that respond to these questions.

6.3.1 *Accounts & Contacts*

Accounts serve two purposes. They must uniquely identify a person and restrict access to any data that is stored online to that person; and secondly, they must allow other users to find each other using only information they have already stored on their phones: namely, phone numbers. So in a nutshell, users identify themselves to the service and to each other through their phone numbers. This means that users can find other users using the phone numbers stored in their address books. The popular chat application WhatsApp exemplifies this approach, whereby when a user logs on to the app she can see and chat with all those people, stored in her phone's Address Book, who also have WhatsApp accounts. We created a User Account scheme, outlined in fig. 21, based on this approach of matching phone numbers.

Users are authenticated using their salted[20] and hashed passwords. The `UserId` field is further used as a foreign key to identify and restrict access

20. In cryptography salt is additional random data that is used when hashing password and helps defend against dictionary attacks.

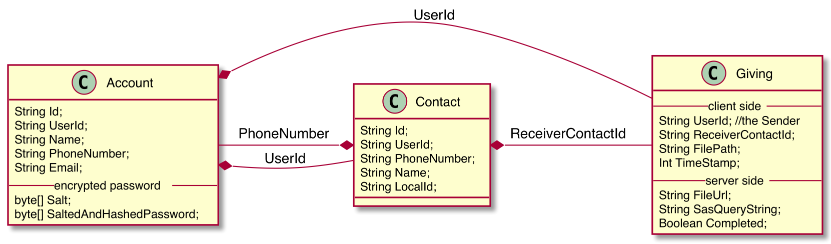


Figure 21: giving class diagram

to user data – for instance, a Contact stored in that User's address book – to that particular user. This foreign key relationship between a User Account and user data stored in other tables is one-to-many. This means that any User Account can have multiple Contacts – for instance one for each user's friends, colleagues, and family – but that every Contact 'belongs' to, and can only be accessed by, one particular user.

When a registered user wants to give a photo to a friend, our system creates two objects: a `Contact` object, which stores the friend's address book entry id and copies of the friend's name and phone number; and a `Giving` object, as seen in fig. 21. When the `Giving` object is first created only the `UserId`, `ReceiverContactId`, `FilePath`, and `TimeStamp` fields are filled by the client-side app and inserted into the local database. In this 'incomplete' state, a `Giving` represents an expression of intent that the user wants to send a particular file to the receiver. Next the local database is synchronised with the server to propagate the changes. Through this synchronisation corresponding objects are re-created on the My Stuff API server. The API server first validates the input, to check if a corresponding receiver account exists by matching the phone number the user signed up with.

6.3.2 *Uploading to, storing on, and downloading from Azure Blog Storage*

After this validation step the API server fills in the remaining fields of the `Giving` object of fig. 21, similar to the process we followed in context of the Todo list sample app. That is the API server generates a Shared Access Signature to allow the client app to create a blob and upload the file to a storage container. Upon successful upload of the file the server finds and/or creates corresponding objects that belong to the receiving user, the `Account`, the `Contact` of the sender, and a `Receiving` object. This `Receiving` object is identical to the `Giving` object but belongs to the receiving user and replaces the `ReceiverContactId` with the corresponding `GiverContactId`.

Next the API server sends a push notification containing the `Receiving` object, upon which the client app downloads the file from the Azure Blog Storage using the `FileURL` field, stores it at a specific FilePath – say `D:\Pictures\ReceivedPhotos\WP_20150512_13_02_33_Pro.jpg` – marks the operation as complete, and finally initialises a synchronisation with the API server, which in turn marks the `Giving` as complete and sends a push notification to the giving user whose client app initialises a final synchronisation to complete the exchange.

Tidying up

It is difficult to resist the temptation to retain the photo on the Azure Blob Storage for safekeeping and my initial instinct was as well to give in to this temptation.

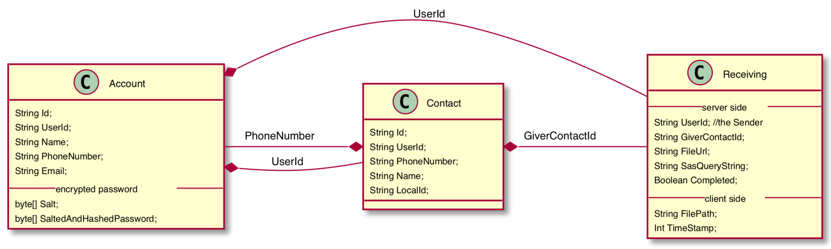


Figure 22: Receiving class diagram

Most Cloud storage providers make impressive reliability guarantees that make secondary storage on the mobile phone seem downright volatile. However, upon further reflection that we turn to in the next section I decided instead to not muddle up giving with safekeeping. And so implemented and scheduled a daily batch job to check for corresponding `Giving` and `Receiving` objects that have been marked as complete or whose initial timestamps are older than a month and remove the matching photos stored on Azure Blob Storage.

6.4 EXTENDING FILE ABSTRACTIONS TO INCORPORATE METADATA

While the first two emerging questions deal primarily with orchestrating the pragmatics of the exchange, the final question asks if and how the original file that is given and the one that is received should change to reflect this act of giving. We elevate the discussion that pertains to this question to a dedicated heading for it deals with the architecture of mobile datastores and speaks directly to how new file abstractions that encompass metadata (see Harper et al. 2013) surrounding social exchange are needed to better support notions of digital possession especially when these possessions move through, and are potentially undermined by, the Cloud (see Odom et al. 2014).

These are lofty goals, so it is again worth pausing for a moment to reflect on how best to approach this problem space. Richard Harper gets to the heart of the matter in the introduction to his book *Trust, Computing, and Society*, where he draws together diverse views and research cultures:

> when designers approach the "problem space" of the Cloud, for example, they have to consider the ontological "sense" that their designs provide. Do users come to trust in their relations to their digital "stuff"? Does that trust resonate with how those people get on with their lives, lives that are suffused with digital materials but that are, also, essentially about material properties that stand as part and parcel of their human endeavors? These are anthropological concerns, as much as they are technological and design questions.
> —Harper (2014, 12).

New file abstractions, in this context, should make visible those "material properties that stand as part and parcel of their human endeavors". And speaking of such endeavours specifically in anthropological terms as the above quote urges us to do, we look back to earlier chapters where we discussed human practices of gifting and sharing. Further, we are again inspired by Tim Ingold arts of noticing through which he boldly claims that "the properties of materials, in short, are not attributes but histories" (Ingold 2011, 32); and how, in a later book review, he draws on the seminal work of the British anthropologist Victor Turner (1967)

Figure 23: Magnet attracting iron filings as a metaphor for new file abstractions.

to explain "the capacity of certain things to attract meanings like iron filings to a magnet – meanings that by no means substituted for the things themselves but rather augmented and enriched them without limit" (Ingold 2014, 517) as seen in fig. 23.

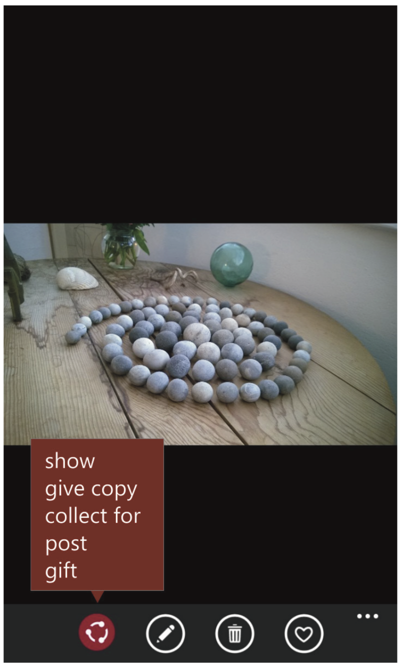
Looking at fig. 23 we can still recognise the objective core, which in our case would correspond with the original photo file. But we can also draw parallels between how the social doings associated with that file – that is, who we showed or gave a copy of that file to and where it came from – contribute to its social history and become *material properties* which are attracted and attached themselves to the objective core, much like iron filings.

We thus leveraged the *My Stuff* prototype to expand on the mobile sharing interface that we problematised in earlier chapters to allow users to express precisely those nuances, of how they wish to share their digital stuff, that are lost when sharing is equated with copying. Through this expanded interface *My Stuff* also explores new file abstractions that can evolve to represent the social life the file gains when it is given, gifted, or collected for others, with the goal of privileging those properties – or rather histories – of files that are salient to human affairs so that people can again have confidence in the provenance of their digital stuff:

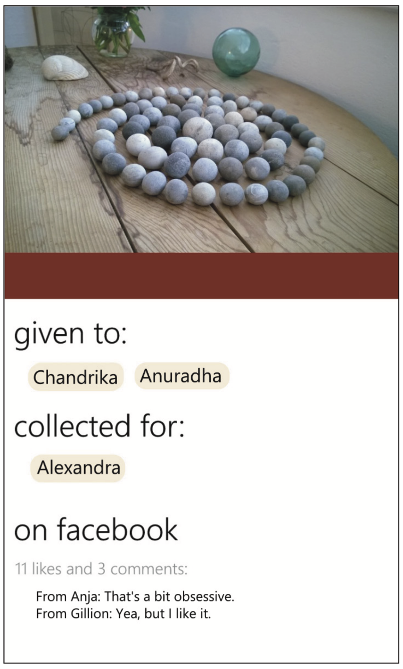
- Where did the file come from?
- Where do its copies reside?
- What is happening to those copies?

We began by iteratively sketching such an expanded sharing interface and corresponding file abstraction by showing, critiquing, and improving nascent design sketches with the Human Experience and Design Group as well as with its research interns at Microsoft Research, who collectively have made pioneering contributions to the nascent research area of digital possession. The emergent sketches we settled on and that I subsequently prototyped are seen in fig. 24.

The main idea behind these sketches is to reimagine the relationship between the Operating System, its datastore, and sharing interfaces to give everyday people better *awareness* and *control* of their digital possession. On the PC of the 90s, time-appropriate file abstractions worked in partnership with the Desktop metaphor to give and reinforce the Gibsonian affordances of the physical world that form the basis of how people reason about their stuff: that I know I have some photos from a skiing holiday, or bank statements for that matter, by virtue of being able to see them filed away in a 'Skiing Holiday' or 'Bank Statements' folder. In short "place and ownership go hand in hand" (Harper & Odom 2014, 284). But in the mobile and networked world of today having and especially sharing mean that storage is de-facto distributed (see Dijck 2008, 68). Even



(a) Expanded grammars of actions: beyond share.


given to:

Chandrika Anuradha

collected for:

Alexandra

on facebook

11 likes and 3 comments:

From Anja: That's a bit obsessive.
From Gillion: Yea, but I like it.

(b) New file abstractions that highlight the social histories of a file.

Figure 24: Sketching and implementing expanded grammars of actions and file abstractions.

as the OS might need to relinquish control over a file in order for the user to give or show it to distant friend. It should at least retain awareness of where initial copies of a file are located, as awareness of place is an important aspect of ownership. These My Stuff sketches give users an indication of where their stuff is, but also enriches and expands on the original object to also encompass the social life it has since gained.

In the following section we explain how we prototyped the MY STUFF system in form of an app. But we believe that it would provide substantial benefits to the user if the Operating System take on this expanded role, especially as the major Operating System providers such as Apple, Google, and Microsoft all offer a form of converged Cloud storage through their respective iCloud, Google Drive, and OneDrive platforms. Through such an expanded role the OS would in effect change the unit of engagement from files across various devices to something that the user thinks of as their digital possession with all that that implies.

6.5 IMPLEMENTING THE EXPANDED MY STUFF PROTOTYPE

We initially tried to implement the *My Stuff* prototype using the recommended system design pattern of MVVM that served us well when we prototyped the co-present photo gallery of Chapter 4. We however quickly found that the APIs that exposes a file's properties such as its size and date were difficult to grasp. And by grasps, I mean more in its mechanical sense of getting a hold of a thing rather than in the cognitive sense of understanding something. On Windows Phone 8 we could access these properties using the common dot notation, but on Windows Phone 8.1 that same property can only be accessed through an asynchronous future, as the simplified code-snippets of lst. 6.1 illustrate.

Listing 6.1 Photo Properties become Futures on WP8.1

```
//Windows Phone 8
foreach (var picture in album)
    Debug.WriteLine("Picture taken on " + picture.Date); //Accessed through field

//Windows Phone 8.1
foreach (var photoFile in photoFolder)
    var imageProperties = await photoFile.GetImagePropertiesAsync()

    Debug.WriteLine("Photo taken on " + imageProperies.DateTaken)
```

The main difference between these two snippets, besides the renamed classes (`Picture` vs `StorageFile` and `PictureAlbum` vs `StorageAlbum`), is that image properties such as the photo's date can only be accessed after an asynchronous method call. The `await` keyword is subtle and crucial here as it indicates to the programmer that this method won't necessarily return a result instantaneously but that it promises to return a result in the future:

> A *future* is a stand-in for a computational result that is initially unknown but becomes available at a later time. The process of calculating the result can occur in parallel with other computations. The Futures pattern integrates task parallelism with the familiar world of arguments and return values[21].

This creates a fascinating juxtaposition, whereby material properties are considered histories when approached anthropologically, but are modelled as a `Future` when trying to handle them programmatically. As a computer scientist I'm sympathetic to the needs of increasing parallelism especially within the limited computational resources of the mobile devices. However, the unfortunate and difficult side-effect of working with such *futures* are that without major workarounds it is impossible to bind to the results of such asynchronous methods from the User Interface.

Accessing and displaying the date the photo was taken, as I did in lst. 6.1, is a simple, self-contained example of this phenomenon that still encapsulates a range of similar difficulties that I was consistently confronted with. For instance, the following is a more realistic scenario: listing all image file names in a folder and displaying the image thumbnail alongside each one. Being able to bind to a property directly or via a simple converter as we did in Chapter 4, is crucial to effectively utilise limited memory, as this shifts responsibility to the list controller for loading thumbnails into a cache and just as importantly invalidating that cache[22] as the user scrolls through the list. This delicate interplay of loading new content into memory while simultaneously freeing up memory is crucial to enabling mobile experiences that don't stutter and stock and prevent crashes caused by out of memory errors. Being able to effectively utilise built-in components that have been stress-tested and verified is preferable to rolling out a custom solution which often ends up being far less robust.

Initially I tried the latter with varying success and ended up with a solution that conflated and tightly coupled together UI state (i.e. the current scroll position in the list), events (i.e. such as when individual thumbnails become available or when a user scrolls the list), with views (the page that displays the list) and models (the individual photos/files). By introducing and responding to events that trigger state changes, which then need to be kept track of, I had to sacrifice clarity in the app architecture. That is, the app worked but would occasionally

21. https://msdn.microsoft.com/en-gb/library/ff963556.aspx
22. Cache invalidation is famously considered one of the two hard problems in computer science. (See: https://martinfowler.com/bliki/TwoHardThings.html)

skip a thumbnail or would crash if the user scrolls too quickly. The paradox is that it was far more difficult to implement the logic to display a photo from the local filesystem than one stored at a remote URL. The former requires manually querying asynchronous data sources and being directly responsible for its results. That is we had to manage the lifecycle of the results – notifying the UI when results are available and clearing up the result from memory when it is no longer required – and all the while being responsive to UI scrolling events. Display a photo stored at a remote URL, on the other hand, only requires a simple binding from the `Image` UI Control to the URL, as illustrated below. The OS handles, and is responsible for, the rest.

```
<Image Name="Photo" Source="{Binding PhotoUrl}">
```

Despite these occasional glitches the system worked well enough to continue prototyping the interface between mobile and cloud datastores and services.

6.5.1 *Implementing expanded file abstractions*

Next to querying the filesystem, the My Stuff app also queries the local database which it keeps in sync with the corresponding database hosted on Azure Mobile Services. That is, whenever the *My Stuff* app encounters a file it checks the local `Giving` and `Receiving` database tables if it has a record associated with the file of the same path. If it does, we populate View Models with all relevant information. User Interface Views bind to this View Model and display the corresponding information in the UI whenever it is available, as can be seen in Fig. 24b.

6.6 CONCLUSION

In this chapter we conducted an analysis of contemporary cloud architectures, we developed a simple cloud service and companion mobile app following tutorials and best practices to better understand the interactions and transactions between the mobile and the various technologies that make up the Cloud. We then applied the lessons we learned from this analysis to the design and development of the *My Stuff* prototype. The design was also shaped through countless interactions with researchers and fellow interns at the Human Experience and Design group during a three-month internship at Microsoft Research in Cambridge. I tested the prototype in Cambridge across a variety of different devices and in different settings – at work, at home, in the pub, and on the bus – to ensure that prototype works as expected. After completing the internship I returned to Cape Town, to evaluate and further develop and refine the prototype and the theories and approaches it embodies (see Bardzell et al. 2015).

7 RELOCATING & REARCHITECTING

7.1 INTRODUCTION

In the previous chapters, we have expanded the *My Stuff* prototype to leverage new APIs introduced in WP8.1 and to interface with the Cloud through a service we implemented on the Azure Mobile Services platform. While the perspectives and experiences we gained studying and designing for communication and sharing practices in rural and urban South African contexts informed the design of the *My Stuff* prototype I developed during a three-month internship at Microsoft Research in Cambridge, UK, I also extended the empirical focus of this research undertaking to include the Cloud – or rather to include the technologies, design patterns, and engineering (best) practices, that are rendered invisible and immaterial by the Cloud metaphor. After the internship, with a functioning prototype in hand, I returned to Cape Town, with the goal of evaluating the prototype through a user study, but quickly discovered that the prototype didn't quite work in the same way in Cape Town than it did in Cambridge. In this chapter I instead interrogate and further develop the *My Stuff* prototype by investigating the subtle, but salient, differences that technologies are confronted with when they travel and the consequences these differences have for this research undertaking: to design tools for and better understand what it means to possess something digital in the age of the mobile and the cloud.

7.2 BACKGROUND

This relocation coincided with efforts to formalise two research groups at the Centre in ICT for Development at the University of Cape Town, which I had the privilege of attending and shaping:

> In our *Networks* research group we design innovative architectures that make communication systems more flexible and accessible in developing countries. Through simulation and experimental techniques, we tackle a range of network performance and service provision issues, including how to improve network performance and resource utilisation to make the most out of developing regions' limited network infrastructure and expensive bandwidth. On a small scale, we focus on localizing networking technologies through community wireless mesh networks and community cloud computing. On a large scale, we explore Internet performance engineering through software defined networking.
>
> As digital technologies facilitate and expand online – and offline – creative capabilities, our *Creative Digital Media* research group extends digital participation among young creatives. As many of these technologies are designed in the context of abundant material resources, they are often a poor fit for young creatives from resource-constrained backgrounds. Through a variety of methods, technical and creative, we explore these relationships and tensions, with a focus on implications for practical designs. This research, at the intersection of creative arts, anthropology, information technology, and media studies, is driven and inspired by a human-first approach.

Although these groups were respectively lead by computer scientists and media & communication scholars, we quickly discovered overlaps and shared interests, albeit coming from different perspectives, and began to form a latent community of purpose that proved fertile and stimulating place to discuss and relate my research.

Through their in-depth studies, communications scholars have for decades provided evidence that there is no such thing as a singular, global Internet (Miller & Slater 2001; Donner 2015). These studies tend to see the Internet as social and cultural phenomenon and focus on issues that surround access, adoption, and appropriations of "the Internet". Without discounting the invaluable contributions these studies make, few have considered in much depth what we might call the technical stuff that undergird the Internet: the cables, plugs, routing and peering arrangement, disks and data-centers, protocols and servers. Nowadays we are probably more inclinded to call this complex arrangement of stuff "the Cloud", as we did in the previous chapter, but the same gap between real and virtual, between moniker and material, still exists. However as Hu points out, "the gap between the real and the virtual betrays a number of less studied consequences, some of which are benign and some of which are not" (Hu 2015, x). It is these consequences – benign or not – that we turn to in this chapter. In doing so, we follow in the steps of commentators like Hu (2015) and Dourish (2015a) but extend their work by considering how location affects how we use – and don't use (see, Baumer et al. 2015) – the Cloud.

Relocating with the prototype that I developed at Microsoft Research in Cambridge, UK back to the University of Cape Town, I was following precisely those information and innovation flows that Lucy Suchman characterizes as both dominant and asymmetric (2002). They are dominant in the sense that centres of research & development, such as the Microsoft Research Lab in Cambridge or those found in Silicon Valley, "maintain a disproportionate hold over the distribution of at least computer-based information technology" (Suchman 2002, 139). And they are asymmetric in the sense that information technologies flow outward to be adopted and appropriated elsewhere. While developing the prototype in Cambridge, we did not have to contend with effects that mostly surface around the margins and edges of the Internet – most notably cost and latency – and which we would discuss at length and in depth during seminars, meetings, reading groups, and chats within and between the networking and creative media research groups.

7.3 FINDINGS

A primary motivation for developing the My Stuff prototype was to convey a stronger sense of ownership and control over the stuff that is stored on the mobile. Access and seeing an overview of what you have is a crucial first step in that direction, as we saw in the previous chapter. However these aspects were significantly influenced by not only how fast your connection to the Internet is, but just as importantly, where you are located on the Internet.

7.3.1 *Latency*

When I first launched the My Stuff app upon returning to Cape Town, it took noticabley longer for the app to launch. About five seconds. Even on a strong and, especially for South African standards, fast 3G connection. Switching to a 2G network would cause further delays (about seven seconds) and sometimes the app would crash.

In the previous chapter, I documented how I had followed the design guidelines and tutorials of the Azure Mobile Service. While developing the app in Cambridge I was mindful of the South African context I had come from and made sure that the app works when not connected to the internet. To be sure, if I put the phone in Airplane mode, the app would only take a second or two to

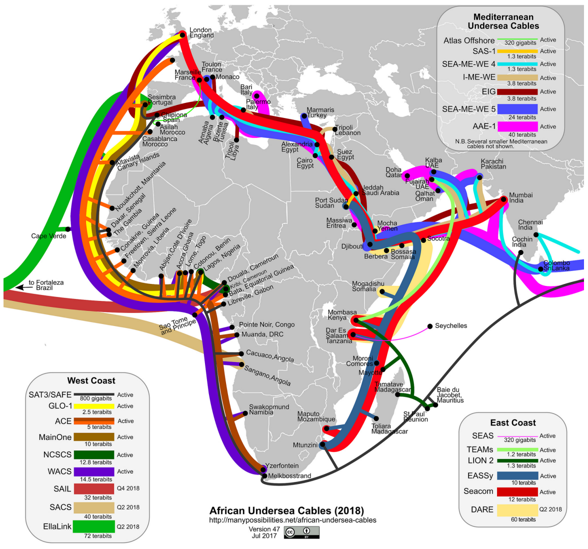

Mediterranean Undersea Cables
Atlas Offshore 320 gigabits Active
SAS-1 1.3 terabits Active
SEA-ME-WE 4 1.3 terabits Active
I-ME-WE 3.8 terabits Active
EIG 3.8 terabits Active
SEA-ME-WE 5 24 terabits Active
AAE-1 40 terabits Active
N.B. Several smaller Mediterranean cables not shown.
West Coast
SAT3/SAFE 800 gigabits Active
GLO-1 2.5 terabits Active
ACE 5 terabits Active
MainOne 10 terabits Active
NCSCS 12.8 terabits Active
WACS 14.5 terabits Active
SAIL 32 terabits Q4 2018
SACS 40 terabits Q2 2018
EllaLink 72 terabits Q2 2018
East Coast
SEAS 320 gigabits Active
TEAMs 1.2 terabits Active
LION 2 1.3 terabits Active
EASSy 10 terabits Active
Seacom 12 terabits Active
DARE 60 terabits Q2 2018
to Fortaleza Brazil
African Undersea Cables (2018)
http://manypossibilities.net/african-undersea-cables
Version 47
Jul 2017

Figure 25: Africa's undersea cables

launch, depending on how many other applications were running at the time. About as long as the *Files* app that we discussed in Chapter 5. This made sense, since when the app first launches it checks if there is internet connectivity. If there is, the app initialises and synchronises the database. If there isn't, the app only initialises the database but skips the synchronisation process. So we have found the cause of the delay: the synchronisation step.

Synchronisation is handled by the Azure Mobile Services client API through two methods – `PushAsync` and `PullAsync` – that we described in the previous chapter. In a nutshell these methods first identify the records that have updated since the last synchronisation event. Then the Azure Mobile Services client API issues an HTTP GET request for each new or updated record on the server side and an HTTP POST or PATCH request for each new or updated record on the client side, respectively. An HTTP DELETE request is issued for records that have been deleted. Setting aside the calls that are made to the API server to establish which records have changed, the AMS client API issues one HTTP request per record that has changed.

The round trip delay for a data package to travel between Cambridge and Dublin, where our API server is located, is about 10ms-15ms. Between Cape Town and Dublin that round trip delay increases to 200ms-250ms. Had I situated the Azure Mobile Services instance in the USA instead of Ireland, these latencies climb to as much as 350ms. Oftentimes these latencies compound when, as in our case, multiple requests need to be made and processed. Or in the case of HTTPS, three roundtrip exchanges are need for the handshake, before the actual request is made. For reference, humans can detect visual stimuli within 150-200ms (Amano et al. 2006), so these delays are noticeable and significant. And they are also unavoidable. For internet transmissions between continents are enabled through undersea fibre-optic cables (see fig. 25), and these transmissions are limited by the speed of light.

7.3.2 *Intermediaries*

There are, however, strategies to lessen the effects. Most notably through intermediary content caches and proxies. The University of Cape Town, for instance, uses an HTTP proxy server to (1) monitor, shape, and limit how much bandwidth students are using and (2) cache content that is requested and serve it to subsequent requests to speed up access and conserve bandwidth. In Cambridge, I had direct (unproxied) access to the Internet.

In earlier chapters we have already seen how our gifting fiction exposed the way in which the representational forms through which data is primarily encountered on Windows Phone and Android platforms – as a stream of data, rather than as a file – constrain the manipulations that are possible on that data. For instance, that it can't be deleted! In Cambridge, I had implemented this gifting case study on the My Stuff prototype and by relocating to Cape Town and experiencing this different access type, the gifting scenario again surfaced subtle but consequential effects.

To increase efficiency I made a couple of changes to the implementation of the gifting action on the API server, compared to how I implemented the action in Chapter 2. Instead of modeling giving a gift as two discrete steps, (1) creating the gift and (2) uploading the file, each requiring a separate HTTP request, I combined them into a single request that I outlined in simplified form.

```
# HTTP Resquest to Send Gift with JSON body and contents of file (selfie.jpg) to be gifted.
POST http://mystuffservice.azure-mobile.net/api/gift
Content: application/json
{
  "userId": "thomas",
  "receiverId": "alex",
  "filename": "selfie.jpg"
}

Content-type: image/jpg
#Stream content of selfie.jpg
```

Upon receiving the request the server responds:

```
# HTTP Response to sender (thomas)
201 (Created)
{
  "giftId": "t1",
  "userId": "thomas",
  "receiverId": "alex",
  "filename": "selfie.jpg",
  "status": "pending"
}
```

And sends a push notification to the receiver, notifying them of the gift that awaits them.

```
# Push notification to receiver (alex)
{
  "giftId": "a1",
  "userId": "alex",
  "senderId": "alex",
  "filename": "selfie.jpg",
  "accessToken": "secret"
}
```

Following this pattern, I also changed the way in which gifts are received. Instead of first requesting and downloading the file and subsequently acknowledging its receipt in a separate request, I combined both actions into a single HTTP request:

```
// HTTP Resquest to receive a gift with JSON body
GET http://mystuffservice.azure-mobile.net/api/gift
Content: application/json
{
  "giftId": "a1",
  "userId": "alex",
  "senderId": "alex",
  "filename": "selfie.jpg",
  "accessToken": "secret"
}
```

The corresponding server responses are shown below:

```
# HTTP Response to receiver (alex)
200 (OK)
# Data stream of contents of selfie.jpg file
```

This time the server sends a push notification to the sender so both the server and the sender can delete the file.

```
# Push notification to sender (thomas)
{
  "giftId": "t1",
  "userId": "thomas",
  "receiverId": "alex",
  "filename": "selfie.jpg",
  "status": "received"
}

# Server & client delete files
```

By combining these actions into a single request for sending and receiving a gift, I was trying to treat the webserver like a network switch, whereby the API server upon receiving the GET request, authenticates it, locates the corresponding blob reference and streams the content of the blob to the client app, which in turn saves the stream to a file. Finally, the API server marks the transfer as complete to:

(1) ensure that future request to the above URL return a '404 Not Found' error.

(2) so a periodically running server-side task can delete all blobs that have already been downloaded.

(3) to notify the sender to delete the file.

I had mapped giving a gift onto the HTTP POST verb and receiving a gift onto the HTTP GET verb. This seemed natural and worked well in Cambridge, where my access to the internet was direct and unmediated. However, in Cape Town my internet access was mediated by a proxy. For the most part, proxies are transparent and do their job well; they speed up requests by serving cached responses and conserve limited and expensive bandwidth in the process. I had, however, failed to properly consider the semantics embedded in specific HTTP verbs. A popular discussion initiated by Evert (2009) on the programmers forum StackOverflow, with over 1000 upvotes and 300 stars, gets to the heart of the issue. While a GET request – just like any HTTP request – can have a body, server side semantics for GET require that the body have no semantic meaning for the request. My mistake was to require and derive meaning from the body. This mistake was of no consequence in Cambridge. But the proxy server in Cape Town had a strict interpretation of the HTTP specification and dropped the body when it forwarded the request to API server, which in turn didn't know what to do with a body-less request and returned an error.

I account for this mistake here to learn from it, but also to show that such mistakes are part of programming. However the deeper issue is that in this case good infrastructure – fast, direct internet access – masks problems with the architecture – inappropriate choice of HTTP verb.

7.3.3 *Discussion*

The issues surrounding latencies that my relocation had surfaced were not isolated to this research undertaking. To the contrary, they were a pervasive theme in both the Networking and Creative Media research groups and deserve more unpacking.

Consider the participants in Alette Schoon's study of hip hop artists in a South African township in Grahmstown and the ways in which they create and share their tracks using Bluetooth and through Cloud-based filesharing hosts (2016).

> Bluetooth was seen as the easiest way to make an impact on a local level in distribution of music. Some hip-hop artists simply adopted a method of non-intervention, sharing the track with a few friends and relying on them to distribute it via Bluetooth, while others took a more active approach, and would 'send out a lightie', a young boy, to go and 'make his rounds', walking around various streets engaging with young people, actively sharing the track with anyone he could find via Bluetooth. –(Schoon 2016, 106)

To reach a broader audience, artists in Schoon's (2016) study also uploaded their tracks onto the Datafilehost platform and shared links on social networking sites and using messaging apps. Part of the allure of online sharing is the metadata[1] that surrounds the file:

> Datafilehost displayed one crucial bit of information that the hip-hop artists would frequently visit the site to check: the download counter which showed how many times the file had been downloaded. Hip-hop artists would get incredibly excited once the downloads started approaching 500, as this signified a hit. Rymgees was particularly proud of the fact that his one song had more than 6000 downloads. –(Schoon 2016, 110).

What artists were unfortunately unaware of is how files, along with their download counters, are automatically removed after 90 days of inactivity. It only takes a moments reflection to understand, why artists aren't drawn to more permanent hosting platforms such as Soundcloud or YouTube, since they force listeners to incur data costs every time they stream and listen to a song/video. Despite such mismatches, participants in Schoon's (2016) illustrate creative appropriations and vernacular forms of design-in-use[2] where audio files are created on backyard computers and shared locally across SD cards, USB flash drives, and mobile phone bluetoothing; *and* in the Cloud, on datasharing platforms such as Datafilehost, and using Facebook and WhatsApp messaging.

Marion Walton calls such mobile media ecologies a form of "pavement internet" to draw attention to the fact that "even as smartphone and feature phone

1. see Odom et al. (2012)
2. see Suchman (2002)

handsets increasingly emphasize cloud-based sharing of media, high prepaid data costs mean that these 'affordances' can be unaffordable" and how such 'digital materialities' shape participation (2014, 451). Through the case study of a bystander video posted on the Facebook page of the *Daily Sun*[3] showing police brutality towards the Mozambican taxi driver Mido Macia, Walton highlights how people switch from costly Cloud services to collocated sharing and messaging:

> Over a quarter of comments on the Facebook post for the original video were requests from readers who shared their cell phone numbers, and BBM[4] pins, asking other readers to send them their own copy of the clip via WhatsApp or BBM … "Those who r able 2 c de video plz watsapp it 4 m @[phone number anonymised][5]. These readers did not want to share the Macia video on Facebook or YouTube, but wanted to pass it to their own WhatsApp contacts or smuggle it via Bluetooth through the cracks of the pavement internet." –(Walton 2014, 456)

These case-studies elaborate socially rich and cost-conscious practices and lay bare digital materialities we fail to consider when adopting the seamless rhetoric of the Cloud.

7.4 REARCHITECTING

Furthermore these case study provoke us to reflect on where a re-invigorated file abstractions and grammars of actions might work beyond individual Cloud silos and within pavement internet ecologies.

For starters these case studies show how mobile phone numbers are a preferred identifier over, say, email addresses. And through the grammars of showing & collecting that we explored in Chapter 4, *My Stuff* supports collocated sharing through gestures[6] of showing and telling. While not dependent on the Cloud, such gestures of sharing effectively render the mobile datastore into a silo of its own, affording no opportunities for files to break free, for instance using Bluetooth or Messaging. Unfortunately, the Windows Phone 8.1 API does not support programmatic access to the Bluetooth Object Push Protocol, so we are unable to prototype and explore such scenarios.

But the *My Stuff* prototype is certainly guilty of functioning a Cloud silo. That is, it is not interoperable across different clients, say for instance with an Android app, and between Cloud services. This is something that can be addressed. However, allowing other clients to interact with the Cloud service using open protocols also means giving up control. To demonstrate this let us return to the gifting scenario. In the above findings we saw that its not possible to collapse receiving a gift into a single HTTP GET request. Changing the HTTP verb from GET to POST, will alleviate most of the issues we identified earlier. However, it still remains risky to remove the file after the HTTP post request to receive a gift is handled by the API server. For instance, even though the server completes the request and marks the gift exchange as complete, the data might have been corrupted in transit, especially if it is transferred through high latency links or over unreliable wireless networks. For instance, participants in Schoon's (2016) study often complained of timeouts and aborted transfers when they try to upload or download their music tracks[7]. To mitigate against this, we need to tease apart this single POST request and create two separate ones: (1) to download

3. a South African tabloid.
4. Blackberry Messaging
5. Those who are able to see the video please WhatsApp (message) it for me.
6. see Flusser (2014)
7. Personal communication (June 9th, 2015)

the contents of the file, and (2) to confirm the successful retrieval of the file to initiate deleting the copies that were created in transit.

```
// HTTP Resquest to receive a gift
GET https://mystuffservice.azure-mobile.net/gift/{giftId}/file

// Server authenticates request, finds gift record, ensures that the gift record
// has not been marked as complete, and locates the corresponding file.
// Server responds with:
//   - stream of file content -- if above criteria are fulfill
//   - error -- if above above criteria aren't fulfilled

// HTTP Resquest to acknowledge file receipt
PUT http://mystuffservice.azure-mobile.net/gift/{giftId}
Content: application/json
{
  "giftId": "a1",
  "userId": "alex",
  "senderId": "alex",
  "filename": "selfie.jpg",
  "status": "complete"
}

// Server marks gift record as complete, notifies the sender, and deletes file.
```

This separation allows the client app to check if the file download completed successfully. If it was not successful the client can attempt to redownload the file. Once the download is successful, the client then needs to notify the server to update its records. While this solution caters for failed downloads, it couples together client and server side components in a way that we can't guarantee adherence to the exchange protocol we outlined above. That is, if we were to open up the API server and make it available to other clients that we did not implement, who is to say that they would keep up with the housekeeping. For instance, they might issue the GET request to download the file, without updating the gift record that it has been completed through the above PUT request.

7.4.1 *Ensuring protocol adherence*

To ensure that other clients can interact with the API server while adhering to the exchange protocol, we can leverage cryptography, an idea that I borrow from Bitcoin (see Antonopoulos 2015). That is, when the file-to-be-gifted is first uploaded to the API server, the API server generates and uses an encryption key to encrypt the file before storing it on Azure Block Storage. Finally the API server generates an `md5sum`[8] hash of the stored file and saves it along with they encryption key. We leverage the `md5sum` hash to verify that the file has not changed while being transmitted. In effect four records are created and persisted on the server database, as illustrated in fig. 26 through a (simplified) class diagram that represents the data schema of the database.

Like before the client requests the file-to-be-gifted, expect this time the server returns the encrypted file contents:

```
// HTTP Resquest to receive a gift
GET https://mystuffservice.azure-mobile.net/ReceivedGift/{Id}/EncryptedFile

// Server authenticates the request, finds gift record, ensures that the gift record
```

8. https://en.wikipedia.org/wiki/Md5sum

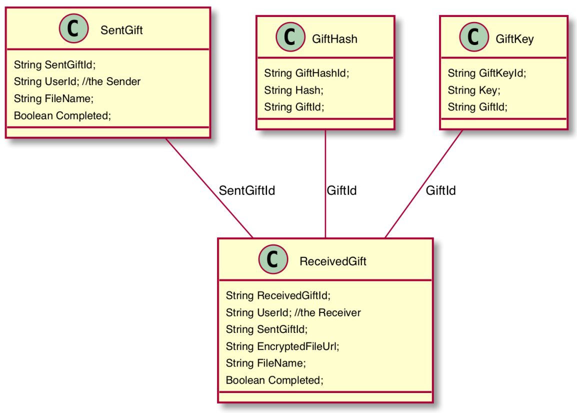


Figure 26: Database schema to support the redesigned gift exchange protocol.

```
// has not been marked as complete, and locates the corresponding encrypted file.
// Server responds with:
//   - stream of encrypted file content -- if above criteria are fulfill
//   - error -- if above above criteria aren't fulfilled
```

As the file is encrypted, the client is forced to request the key from the API server. To access the key, the API server requires the client to send the correct `md5sum` hash of the encrypted to the API endpoint:

```
// HTTP Resquest to receive encryption key
POST https://mystuffservice.azure-mobile.net/ReceivedGift/{Id}/Key
Content: application/json
{
  "receivedGiftId": "a1",
  "hash": "7abf361a90ba8b523b7b9277547da3e4"
}


// Server authenticates the request; finds gift, gift hash, and gift key records;
// checks that the client submitted hash matches the gift hash record.

// if the hashes DO NOT match the server returns an error.

// Otherwise, the server returns the encryption key, marks the gift exchange as complete,
// notifies the sender, and deletes the encrypted file.
```

Finally, the client app can decrypt the file and the server can be sure that the transfer is successful before marking it as complete and deleting file copies hosted in transit. Unlike before, these redesigned HTTP endpoints respect the semantics of common HTTP methods, outlined in tbl. 2. Especially the GET request that we outlined above, now respects the convention that a “GET method should not have the significance of taking an action other than retrieval” (RFC2616 1999), or in short, that GET methods are safe. Additionally, GET methods are idempotent, a property that refers to “(aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request” (RFC2616 1999). In our case, multiple GET requests to the above API endpoint will only ever retrieve the encrypted file contents, or return an error, and carry no additional side-effects. The POST request, according to the HTTP specification does not need to be safe nor idempotent (RFC2616 1999). The side effect of the above POST method is to initiate the state changes of the gift and encrypted gift file resources by marking the former as complete and deleting the latter.

Table 2: Summary of common HTTP methods.

Method	Description	Idempotent	Safe
GET	Reads a resource representation	Yes	Yes
POST	Creates a resource with the request payload or initiates a data-handling process with state changes.	No	No
DELETE	Deletes a resource representation	No	No
PUT	Replace current resource representation with the request payload	No	No
PATCH	Apply partial modification to the current resource representation.	No	No

The principles of these redesigned API endpoints – documented in the above, rather lengthy and dry descriptions – can also be applied to the giving exchange protocol of the previous chapter, and allow other clients, outside of our direct control, to interact and transact with the API server. Such interoperability is a strength of the HTTP protocol, but requires us to respect its semantics and conventions.

7.4.2 *Dealing with Latency & Asynchronicity*

We will spare the reader dry, low-level descriptions of the changes we implemented to deal with issues surrounding latency and asynchronicity that our relocation to Cape Town surfaced. Instead we adopt a higher level approach. The problem, expressed in a nutshell, is that asynchronous programming – or rather dealing with asynchronous sources while programming – is complex. This is an issue that we already alluded to in the previous chapter, when trying to display photo thumbnails in a simple scrollable list UI. I found it difficult to directly handle and display photo thumbnails because they needed to be requested through an asynchronous method which returns a `Future` – stand-ins for computational results that are initially unknown but may become available or return an error at some later time. At the time and with moderate success – in the sense that the app worked so long as the user didn't scroll too quickly – I was able to implement the logic to query these ansynchronous method, keep track of the results, populate the UI once the results became available, and free up resources when the thumbnails are no longer needed – when the user scrolls. Daunted by the prospect of implementing a similar complex workaround, I 'cheated' when implementing the logic to synchronize the database. That is, I blocked the UI and displayed an initialization splash screen while I waited for the asynchronous synchronization methods to complete and assumed that these methods would complete. In Cambridge, as we discussed earlier, I got away with this 'cheat' because I only needed to block the UI for an instant before the synchronization completed. The significantly higher latencies that we experience in Cape Town, however, exposed this 'cheat'. Not only did the app take longer to load, but would sometimes crash if the synchronization would timeout or be otherwise disrupted.

Upon consulting with fellow interns from the Programming languages and principles group at Microsoft Research, who I had met while in Cambridge, I learned how imperative programming paradigms treat state and events as second class citizens and that I should consider re-implementing the app within a functional-reactive paradigm. I then discovered the reactive extensions (Rx) to the .NET framework (MSDN 2015) and the ReactiveUI library[9]. The reactive extensions are a library for "composing asynchronous and event-based programs

9. https://reactiveui.net/

using observable sequences and [...] query operators" (MSDN 2015) and the Reactive UI library provides the framework for leveraging these libraries for User Interfaces. This approach to programming, however, requires a declarative style that is more adept at expressing relationships between groups of things that are changing and compose these into reactive pipelines. That is, programming becomes more about describing what you want to happen (declarative) and when it should happen (reactive) rather than how to do it (imperative). Reflecting on this issue, the influential mobile developer Jake Wharton notes that "having a single asynchronous source will ultimately break the traditional imperative style of programming we're used to. Not 'break' in the sense that it stops working, but in the sense that it pushes the complexity onto you [the programmer], and you start losing the things that imperative programming is really good for" (2016). This functional reactive approach to (UI) programming, required a substantial rewrite of the *My Stuff* prototype, but using this approach I was able to model and better respond to issues surrounding asynchronicity and latency and eliminate the crashes and long loading times experienced on higher latency networks.

As this overview illustrates, programming tools and language constructs, such as the Reactive Extensions, are only just catching up – or perhaps more accurately, gaining the attention and advocacy in mobile developer communities they deserve – to the difficulties and complexities of programming with asynchronous sources. Especially in resource constraint contexts – where such complexities are not only more likely to arise but also are more consequential, this is of paramount importance.

7.5 CONCLUSION

In this chapter we have interrogated the subtle and insidious ways in which our relocation to Cape Town, to the margins of the Internet, undermined our earlier efforts to develop tools to give people better awareness, ownership, and control over the stuff that they store and share on their mobiles and in the Cloud. In one extreme case, the combination of latency and unreliable wireless networks, effectively locked the user out from accessing the *My Stuff* datastore – the polar opposite of the sense of digital possession I had intended to convey. Even if the architecture supports and conveys strong ownership and control, this is of little help, if the User Interface does not reinforce it. After all for the PC user, the file icon on the Desktop does as much work of giving a file a sense of place as the file system does, and effectively work together to reinforce this sense of place.

Especially with the mobile we are invited to think about technology in global terms (TOYAMA 2013) and the rhetoric and abstractions of the Cloud and popular accounts of the Internet make it seem like place – where we are within the network and where our stuff is stored – no longer matters. In the development of the *My Stuff* prototype we have been leveraging the HTTP protocol, at the highest level of abstraction in the OSI stack: the application layer (SILBERSCHATZ et al. 2013, 759). And the way such abstractions work is to relieve those doing the programming from worrying about the underlying implementations details and infrastructures. After all "computing infrastructure", as Blanchette reminds us, "is precisely task with relieving users and programmers from the specifics [sic] constraints of the material resources of computation" (2011, 1042). The placeless rhetoric of the Cloud combined with such programming abstractions are deeply engrained and are, in practice, difficult to escape. Our relocation exposed these biases. And the digital materialities of South African media ecologies, call into question the power and politics of having a local database that periodically needs to be brought into sync with the master version, stored in the Cloud, as the Azure Mobile Services tutorials and reference implementations advocate. We have found inspiration in the multifaceted and cost-conscious media sharing practices in South Africa that highlight the value of metadata and the need for mobile media to travel freely across and between local mobile media materialities that foreground SD-cards, USB sticks, and Bluetooth and the Cloud. The rearchitected *My Stuff* prototype – enriched through its relocation – is an important step in that direction.

8 CONCLUSION

We began this dissertation with a deceptively simple question – *Does Windows Phone 8 support gifting?* – and have seen our attempts to answer that question transformed into a whole range of concerns: locating this research undertaking within the computing architectures of our time – the mobile and the Cloud – and setting it into correspondence with rich human communication and sharing practices as well as the human values such practices articulate. It is through gifting that we are forced to think of digital objects less as some immaterial, virtual entity, but more as something physical, something with material properties and social significance. In short, gifting not only demands, but also focuses a material perspective of information.

Through this perspective, we have studied multiple mobile platforms – Windows Phone 8, Android, and Windows Phone 8.1 – and interrogated, compared, and contrasted how we encounter objects, or rather photos more specifically, on those mobile platforms and the representation practices and digital materialities these encounters manifest. For instance, we have seen that the changing landscape of computing – one that is moving from the desktop to the mobile – is mirrored in a corresponding shift in the core abstraction of how data is represented in computing architecture and funnelled through interfaces: from a `File` to a `DataStream`, the latter of which does not support gifting. And thus, we found our initial question transformed to now consider the *interplay* between the mobile datastore and the mobile sharing interface as a problematic site with opportunities for redesign.

These technical investigations, however, took us further and further away from the human communication and sharing practices we endeavoured to support. Recent scholarship on digital materialities of information, advocates for "investigations that are *simultaneously* technical, social and cultural, and that seek to find, with the particular configurations of code and digital objects, manifestations of and provocations for the cultural setting in which they are developed and deployed" (Dourish 2017, 57, added emphasis). While this research undertaking in large part affirms this perspective, I worry about the temporal association of the word I emphasised: simultaneously. The research of this dissertation on a whole is, rather, *concomitantly* technical, social, and cultural. The research I have documented in this dissertation, show that a temporal simultaneity of the technical, social, and cultural is difficult to achieve in practice. I tend to think of myself more as a snorkeler exploring a reef, taking a deep breath as I dive down into the technical realities of the mobile and explore this material, studying its fluxes and flows as well as the ways in which it is pliable or recalcitrant. But with such technical investigations that involve programming, I am, as Phil Agre points out, placing myself "imaginatively inside the system" (1995, 73). And so, like the snorkeler, it is important for us to come up for air and to sustain our concomitant approach by placing technical concerns into *correspondence* with social and cultural practices.

And this is precisely what we did, by sensitising ourselves to issues surrounding identity performance, context, storytelling, and communication, but also to look at our unique and diverse, but also resource-constrained South African context. We identified co-located interactions as a salient design space that we explored through a of technology probe to further interrogate the mobile sharing interface. These investigations lead us to develop an early datastore prototype, a co-present photo gallery, tailored to how people present their stuff (and themselves in the process) to those that surround them. Just like with the previous gifting fiction and technology probe, the prototype attempted to work around the incongruence with the specific ways in which information is passed between datastore and sharing interface. Just as these deeper concerns were developing, Microsoft announced a major architectural change to the Windows Phone platform that re-introduced the file as a major component of the Windows Phone 8.1 platform, datastore, and sharing interface.

We studied those architectural changes and contextualised these in the history of the file, which showed a strange juxtaposition, that despite their central role in system design, files seem outdated in our increasingly networked and mobile lives. Looking at how Windows Phone 8.1 attempts to re-surface files through the *Files* manager were, however, muddled. Closer investigations revealed that the extended grammar of action that *Files*, in contrast to PC file managers, now support create proliferations of copies that designers of earlier systems sought to avoid (see Smith et al. 1982). We leveraged these newly introduced APIs, and created a prototype datastore, *My Stuff*, to tidy up the muddle and further our investigation.

We took this investigation to the Human Experience and Design research group at Microsoft Research in Cambridge, UK where we integrated our material perspective with the nascent field of research – pioneered in Cambridge – that positions the stuff that people store and share on their personal devices and in the Cloud as *digital possessions*, which orients us to the social and moral dimensions of that stuff. Leveraging *My Stuff*, we prototyped extended file abstractions to incorporate metadata surrounding social exchanges to better support notions of digital possession especially when these possession move through the Cloud. A topic that we explored, by studying contemporary cloud architectures and developing a Cloud service to support new grammars of action – to give a copy & to gift – which we implemented on the *My Stuff* prototype.

Finally, we relocated our research back to Cape Town only to discover that the *My Stuff* prototype and Cloud service did not work in the same way it did in Cambridge. This lead us to interrogate – contrary to the rhetoric of the Cloud – how physical location affects our use of the Cloud. We uncovered how good infrastructure can mask problems with computer architecture, and documented how we re-architected the *My Stuff* datastore and Cloud service to better cope with the material realities of marginal internet connectivity. This rearchitected and Cloud service finds its inspiration in local mobile media materialities and sharing practices that articulate a need for mobile media to travel freely between local media ecologies and the Cloud.

8.1 SUMMARY OF CONTRIBUTIONS

Throughout this dissertation we have argued for a material perspective of information and leveraged this perspective to study our contemporary computing landscape. We demonstrated through a portfolio of prototypes a progressive variation on theme: giving people better awareness and control over the stuff that matters to them. This portfolio foregrounds the need for more human-centred computer architectures. Applying a material perspective to think of the stuff we store and share on our mobile phones as a form of digital possession, as we have demonstrated throughout this thesis, is a powerful generative metaphor.

Consequently the research presented in this dissertation contributes to and advances the state of the art in three emerging research areas: the *materialities of information*, *digital possessions*, and *research through design*.

We contribute to wider research agendas surrounding the *materialities of information* by extending its empirical to mobile architectures and scrutinising at a foundational level the specific technological arrangements and interfaces that constitute these architectures and the social consequences of these arrangements. We have documented these, with what can at times seem like excruciating detail, however:

> As Blaauw & Brooks remark in their monumental study of computer architecture, "when reading the professional paper describing the architecture of a new machine, it is often difficult to discern the real design dilemmas, compromises, and struggles behind the smooth, after-the-fact description" (1997, 7). Yet these, dilemmas, the compromises, these struggles will increasingly matter, as the software infrastructure comes to mediate a breathtaking proportion of social relations. –(Blanchette 2011, 1056)

We make similar contributions to wider research agendas into *digital possessions* by again extending the empirical focus to the mobile. We show how contemporary mobile architectures do not support the age-old practice of gifting nor notions of possession. We show how reinvigorated file abstractions and more social grammars of action – prototyped on the *My Stuff* app and Cloud service – can support possession through better awareness and control. Relocating this research between Cambridge and Cape Town show how appropriate architectures are needed to cope with infrastructural realities along the edges of the Internet to effectively transact and interact with Cloud services, and ultimately relate to one another through its mediating influence. Foregrounding these realities – rather than obscuring and abstracting away from them – are crucial and highlight potentials for how redesigned file abstractions can work across mobiles and the Cloud.

Finally, we advance the state of the art of the *Research through Design* agenda, by responding to Zimmerman et al.'s call to action to document the *whole* Research through Design process, showing "how theories from other disciplines were integrated" and beginning with the crucial first step: *problem framing* (Zimmerman et al. 2010, 316). By interrelating the discussions, theories, and commentators we engage throughout this process and showing how these inform the design and develop of prototypes we provide further evidence of the central role artefacts play for Research through Design methodologies. Galey & Ruecker, for instance, show how prototypes on their own reify arguments, embody theory, and articulate and contribute to knowledge (2010). This is an idea that Bardzell et al. find both interesting and provocative, but wonder "what that would look like" in practice (2015). This dissertation fills in that gap.

8.2 LIMITATIONS

Following Galey & Ruecker (2010), the prototypes we designed and developed as part of this research make and embody important conceptual contributions that are reinforced and annotated (see Löwgren 2013) through detailed descriptions of the theories and insights from diverse academic disciplines. But we have not investigated the use of the *My Stuff* prototype through longitudinal trials. Ideally such a study would see users use their own devices, already containing their own data, to uncover the ways in which they appropriate the prototype. In Cape Town, such a study would have been impossible to conduct since the Windows Phone 8.1 platform has poor adoption. For studies that give users devices with the prototype installed, it would be difficult to isolate novelty effects of using a new and unfamiliar device. While we might lament that the choice of device is unfortunate, it is important to remember that the unfamiliarity and novelty of the Windows Phone platform helped me better understand the digital materialities of mobile architectures in the first place, by virtue of being able to compare it to the way things are on Android.

8.3 FUTURE WORK

This dissertation uncovers several opportunities for future work. Most notably to port the *My Stuff* prototype to the Android platform and investigate if the Android Storage Framework would support the development of redesigned file abstractions. Given the high prevalence of Android devices, this would also make it easier to design a longitudinal study where people can use their own devices and would ultimately shed light on how users encounter and appropriate design ideas and their manifestations. Additionally, it would be interesting to see if such a prototype could be extended to support the mobile media sharing practices of the participants of Schoon's (2016) study, which we discussed in the previous chapter, and that involve Bluetooth as well as Cloud services.

We remain intrigued by the insights, dilemmas, and opportunities our relocations afforded us: once when we traveled from Cape Town to Cambridge, reversing, and redirecting the pervasive and asymmetric information and innovation flows and back to Cape Town again along those flows (see Suchman 2002, 139). Actively seeking such relocations could form the basis of a design practice that is *itinerative* rather than *iterative*.

"The Cloud", as Hu reminds us, "produces users rather than publics, and therefore individual rather than collective action (2015, 147). Especially in our South African context, commentators have noted that the 'networked individualism' (Walton et al. 2012) and individualist logics (Bidwell 2014) embedded in design can relegate communication genres and forms of sociality that emphasise collectivity to the margins. An important area of future work is research into mobile and cloud architectures to support collectives and commons: stuff that is collectively owned.

The *My Stuff* prototype could further be leveraged to rethink the grammar of *delete*. Our redesigned file abstractions already account for and represent where the copies of our files are residing. Deleting a file would therefore require coordination of diverse and distributed datastores along with thoughtful interaction design.

There is much work to be done, and this – like the work presented in this dissertation – will be a substantial technical undertaking. But it is equally important to remember the broader context in which this work is situated. Richard Harper says it with so much compassion and eloquence that I give him the last word:

> when designers approach the "problem space" of the Cloud, for example, they have to consider the ontological "sense" that their designs provide. Do users come to trust in their relations to their digital "stuff"? Does that trust resonate with how those people get on with their lives, lives that are suffused with digital materials but that are, also, essentially about material properties that stand as part and parcel of their human endeavors? These are anthropological concerns, as much as they are technological and design questions.
> —(Harper 2014, 12).

BIBLIOGRAPHY

Agre, P. 1995. "Conceptions of the User in Computer Systems Design." In *The Social and Interactional Dimensions of Human–Computer Interfaces,* edited by Thomas, P., 67–106. Cambirdge, UK: Cambridge University Press. (Cited on pages 28, 93).

Álvaro. 2016. "Get Real Path from URI, Android KitKat New Storage Access Framework - Stack Overflow." Accessed March 15. https://stackoverflow.com/questions/20067508/get-real-path-from-uri-android-kitkat-new-storage-access-framework?lq=1. (Cited on pages 16, 23).

Amano, K., N. Goda, S. Nishida, Y. Ejima, T. Takeda, & Y. Ohtani. 2006. "Estimation of the Timing of Human Visual Perception from Magnetoencephalography." *Journal of Neuroscience* 26 (15): 3981–3991. doi: 10.1523/JNEUROSCI.4343-05.2006. pmid: 16611814. (Cited on page 83).

Antonopoulos, A. M. 2015. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies.* 1. ed. OCLC: 876351095. Beijing: O'Reilly. (Cited on page 88).

Ballendat, T., N. Marquardt, & S. Greenberg. 2010. "Proxemic Interaction: Designing for a Proximity and Orientation-Aware Environment." In *ACM International Conference on Interactive Tabletops and Surfaces: ITS '10,* 121–130. New York, NY, USA: ACM. doi: 10.1145/1936652.1936676. (Cited on page 37).

Banks, R. 2014. "Trust in Design." In *Trust, Computing, and Society,* edited by Harper, R. Cambirdge, UK: Cambridge University Press. (Cited on pages 61, 63).

Barad, K. 2003. "Posthumanist Performativity: Toward an Understanding of How Matter Comes to Matter." *Signs: Journal of Women in Culture and Society* 28 (3). (Cited on page 2).

Bardzell, J., & S. Bardzell. 2015. "Humanistic HCI." *Synthesis Lectures on Human-Centered Informatics* 8 (4): 1–185. doi: 10.2200/S00664ED1V01Y201508HCI031. (Cited on pages 3, 5).

Bardzell, J., S. Bardzell, & L. K. Hansen. 2015. "Immodest Proposals: Research Through Design and Knowledge." In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems: CHI '15,* 2093–2102. New York, NY, USA: ACM. doi: 10.1145/2702123.2702400. (Cited on pages 3, 4, 28, 80, 95).

Baumer, E. P. S., J. Burrell, M. G. Ames, J. R. Brubaker, & P. Dourish. 2015. "On the Importance and Implications of Studying Technology Non-Use." *interactions* 22 (2): 52–56. doi: 10.1145/2723667. (Cited on page 82).

Bell, G., M. Blythe, & P. Sengers. 2005. "Making by Making Strange: Defamiliarization and the Design of Domestic Technologies." *ACM Trans. Comput.-Hum. Interact.* 12 (2): 149–173. doi: 10.1145/1067860.1067862. (Cited on page 10).

Bidwell, N. J. 2014. "Moving the Centre to Design Social Media in Rural Africa." *AI & SOCIETY* (September 19): 1–27. doi: 10.1007/s00146-014-0564-5. (Cited on page 96).

Bidwell, N. J., T. Reitmaier, & K. Jampo. 2014. “Orality, Gender and Social Audio in Rural Africa.” In *Proceedings of the 11th International Conference on the Design of Cooperative Systems: COOP’14,* edited by Rossitto, C., Ciolfi, L., Martin, D., & Conein, B., 225–241. Springer. doi: 10.1007/978-3-319-06498-7_14. (Cited on page 10).

Bidwell, N. J., S. Robinson, E. Vartiainen, M. Jones, M. J. Siya, T. Reitmaier, G. Marsden, & M. Lalmas. 2014. “Designing Social Media for Community Information Sharing in Rural South Africa.” In *Proceedings of the Southern African Institute for Computer Scientists and Information Technologists Annual Conference: SAICSIT ’14,* 104–114. New York, NY, USA: ACM. doi: 10.1145/2664591.2664615. (Cited on page 10).

Blaauw, G. A., & F. P. Brooks Jr. 1997. *Computer Architecture: Concepts and Evolution.* 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on page 95).

Blanchette, J.-F. 2011. “A Material History of Bits.” *Journal of the American Society for Information Science and Technology* 62 (6): 1042–1057. doi: 10.1002/asi.21542. (Cited on pages 22, 91, 95).

Bluetooth Users Against Bush. 2004. Accessed June 13, 2016. http://www.bluetoothusersagainstbush.com/. (Cited on page 42).

boyd, d. 2008. “Why Youth (Heart) Social Network Sites: The Role of Networked Publics in Teenage Social Life.” In *Youth, Identity, and Digital Media,* edited by Buckingham, D. Cambirdge, MA: MIT Press. http://papers.ssrn.com/abstract=1345415. (Cited on page 37).

Bright, P. 2016. “OneCore to Rule Them All: How Windows Everywhere Finally Happened.” May 20. Accessed May 23, 2016. http://arstechnica.com/information-technology/2016/05/onecore-to-rule-them-all-how-windows-everywhere-finally-happened/. (Cited on page 51).

Buckingham, D. 2008. “Introducing Identity.” In *Youth, Identity, and Digital Media,* edited by Buckingham, D., 1–20. Cambirdge, MA: MIT Press. http://papers.ssrn.com/abstract=1345415. (Cited on pages 33–35).

Burke, P. 2013. “iPaulPro/aFileChooser.” Accessed April 2, 2016. https://github.com/iPaulPro/aFileChooser. (Cited on page 16).

Chalfen, R. 1987. *Snapshot Versions of Life.* Madison, WI: University of Wisconsin Press. (Cited on pages 34–36).

Chimero, F. 2013. “What Screens Want.” http://www.frankchimero.com/writing/what-screens-want/. (Cited on page 10).

Clawson, J., A. Voida, N. Patel, & K. Lyons. 2008. “Mobiphos: A Collocated-Synchronous Mobile Photo Sharing Application.” In *Proceedings of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services: MobileHCI ’08,* 187–195. New York, NY, USA: ACM. doi: 10.1145/1409240.1409261. (Cited on page 36).

Dalsgaard, P. 2016. “Experimental Systems in Research Through Design.” In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems: CHI ’16,* 4991–4996. New York, NY, USA: ACM. doi: 10.1145/2858036.2858310. (Cited on page 28).

Developers, A. 2016. “Storage Access Framework.” Accessed April 2. https://developer.android.com/guide/topics/providers/document-provider.html. (Cited on page 22).

Developers, G. 2015. “Google Cloud Messaging: Overview.” July 24. Accessed September 8, 2015. https://developers.google.com/cloud-messaging/gcm. (Cited on page 73).

Dijck, J. van. 2008. “Digital Photography: Communication, Identity, Memory.” *Visual Communication* 7 (1): 57–76. doi: 10.1177/1470357207084865. (Cited on pages 10, 27, 77).

Donner, J. 2015. *After Access: Inclusion, Development, and a More Mobile Internet.* The information society series. Cambridge, Massachusetts: The MIT Press. (Cited on page 82).

Dourish, P. 2004. “What We Talk About When We Talk About Context.” *Personal and Ubiquitous Computing* 8 (1): 19–30. doi: 10.1007/s00779-003-0253-8. (Cited on pages 34, 37, 38, 49).

———. 2014. “NoSQL: The Shifting Materialities of Database Technology.” *Computational Culture,* no. 4. http://computationalculture.net/article/no-sql-the-shifting-materialities-of-database-technology. (Cited on pages 2, 3, 10, 21, 51).

———. 2015a. “Not The Internet, but This Internet: How Othernets Illuminate Our Feudal Internet.” In *5th Decennial Aarhus Conference on Critical Alternatives: AARHUS'15,* 12. doi: 10.7146/aahcc.v1i1.21200. (Cited on pages 10, 82).

———. 2015b. “Packets, Protocols, and Proximity: The Materiality of Internet Routing.” In *Signal Traffic: Media Infrastructure and Globalization,* edited by Parks, L. & Starosielski, N. Urbana, IL: University of Illinois Press. (Cited on page 2).

———. 2017. *The Stuff of Bits: An Essay on the Materialities of Information.* Cambridge, Massachusetts: The MIT Press. (Cited on page 93).

Dourish, P., & G. Bell. 2011. *Divining a Digital Future: Mess and Mythology in Ubibquitous Computing.* Cambirdge, MA: MIT Press. (Cited on pages 28, 59).

Dourish, P., & M. Mazmanian. 2013. “Media as Material: Information Representations as Material Foundations for Organizational Practice.” In *How Matter Matters,* edited by Carlile, P. R., Nicolini, D., Langley, A., & Tsoukas, H., 92–118. Oxford, UK: Oxford University Press. doi: 10.1093/acprof:oso/9780199671533.003.0005. (Cited on pages 1–3, 21).

Downey, G. L. 1998. *The Machine in Me: An Anthropologist Sits Among Computer Engineers.* New York: Routledge. (Cited on pages 27, 28).

Evert. 2009. “HTTP GET with Request Body.” Accessed May 26, 2015. https://stackoverflow.com/questions/978061/http-get-with-request-body. (Cited on page 85).

Farman, J. 2012. *Mobile Interface Theory: Embodied Space and Locative Media.* Oxon, UK: Routledge. (Cited on page 3).

Fiveash, K. 2015. “AWS Outage Knocks Amazon, Netflix, Tinder and IMDb in MEGA Data Collapse.” September 20. Accessed December 2, 2015. http://www.theregister.co.uk/2015/09/20/aws_database_outage/. (Cited on page 1).

Flusser, V. 1999. *The Shape of Things: A Philosophy of Design.* Edited by Mathews, A. London: Reaktion Books. (Cited on pages 29, 63).

———. 2014. *Gestures.* Minneapolis: University of Minnesota Press. (Cited on page 87).

Frayling, C. 1993. "Research in Art and Design." *Royal College of Art Research Papers* 1 (1): 1–5. (Cited on page 5).

Galey, A., & S. Ruecker. 2010. "How a Prototype Argues." *Literary and Linguistic Computing* 25 (4): 405–424. doi: 10.1093/llc/fqq021. (Cited on page 95).

Gallardo, J., & A. Singh. 2014. "Contracts and Pickers: Building Apps That Work Together on Windows – BUILD2014." BUILD2014. https://channel9.msdn.com/Events/Build/2014/2-520. (Cited on pages 56, 57).

Gaver, W. 2012. "What Should We Expect from Research Through Design?" In *Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems: CHI '12,* 937–946. New York, NY, USA: ACM. doi: 10.1145/2208516.2208538. (Cited on pages 39, 49).

Goffman, E. 1959. *The Presentation of Self in Everyday Life.* New York, NY: Anchor Books. (Cited on pages 33, 34, 36, 37, 49).

———. 1961. *Asylums.* New York, NY: Anchor Books. (Cited on page 37).

Google Cloud Platform Live: Keynote from Urs Hölzle. 2014. March 25. https://www.youtube.com/watch?v=qokEYBNWA_0. (Cited on pages 65, 66).

Greenberg, S., N. Marquardt, T. Ballendat, R. Diaz-Marino, & M. Wang. 2011. "Proxemic Interactions: The New Ubicomp?" *interactions* 18 (1): 42–50. doi: 10.1145/1897239.1897250. (Cited on pages 32, 36, 37).

Hall, E. T. 1966. *The Hidden Dimension.* New York: Doubleday. (Cited on pages 32, 37, 49).

Hansen, L. K. 2014. "What's in a Word?" *interactions* 21 (1): 22–23. doi: 10.1145/2541248. (Cited on page 25).

Harper, R. 2010. *Texture: Human Expression in the Age of Communications Overload.* Cambirdge, MA: MIT Press. (Cited on pages 31, 33, 34).

———. 2014. "Introduction and Overview." In *Trust, Computing, and Society,* edited by Harper, R. Cambirdge, UK: Cambridge University Press. (Cited on pages 76, 96).

Harper, R., & W. Odom. 2014. "Trusting Oneself: An Anthropology of Digital Things and Personal Competence." In *Trust, Computing, and Society,* edited by Harper, R. Cambirdge, UK: Cambridge University Press. (Cited on pages 63, 77).

Harper, R., T. Regan, S. Izadi, K. A. Mosawi, M. Rouncefield, & S. Rubens. 2007. "Trafficking: Design for the Viral Exchange of TV Content on Mobile Phones." In *Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services: MobileHCI '07,* 249–256. New York, NY, USA: ACM. doi: 10.1145/1377999.1378015. (Cited on pages 41, 42).

Harper, R., E. Thereska, S. E. Lindley, R. Banks, P. Gosset, W. Odom, G. Smyth, & E. Whitworth. 2013. "What Is a File?" In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work: CSCW '13,* 1125–1136. New York, NY, USA: ACM. doi: 10.1145/2441776.2441903. (Cited on pages 29, 30, 60, 62, 63, 76).

Hollan, J., & S. Stornetta. 1992. "Beyond Being There." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '92,* 119–125. New York, NY, USA: ACM. doi: 10.1145/142750.142769. (Cited on page 38).

Hu, T.-H. 2015. *A Prehistory of the Cloud.* Cambridge, Massachusetts: The MIT Press. (Cited on pages 82, 96).

Hutchins, E. 1995. *Cognition in the Wild.* MIT Press. (Cited on pages 33, 35).

———. 2010. "Enaction, Imagination, and Insight." In *Enaction: Towards a New Paradigm for Cognitive Science,* edited by Stewart, J., Gapenne, O., & Di Paolo, E. A., 425–450. Cambirdge, MA: The MIT Press. http://mitpress.universitypressscholarship.com/view/10.7551/mitpress/9780262014601.001.0001/upso-9780262014601-chapter-16. (Cited on page 33).

Hutchinson, H., W. Mackay, B. Westerlund, B. B. Bederson, A. Druin, C. Plaisant, M. Beaudouin-Lafon, et al. 2003. "Technology Probes: Inspiring Design for and with Families." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '03,* 17–24. New York, NY, USA: ACM. doi: 10.1145/642611.642616. (Cited on page 39).

Ingold, T. 2000. *The Perception of the Environment: Essays on Livelihood, Dwelling and Skill.* Oxon, UK: Routledge. (Cited on pages 9, 41).

———. 2007. *Lines: A Brief History.* Oxon, UK: Routledge. (Cited on page 49).

———. 2011. *Being Alive: Essays on Movement, Knowledge and Description.* Oxon, UK: Routledge. (Cited on pages 29, 76).

———. 2013. *Making: Anthropology, Archaeology, Art and Architecture.* Oxon, UK: Routledge. (Cited on pages 5, 28, 48).

———. 2014. "Resonators Uncased: Mundane Objects or Bundles of Affect?" *HAU: Journal of Ethnographic Theory* 4 (1): 517–521. doi: 10.14318/hau4.1.035. (Cited on page 77).

———. 2015. *The Life of Lines.* Oxon, UK: Routledge. (Cited on page 42).

Kindberg, T., M. Spasojevic, R. Fleck, & A. Sellen. 2005a. "The Ubiquitous Camera: An in-Depth Study of Camera Phone Use." *IEEE Pervasive Computing* 4 (2): 42–50. doi: 10.1109/MPRV.2005.42. (Cited on pages 10, 34, 35, 38, 39).

Kindberg, T., M. Spasojevic, R. Fleck, & A. Sellen. 2005b. "I Saw This and Thought of You: Some Social Uses of Camera Phones." In *CHI '05 Extended Abstracts on Human Factors in Computing Systems: CHI EA '05,* 1545–1548. New York, NY, USA: ACM. doi: 10.1145/1056808.1056962. (Cited on pages 9, 39).

Koskinen, I., J. Zimmerman, T. Binder, J. Redstrom, & S. Wensveen. 2011. *Design Research Through Practice: From the Lab, Field, and Showroom.* Waltham, MA: Morgan Kaufmann. (Cited on page 28).

Lalonde, L., & D. R. Totzke. 2013. *Windows Phone 8 Recipes: A Problem-Solution Approach.* OCLC: ocn839661114. Berkeley, CA: Apress. (Cited on pages 44–46).

Law, J. 2004. *After Method: Mess in Social Science Research.* International Library of Sociology. Oxon, UK: Routledge. (Cited on page 43).

Lefebvre, H. 2004. *Rhythmanalysis: Space, Time and Everyday Life.* Translated by Elden, S. & Moore, G. Bloomsbury. (Cited on page 38).

Lindley, S. E., C. C. Marshall, R. Banks, A. Sellen, & T. Regan. 2013. "Rethinking the Web As a Personal Archive." In *Proceedings of the 22nd International Conference on World Wide Web: WWW '13,* 749–760. International World Wide Web Conferences Steering Committee. http://dl.acm.org/citation.cfm?id=2488388.2488454. (Cited on pages 60, 63).

Ling, R. S. 2008. *New Tech, New Ties: How Mobile Communication Is Reshaping Social Cohesion.* Cambirdge, MA: MIT Press. (Cited on pages 3, 37, 38, 42).

Löwgren, J. 2013. "Annotated Portfolios and Other Forms of Intermediate-Level Knowledge." *interactions* 20 (1): 30–34. doi: 10.1145/2405716.2405725. (Cited on page 95).

Löwgren, J., & E. Stolterman. 2004. *Thoughtful Interaction Design: A Design Perspective on Information Technology.* Cambirdge, MA: The MIT Press. (Cited on page 39).

Lucero, A., J. Holopainen, & T. Jokela. 2011. "Pass-Them-around: Collaborative Use of Mobile Phones for Photo Sharing." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '11,* 1787–1796. New York, NY, USA: ACM. doi: 10.1145/1978942.1979201. (Cited on pages 36, 47).

Malayeri, D. 2015. "Using Offline Data Sync in Azure Mobile Apps." September 10. Accessed September 10, 2015. https://github.com/Azure/azure-content/blob/b9ccba8528/articles/app-service-mobile/app-service-mobile-windows-store-dotnet-get-started-offline-data-preview.md. (Cited on page 70).

Manovich, L. 2001. *The Language of New Media.* Leonardo. Cambridge, MA: MIT Press. (Cited on page 8).

Marsden, G., A. Maunder, & M. Parker. 2008. "People Are People, but Technology Is Not Technology." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366 (1881): 3795–3804. doi: 10.1098/rsta.2008.0119. (Cited on page 39).

Mauss, M. (1950) 2000. *The Gift: The Form and Reason for Exchange in Archaic Societies.* Translated by Halls, W. New York, NY: W.W. Norton. (Cited on pages 8, 9, 34, 42).

Mawson, N. 2013. "Seacom Suffers Cable Break." March 25. Accessed December 2, 2015. http://www.itweb.co.za/index.php?option=com_content&view=article&id=62711. (Cited on page 1).

McLeod, J. M., & S. H. Chaffee. 1973. "Interpersonal Approaches to Communication Research." *American Behavioral Scientist* 16 (4): 469–499. doi: 10.1177/000276427301600402. (Cited on page 38).

Microsoft. 2014. "Files – Windows Apps on Microsoft Store." Accessed May 14, 2016. https://www.microsoft.com/en-us/store/apps/files/9wzdncrfj3pl. (Cited on page 55).

mikegr. 2013. "Android Gallery on KitKat Returns Different Uri for Intent.ACTION_GET_CONTENT." November 19. Accessed March 15, 2016. http://stackoverflow.com/questions/19834842/android-gallery-on-kitkat-returns-different-uri-for-intent-action-get-content. (Cited on pages 16, 21, 23).

Miller, D., & D. Slater. 2001. *The Internet: An Ethnographic Approach.* Oxford ; New York: Berg. (Cited on page 82).

Moore, G. 1965. "Cramming More Components onto Integrated Circuits." *Electronics* (April 19): 114–117. (Cited on page 65).

MSDN. 2014. "The MVVM Pattern." Accessed April 8. https://msdn.microsoft.com/en-gb/library/hh848246.aspx. (Cited on pages 44, 58).

———. 2015. "Reactive Extensions." Accessed August 10. https://msdn.microsoft.com/en-us/library/hh242985%28v=vs.103%29.aspx?f=255&MSPPError=-2147217396. (Cited on pages 90, 91).

MSDN. 2016a. "PhotoResult Class." Accessed March 8. https://msdn.microsoft.com/en-us/library/microsoft.phone.tasks.photoresult(v=vs.105).aspx. (Cited on page 14).

———. 2016b. "Picture Class Members." Accessed March 4. https://msdn.microsoft.com/en-us/library/microsoft.xna.framework.media.picture_members.aspx. (Cited on page 14).

———. 2016c. "PictureAlbum Class." Accessed March 4. https://msdn.microsoft.com/en-us/library/microsoft.xna.framework.media.picturealbum_members.aspx. (Cited on page 15).

Myers, T. 2015. "Introduction to Microsoft Azure Storage." August 3. Accessed September 1, 2015. https://github.com/Azure/azure-content/blob/611f9feda9/articles/storage/storage-introduction.md. (Cited on page 71).

Naaman, M., R. Nair, & V. Kaplun. 2008. "Photos on the Go: A Mobile Application Case Study." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '08,* 1739–1748. New York, NY, USA: ACM. doi: 10.1145/1357054.1357326. (Cited on page 35).

Negroponte, N. 1993–1998. "WIRED Columns 1993-1998." Accessed December 1, 2015. http://web.media.mit.edu/~nicholas/Wired/. (Cited on page 1).

———. 1995. "Being Digital - A Book (p)Review." *Wired,* no. 2. http://web.media.mit.edu/~nicholas/Wired/WIRED3-02.html. (Cited on page 1).

———. 1996. *Being Digital.* New York, NY: Vintage Books. (Cited on page 1).

O'Hara, K. P., M. Massimi, R. Harper, S. Rubens, & J. Morris. 2014. "Everyday Dwelling with WhatsApp." In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing: CSCW '14,* 1131–1143. New York, NY, USA: ACM. doi: 10.1145/2531602.2531679. (Cited on pages 3, 41).

Odom, W., A. Sellen, R. Harper, & E. Thereska. 2012. "Lost in Translation: Understanding the Possession of Digital Things in the Cloud." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '12,* 781–790. New York, NY, USA: ACM. doi: 10.1145/2207676.2207789. (Cited on pages 60–63, 86).

Odom, W., J. Zimmerman, & J. Forlizzi. 2011. "Teenagers and Their Virtual Possessions: Design Opportunities and Issues." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '11,* 1491–1500. New York, NY, USA: ACM. doi: 10.1145/1978942.1979161. (Cited on page 60).

———. 2014. "Placelessness, Spacelessness, and Formlessness: Experiential Qualities of Virtual Possessions." In *Proceedings of the 2014 Conference on Designing Interactive Systems: DIS '14,* 985–994. New York, NY, USA: ACM. doi: 10.1145/2598510.2598577. (Cited on pages 60, 61, 76).

Oram, A., & G. Wilson, eds. 2007. *Beautiful Code.* 1st. ed. Theory in practice series. OCLC: ocn163289538. Beijing ; Sebastapol, Calif: O'Reilly. (Cited on page 46).

Rainie, L., & B. Wellman. 2012. *Networked: The New Social Operating System.* Cambirdge, MA: MIT Press. (Cited on page 42).

Reese, G. 2009. *Cloud Application Architectures: Building Applications and Infrastructure in the Cloud.* Sebastopol, CA: O'Reilly. (Cited on pages 63–65).

Reitmaier, T. 2011. "'She Looked Deep into Our Eyes:' Reflections on Cross-Cultural Practice." In *Proceedings of the Indigenous Knowledge Technology Conference: Embracing Indigenous Knowledge Systems in a New Technology Design Paradigm: IKTC '11,* edited by Bidwell, N. J. & Winschiers-Theophilus, H., 100–107. Windhoek, Namibia. (Cited on pages 3, 36).

Reitmaier, T., P. Benz, & G. Marsden. 2013. "Designing and Theorizing Co-Located Interactions." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '13,* 381–390. New York, NY, USA: ACM. doi: 10.1145/2470654.2470709. (Cited on page III).

Reitmaier, T., N. J. Bidwell, & G. Marsden. 2011. "Situating Digital Storytelling Within African Communities." *International Journal of Human-Computer Studies* 69 (10): 658–668. doi: 10.1016/j.ijhcs.2010.12.008. (Cited on pages 10, 35).

Reitmaier, T., N. J. Bidwell, M. Siya, G. Marsden, & W. D. Tucker. 2012. "Communicating in Designing an Oral Repository for Rural African Villages." In *Proceedings of IST-Africa: Regional Impact of Information Society Technologies in Africa: IST-Africa '12.* Dar es Salem, Tanzania. (Cited on pages 10, 25).

RFC2616. 1999. "HTTP/1.1: Method Definitions." Accessed April 2, 2015. https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. (Cited on page 89).

Robinson, S., J. Pearson, T. Reitmaier, S. Ahire, & M. Jones. 2018. "Make Yourself at Phone: Reimagining Mobile Interaction Architectures With Emergent Users." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '18.* New York, NY: ACM. doi: 10.1145/3173574.3173981. (Cited on page III).

Rogers, Y. 2012. "HCI Theory: Classical, Modern, and Contemporary." *Synthesis Lectures on Human-Centered Informatics* 5 (2): 1–129. doi: 10.2200/S00418ED1V01Y201205HCI014. (Cited on pages 39, 49).

Schön, D. A. 1983. *The Reflective Practitioner: How Professionals Think in Action.* New York: Basic Books. (Cited on page 3).

Schoon, A. 2016. "Distributing Hip-Hop in a South African Town: From the Digital Backyard Studio to the Translocal Ghetto Internet." In *Proceedings of the First African Conference on Human Computer Interaction: AfriCHI'16,* 104–113. New York, NY, USA: ACM. doi: 10.1145/2998581.2998592. (Cited on pages 86, 87, 96).

Sengers, P., K. Boehner, S. David, & J. '. Kaye. 2005. "Reflective Design." In *Proceedings of the 4th Decennial Conference on Critical Computing: Between Sense and Sensibility: CC '05,* 49–58. New York, NY, USA: ACM. doi: 10.1145/1094562.1094569. (Cited on page 39).

Sengers, P., & B. Gaver. 2006. "Staying Open to Interpretation: Engaging Multiple Meanings in Design and Evaluation." In *Proceedings of the 6th Conference on Designing Interactive Systems: DIS '06,* 99–108. New York, NY, USA: ACM. doi: 10.1145/1142405.1142422. (Cited on page 39).

Shove, E., ed. 2007. *The Design of Everyday Life.* Cultures of consumption series. New York, NY: Berg. (Cited on page 34).

Silberschatz, A., P. B. Galvin, & G. Gagne. 2013. *Operating System Concepts.* 9th. New York, NY: Wiley. (Cited on pages 23, 29, 60, 61, 63, 64, 91).

Silberschatz, A., H. F. Korth, & S. Sudarshan. 2011. *Database System Concepts.* McGraw-Hill. (Cited on pages 29, 69).

Simmons, B. 2015. "Learn How Vesper Built Offline Sync Using Azure Mobile Services." Accessed August 11. `https://channel9.msdn.com/Blogs/Windows-Azure/Learn-how-Vesper-built-offline-sync-using-Azure-Mobile-Services-`. (Cited on page 66).

Smith, D. C., C. Irby, R. Kimball, & E. Harslem. 1982. "The Star User Interface: An Overview." In *Proceedings of the June 7-10, 1982, National Computer Conference: AFIPS '82,* 515–528. New York, NY, USA: ACM. doi: `10.1145/1500774.1500840`. (Cited on pages 30, 58, 61, 94).

"Snapchat." 2014. Accessed March 12, 2014. `https://play.google.com/store/apps/details?id=com.snapchat.android`. (Cited on pages 25, 26).

Star, S. L. 2010. "This Is Not a Boundary Object: Reflections on the Origin of a Concept." *Science, Technology & Human Values* 35 (5): 601–617. doi: `10.1177/0162243910377624`. (Cited on page 29).

Stolterman, E., & M. Wiberg. 2010. "Concept-Driven Interaction Design Research." *Human–Computer Interaction* 25 (2): 95–118. doi: `10.1080/07370020903586696`. (Cited on page 28).

Suchman, L. 2002. "Practice-Based Design of Information Systems: Notes from the Hyperdeveloped World." *The Information Society* 18 (2): 139–144. doi: `10.1080/01972240290075066`. (Cited on pages 82, 86, 96).

———. 2007. *Human-Machine Reconfigurations: Plans and Situated Actions.* 2nd ed. Cambirdge, UK: Cambridge University Press. (Cited on pages 2, 28, 33, 35, 38, 40, 49).

Suchman, L., R. Trigg, & J. Blomberg. 2002. "Working Artefacts: Ethnomethods of the Prototype." *The British Journal of Sociology* 53 (2): 163–179. doi: `10.1080/00071310220133287`. (Cited on pages 28, 39).

Support, R. 2013. "Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS." October 22. Accessed July 24, 2015. `http://www.rackspace.com/knowledge_center/whitepaper/understanding-the-cloud-computing-stack-saas-paas-iaas`. (Cited on pages 64, 66).

Taylor, A. S., & R. Harper. 2003. "The Gift of the Gab?: A Design Oriented Sociology of Young People's Use of Mobiles." *Computer Supported Cooperative Work (CSCW)* 12 (3): 267–296. doi: `10.1023/A:1025091532662`. (Cited on pages 3, 9, 10, 41).

Thereska, E., O. Riva, R. Banks, S. E. Lindley, R. Harper, & W. Odom. 2013. *Beyond File Systems: Understanding the Nature of Places Where People Store Their Data* MSR-TR-2013-26. Microsoft Research, February. `http://research.microsoft.com/apps/pubs/default.aspx?id=184802`. (Cited on page 59).

TodoMVC. 2015. "Readme." March 15. Accessed July 28, 2015. `https://github.com/tastejs/todomvc/blob/0b2c672341/readme.md`. (Cited on page 69).

Toyama, K. 2013. "Reflections on HCI for Development." *interactions* 20 (6): 64–67. doi: `10.1145/2527298`. (Cited on page 91).

Turkle, S. 2007. "Introduction: The Things That Matter." In *Evocative Objects: Things We Think With,* edited by Turkle, S. Cambirdge, MA: MIT Press. (Cited on page 26).

Turner, V. 1967. *The Forest of Symbols: Aspects of Ndembu Ritual.* 12. paperback printing. Cornell paperbacks 101. OCLC: 255259788. Ithaca, NY: Cornell Univ. Press. (Cited on page 76).

Van House, N. A. 2009. "Collocated Photo Sharing, Story-Telling, and the Performance of Self." *International Journal of Human-Computer Studies,* Collocated Social Practices Surrounding Photos, 67 (12): 1073–1086. doi: 10.1016/j.ijhcs.2009.09.003. (Cited on pages 10, 34, 35, 38).

Walton, M. 2014. "Pavement Internet: Mobile Media Economies and Ecologies for Young People in South Africa." In *The Routledge Companion to Mobile Media,* edited by Goggin, G. & Hjorth, L. London, UK: Routledge. (Cited on pages 3, 87).

Walton, M., S. Hassreiter, G. Marsden, & S. Allen. 2012. "Degrees of Sharing: Proximate Media Sharing and Messaging by Young People in Khayelitsha." In *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services: MobileHCI '12,* 403–412. New York, NY, USA: ACM. doi: 10.1145/2371574.2371636. (Cited on pages 41–43, 96).

Wharton, J. 2016. "Exploring RxJava 2 for Android." In *GOTO Copenhagen 2016.* http://academy.realm.io/posts/gotocph-jake-wharton-exploring-rxjava2-android/. (Cited on page 91).

Whitechapel, A. 2013. *Windows Phone 8 Development Internals.* In collaboration with McKenna, S. Sebastopol, California: O'Reilly Media, Inc. (Cited on pages 10, 11, 14).

Windows Dev Center. 2016. "App and User Data." Accessed May 14. https://msdn.microsoft.com/en-us/library/windows/apps/jj553522.aspx. (Cited on page 55).

Zhao, X., & S. E. Lindley. 2014. "Curation Through Use: Understanding the Personal Value of Social Media." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '14,* 2431–2440. New York, NY, USA: ACM. doi: 10.1145/2556288.2557291. (Cited on pages 60, 61).

Zimmerman, J., J. Forlizzi, & S. Evenson. 2007. "Research Through Design As a Method for Interaction Design Research in HCI." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: CHI '07,* 493–502. New York, NY, USA: ACM. doi: 10.1145/1240624.1240704. (Cited on page 39).

Zimmerman, J., E. Stolterman, & J. Forlizzi. 2010. "An Analysis and Critique of Research Through Design: Towards a Formalization of a Research Approach." In *Proceedings of the 8th ACM Conference on Designing Interactive Systems: DIS '10,* 310–319. New York, NY, USA: ACM. doi: 10.1145/1858171.1858228. (Cited on pages 2, 3, 5, 28, 39, 49, 95).