Vrije Universiteit Brussel

FACULTY OF SCIENCE AND
BIO-ENGINEERING SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

May the Personal Information Manager be With You: A Ubiquitous Distributed PIM System

Graduation thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Applied Informatics

Evgeni Ivakhnov

Promoter: Prof. Dr. Beat Signer
Advisor: Sandra Trullemans

Academic year 2014-2015

©Vrije Universiteit Brussel, all rights reserved.

Vrije Universiteit Brussel

FACULTY WETENSCHAPPEN EN
BIO-INGENIEURSWETENSCHAPPEN
VAKGROEP COMPUTERWETENSCHAPPEN

May the Personal Information Manager be With You: A Ubiquitous Distributed PIM System

Afstudeer eindwerk ingediend in gedeeltelijke vervulling van de eisen
voor het behalen van de graad
Master of Science in de Toegepaste Informatica

Evgeni Ivakhnov

Promoter: Prof. Dr. Beat Signer
Advisor: Sandra Trullemans

Academiejaar 2014-2015

©Vrije Universiteit Brussel, alle rechten voorbehouden.

Abstract

On a daily basis, people get in touch with the challenge of analysing and filtering a huge amount of information while performing professional and personal tasks. Eventually, some user data is accumulated and stored in document files such as, emails, articles, notes, etc. Aggregation of those files forms a certain information space of the user. Creation and addition of each new document rises the question of organising and (re)structuring the information space. While people already suffer while re-finding in the usual of organisational structures, the last decade they also use multiple digital devices. Usually, people use different devices to solve different tasks, while the set of the use cases of a device can be limited by its physical characteristics. Moreover, people use different cloud services, which means that the information organisation problem becomes even more challenging. In the end people want to have access to all their information regardless the device they are working on. In order to synchronise personal files over different devices, people often fall back to unified repository services. However, the files that are controlled by a cloud service are doomed to be isolated inside the cloud services. This is because the service makes copies of files and stores them on their server. Hence people use multiple cloud services, but also multiple devices. So, a document file can be copied over different devices and at the same time be used in multiple cloud services which results in disastrous fragmentation of personal information space. One of the main conceptual contributions of this thesis is the integration of different devices and cloud services in one single information space. This provides possibility to keep track of documents which may be duplicated across isolated parts of one's personal information space, in order to prevent inconsistency and fragmentation. Knowing what is where, significantly helps in the re-finding, organising and structuring process. Moreover, as it turns out, people not only access their information space while ingesting or retrieving information, they tend to proactively re-explore their space and reflect on it. As people memorise something, they tend to create mnemonic links between different information pieces and events originating from that particular period of time. By recalling, the linked information pieces are often recalled as well. For example, as in case of Facebook, people can explore photos, events, news, etc. in a time based way using the timeline. Those information pieces can contain additional links to things such as geographical tags or comments, which people re-explore sometimes. However, there do not exist solutions which would make reflection ubiquitous, by making it possible to reflect on all kinds of documents while exploring it on all islands of one's information space. For that purpose an application called EverSync has been implemented. This is an application which successfully targets the described issues. In practise, EverSync can be extended with plugins in order to support multiple isolated information spaces, mainly cloud services. It is implemented using the client-server architecture in order to support multiple user's devices. Moreover, it is a cross-platform application. Since people find it easy to work with interfaces they have a familiarity with, it was a true

challenge to design the interface for the EverSync. The result is ubiquitous, keeps track of documents, their copies and blurs away the boundaries between devices, services and social media.

Contents

1 Introduction

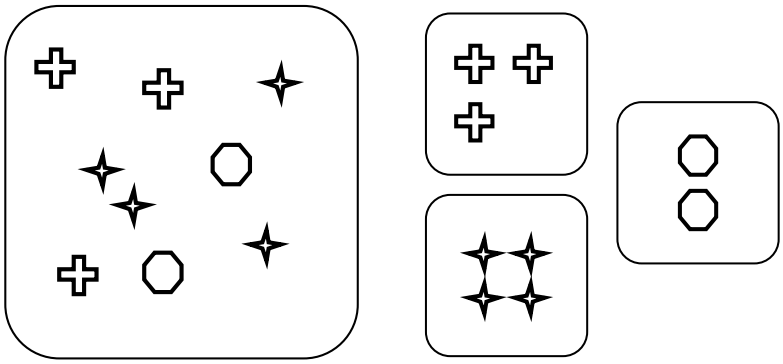
Daily, while performing professional and personal tasks, a modern human being gets in touch with the challenge of analysing and filtering a huge amount of information. Eventually, some user data is accumulated and stored in document files: emails, articles, notes, etc. Aggregation of those files forms a certain information space of the user. Creation and addition of each new document rises the question of organising and (re)structuring the information space. Hereby, in process of time few file folders with organised documents inside, can evolve to huge stores of documents which are difficult to maintain. Clearly, it is very challenging to concretise, categorise and visualise the user's information space. This chapter will introduce how computing devices and cloud services are incorporated in one's personal digital information space. Why and how people use cloud in combination with their main computers and mobile devices. Further, the main purpose and contributions of this thesis will be explained, followed by the description of the overall thesis structure.

1.1 Information on the Desktop

With the emerging ubiquity of modern technology in all aspects of a human life, the boundaries between facets of the physical identity in the real world and virtual identity in the digital world get blurred [55]. However, the distinction between those two worlds will never fade out completely since research on this popular psychological domain shows that it is of the deep human nature to cultivate and maintain multiple identities [25]. Those identities consist of multiple facets and altogether

form the social-self [13]. This means that people behave differently depending on the circumstances and tend to associate different contexts and situations with their different identities and/or adopt their identity to the context of the problem [21]. This psychological subject of multifaceted identities not only involves questions of individuals' behaviour in for example social networks, but also covers the subject of human-computer interaction in the broad sense, and in particular personal information management using modern technology.

Depending on the facets of the developed identity, the behavioural patterns can be categorised into *aggregation* and *segmentation* [57] of information. The mental models that people use hereby is depicted in Figure 1.1. Suppose that each type of figure (a star, a cross or an octagon) stands for a different type of information. This can be books, photo's etc. Segregators try to segment their information in different information spaces. An illustration of segmentation by information type is shown in Figure 1.1b. For example, people keep their books on a bookshelf and store their photo's in one photo album. Of course, we could have supposed that the figure types stand for context related information pieces. Segmentation can be done not only by information type. The important thing here to understand, is that some people keep their things separated. Aggregators on the other hand, prefer to have one single information space where they store all their personal information, for example, a working desk.



(a) Aggregating (b) Segregating by type

Figure 1.1: Mental model of information management.

Regardless the approach, information storage implies (re)finding in a later point of time. Finding and storing information both require some effort. The difference between the two mental models of information organisation lies in the phase of the storing process which requires the most effort. Section 2.1 describes that the aggregation offers very easy storing stage but require much more effort to refind things [35]. Namely, less attention is given to classifying and sorting items which makes it difficult to search. Segmentation on the other hand exhibits a storing phase with heavy cognitive load but then things can be found very easily.

Transposing the information organisation subject to the digital world does not change human's nature and the concept of multifaceted identities remains. Either way, information management solely relies on processes of categorisation, recognition and recall [35, 34]. This manifests in different techniques to organise their files and different organisational structures (i.e. files and piles) that people use, which is described more deeply in subsection 2.1.2. Additional challenge is keep track of context related items. For example, an email about a presentation, a calendar record about this presentation, powerpoint files that will be presented and text files with the text to present. If the powerpoint and text files can be "linked" by placing them, for example, in one folder, the the email and the calendar record are isolated in the mail messenger and calendar application respectively. This implies uncontrolled fragmentation of information.

To solve the aforementioned organisational issues there exist solutions such *personal information management* (PIM) systems. PIM systems help organising personal information, offer one single access point to all files and foresee possibility to create logical links between related items. However, mostly such PIM applications can only work with local files. Meaning that the application only has access to the files on the device where it is installed. In order to support multiple devices often cloud services are used.

1.2 Information in the Cloud

The concept of cloud computing in the broad sense of the term means providing end users possibilities for remote access to dynamic computing resources and applications via a network connection. That is, the end user gets access to computing resources in full compliance with his original requirements. The physical source of the computing resources stays unknown, metaphorically speaking, it is somewhere in the clouds. In fact, the source itself is not important for the end user. The solely important thing is the availability of a communication channel to the cloud. As a rule, the cloud service can providing all kinds of computer resources, including CPU, memory, disk space and communication systems. However, the ideology of cloud technologies is more versatile. It provides convenient conditions for performing all kinds of user tasks. Hereby, under the *convenient conditions* one can understand a whole set of requirements for the service, including a high level of security (in all aspects), flexibility and versatility, scalability and adaptability to specific client needs.

Now that the concept of *cloud* is introduced, we can proceed and focus on the aspect of how people use cloud services for storage of personal information. Hereby, the processes of aggregation and segmentation described in the previous section also hold for cloud use [57]. Before even actually using an application at its full functional capacity, especially based on previous interaction experience with

similar applications and based on the application's presentation and interface design, a user creates certain predictions and expectations about how the application works. That is called a conceptual model. As research shows, people though seem to use cloud services differently for information management than their desktops because of the conflicting conceptual models [26]. People use their desktop computers for different purposes than they use their mobile devices and cloud services. The conflict of the conceptual models can be explained by the fact that the most research in this area is directed towards improvement of some particular type of cloud service, for example [2]. Based on such research, various cloud solutions for information storage and management have been developed, for example, DropBox, Google Docs, iCloud, Evernote, etc. But each cloud application provides its own set of functionalities, its own possibilities for interaction and its own unique advantages. This also implies that each cloud application has its own set of limitations and disadvantages. Hereby, Marshall and Tang state that people have five different use cases for cloud services: "personal cloud repository, shared cloud repository, personal replicated store, shared replicated store, and synchronisation mechanism" [36]. Using a cloud service as a personal cloud repository means that people use it as a single storage place of their personal documents and information. Shared cloud repository means that a cloud service is used as a sharing platform. Whenever a document is downloaded of viewed by another person, mostly the owner removes it from the cloud service. In comparison with the personal cloud repository, the personal replicated store means that people use it not a single central store place, but in combination with other services and storage space on their devices and hard drives. Again, shared replicated store implies using the cloud service to share information with other people. However, in case of replicated store, people tend to keep information that has been shared in the cloud storage space instead of deleting it. The last way of cloud service usage is when people store their information on a cloud storage space in order to be able to access this information on one of their other devices. In that case, the service is intended to keep users' devices in sync.

There's done a lot of research on this field which shows that people use cloud service for their daily challenges of personal information management [17, 40, 57]. However, those five use cases share one common issue – interaction with the cloud. Most of the users' attention goes to the limitations of each application instead of focusing on the interaction [24]. Another point of struggling is the isolation of information inside each application. Almost each cloud service which controls some kind of information, isolates this information inside the scope of this service. For example, files inside the DropBox folder cannot be controlled by Google Docs or vice versa. In fact, this is not only typical for cloud services but also holds for local applications on a device. For example, the emails are isolated in the mail messenger app and are not visible in the file tray which is used to organise files. This problem is described more deeply in section 2.3.

1.3 Problem Statement

The files that are controlled by a cloud service are doomed to be isolated inside this cloud service. Here comes the fact that people not only use multiple cloud services, but also multiple devices. So, a document file can be copied over different devices and at the same time be used in multiple cloud services which in turn make copies of it because of their isolation nature. As a result, if the user makes changes to this particular file either on one of the devices or via the application of one of the cloud services, it does not necessarily mean that also other instances of that file will be updated. Hereby, in process of time, the user looses control of where this particular file is used and stored. Consider the following example situation in order to outline the problem.

Benjamin is on a citytrip and takes beautiful photos with his iPhone. When he arrives at his hotel room, Benjamin takes his Android tablet to surf on the internet because he finds its bigger screen more convenient. Then he surfs to Facebook and decides to share his photos with friends. Of course, Benjamin has foresee such scenario and have had installed DropBox applications on both: his iPhone and his Android tablet. So, seamlessly his photos are synchronised on both devices. Without any problem Benjamin selects the photo from his DropBox application and uploads them to Facebook. Later this evening he decides to somehow record his route so that he can remember it the next time he visits the city. So he creates a note in Evernote and adds some of the pictures with the conjunctions and route directions. At home Benjamin has a Mac computer and at the end of his vacation when he arrives home, all his photos will be available on it since both Apple devices are synchronised through iCloud.

So to summarise, Benjamin has his photos on:

- iPhone
- DropBox account and thus also on his Android tablet
- Facebook
- Evernote
- Mac desktop via iCloud

This implies the following problems. What if Benjamin finds out that he has made two almost identical photos and decides to remove one of them? Actually, he has to remove it manually multiple times from each service and/or each device. What if Benjamin decides to adjust contrast of his photos in Photoshop on his desktop? Except of the iPhone, all the other copies won't be changed automatically. Moreover, he has to remember where he has used and stored the photo that he has adjusted in order to replace it with the improved version. After some period of time, Benjamin won't be able to remember where he has copies of his photos.

Information fragmentation over cloud services is the main problem of personal information management using multiple devices since current PIM systems have no support for the cloud.

1.4 Thesis Contributions

Shortly, the main conceptual contribution of this thesis is the integration of different devices and cloud services in one single information space in order to provide possibility to keep track of duplicated files in order to prevent inconsistency in fragmented personal information. The EverSync application has been developed for this purpose. The application can be extended with plugins whereas a plugin integrates a third party (cloud) service with EverSync. Moreover, EverSync is a cross-platform application. This application contributes in different ways to the PIM research field. First of all because the EverSync application bridges the gap between the cloud and the desktop. Secondly, the application is very extensible due to its plugin system. Thirdly, EverSync integrates with current organisational behaviour. And finally, the application supports personal devices. Hence, by installing the EverSync application on each of user's personal devices and extending it with plugins for the services where users stores his personal information, the fragmented personal information space will be much more manageable.

1.5 Thesis Structure

In this section, a general thesis roadmap will be described by providing an outline of the next following chapters. Firstly, in order to describe the research field of the thesis, main concepts of personal information space are described in chapter 2. This chapter contains an extensive literature review followed by the problem statement. In general, this chapter provides a motivation for this thesis. Mostly, the application development contains a phase of prototyping and definition of requirements before the implementation. The development of the EverSync application for this thesis is not different. The **??** contains description of those two early and at the same time very important steps. In chapter 4 the most interesting aspects of the implementation are described. Challenges and solutions are illustrated and discussed. The subsequent chapter 5 contains illustrations and screenshots of how the developed EverSync actually can be used. An example use case is shows how the EverSync works in real circumstances. Finally, the chapter 6 concludes the dissertation, wraps the main challenges and how they are solved with the developed EverSync application. In this chapter any possible future work is also described.

2 Literature Review

In this chapter a description will be given of different aspects of *Personal Information Management* (PIM). Organising personal information space makes it easier to re-find documents later. In fact, storage and re-finding of information are the main structural challenges of PIM. They will be introduced by a description of the Memex system which was ahead of its time and intended to be the ultimate way for information re-finding by association. A description will be given of different types of information spaces and their interrelation. However, the issues around re-finding which will be discussed, get another dimension when the information space spans multiple devices. As we will see, multi-device usage implies information fragmentation over the devices. This triggers different challenges and their solutions. After having discussed the state of the art, the main issues and challenges will be summarised again.

2.1 Challenges of PIM: storage and re-finding

A familiar sight – a working desk with a computer, couple of books, files, sheets of paper and sticky notes. People store their information everywhere and organise their physical and digital information space to be able to re-find things from all these different places. Nevertheless, organising information doesn't necessarily mean ordering everything alphabetically or in some other strict order. Information can follow more subtle organisational patterns where everything looks like a mess at the first sight.
The PIM research targets the personal aspect of the study on how people keep,

organise and re-find information. Those basic PIM activities were described in the Keep, Organise and Re-find (KOR) Theory of W. Jones [27].

2.1.1 Keeping

Worldwide, an endless stream of information is being generated every day. Just think about all the newspapers and thousands of emails that are daily sent. But not all this information is meant and accessed by everyone. Some of this information will always be public accessible (e.g. books in the library) while other information such as an email will almost always remain private. This is the main difference between the *public information space* and the *private information space*. People do not want to keep every single piece of unordered information since they will not use it anymore so it is not worth to keep. For example old, payed house rent bills. Mostly people throw them away after having transferred the money. However, keeping everything publicly accessible is not what people do neither. Both information spaces are illustrated in the same Figure 2.1 in order to better understand their interrelation. There are some information pieces that are strictly personal, for example post-its and personal notes. This was not public and will not be public. It is created to be private and therefore is part of the personal information space. The public information space consists of things that can be accessed by everyone, for example books in the library. However, there are information pieces which make part of both spaces.

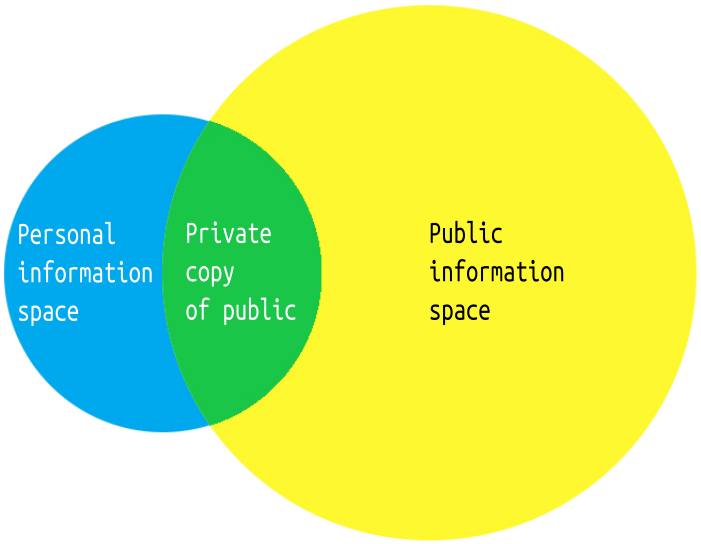


Figure 2.1: Information spaces intersection.

For example, a historical textbook of pictorial arts which can be added to someones personal information space. This book contains many little reproductions of the original paintings (i.e. pages with images of the paintings). The original canvasses are still publicly accessible in the museums, though they are copied to someones personal information space via the textbook. These paintings are copied to the personal information space. However, copying is not the only way of making public information piece private. One can translocate it. Hence, buying the original canvas and hanging it up in the living room is translocating. Of course, the other way around is also possible. A writer publics his finished book. It is then printed many times and his original manuscript is copied into the private information space namely the libraries. To illustrate the example of translocation from private to public information space we can again use the example of paintings. Museums posses mostly the original canvasses, not reproductions. Therefore, before being exhibited, the canvas is translocated from the personal information space of the painter to the public information space of the museum.

2.1.2 Organising

From the examples given in the previous section, it becomes very clear that the information space of an information piece can change. However, whether only you have access to it or multiple people, it has to be accessed anyhow. Accessing means knowing where to go, look and search. To increase the easiness of re-finding and fasten this process, the documents have to be organised. There are different ways to organise the information. Although, the information can follow different organisational patters including the subtle ones where at the first sight everything looks like a mess. The two main strategies of information organisation given by Malone [35] are *files* and *piles*. Files are indexed documents and the re-finding happens through recall. One has to recall where the information space is stored, he has to go through the full path to get it. Piles are nameless collections of unindexed documents and the re-finding happens through recognition. By looking at a pile of files one will recognise it and use the spatial awareness to re-find the desired document. However, there exist also a third strategy which is very widely used. It is called *mixing* and partly combines the two previous approaches trying to be the perfect match.

	Files		**Piles**	
	Groups	*Elements*	*Groups*	*Elements*
Titled	?	Yes	No	?
Ordered	?	Yes	?	No

Table 2.1: Organising information of Files and Piles.

Table 2.1 defines the strict rules to consider an organisational structure a pile or a files. The "Yes" make a characteristic obligatory and "No" not allowed. The "?" means that this characteristic may occur, but it is not obligatory and not disallowed. For example, all the elements of a file have to be ordered and they all have to be titled, while groups of files can have no title and to order. Violation one of the restrictions (i.e. "Yes" and "No") results in a mixing structure. In fact, mixed organisational structures are the most popular ones. For example, in physical space they use labeled file trays. Documents inside each tray are unordered, but each tray has its own label as can be seen in Figure 2.4b. A good example for digital space are folders in Mac OS. Those are system folders, which of course have a folder name, however they do not have to be ordered. A folder can consist of one or more piles.

Files

One of the most popular strategies in organising personal information is *filing*. In this context, files are organisational structures not documents. Filing can of course be applied on both information spaces, namely digital on your computer and physical on your desk.

(a) Digital files (b) Physical files

Figure 2.2: Files in different information spaces

Files consist of documents but can of course contain other files and form tree structures, such as shown in Figure 2.2a. Depending on the object each file has to have a title (e.g. book title) or some other label (e.g. theme of a book). Furthermore, the documents need to be ordered by these labels. Again, depending on the situation this can be an alphabetical order, based on the file size or the chronological order. The main restriction in this organisation strategy is that the files are assigned a place in the file structure, and may not be moved around in order to preserve the order. For example, public libraries implement this approach using

the *Universal Decimal Classification Model* [38] where each label is mapped to a decimal number. The Figure 2.2 illustrates files in personal information spaces. However, there are classification issues with this approach. First of all, what happens if a document belongs to multiple categories. For example, post-it notes. They can be reminders with deadlines, shopping lists, to-do's, and they all have a topic, creation date, etc. The ambiguity of classification makes it impossible to predict which of the adopted structures will be the most useful for later usage [15]. For example, ordering post-it note by topic might seem the most reasonable by time of creation. But several days later at retrieval, it turns out that ordering by date of creation would have been a better choice. Secondly, the attempt to provide as much contextual information as possible with the conviction that this will ease the re-finding. But this leads to very long and knotty labels.

Piles

(a) Digital piles in Bumptop (b) Physical pile

Figure 2.3: Piles in different information spaces

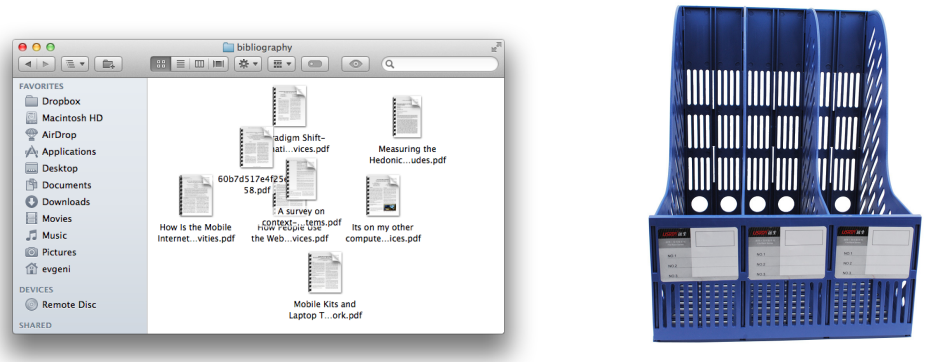
A second organisational structure are *piles*. Basically, these are stacks of documents. Figure 2.3a illustrates a typical digital pile organisation in Bumptop[1] application. Figure 2.3b show physical pile, which is basically a stack with papers. The documents are not strictly ordered and are not classified into subsections. Although a document may have a label, the pile as a whole may not. Nevertheless, the lack of labels on the document and no explicit order in the pile, makes one think that re-finding information is impossible. In fact, that's not true. The spatial awareness which is basically the knowledge of where which information is stored makes it possible to re-find documents [15]. In case of PIM, mostly the user of a pile is the same person as the creator of it. Therefore at creation time

[1] http://www.bumptop.com

of the pile this person certainly knew where is which document. This knowledge fades over time and gets triggered again by recognition [35]. The actual recognition comes when browsing through a pile in order to find the desired document. Since piles do not hold any order of documents, the worst case is that one has to pass through every single document. But while doing so, all the encountered documents are recognised which triggers spacial memory [46]. Teevan [51] states that recognition is easier than recall for humans and therefore more natural. However, going through one pile might be done fast but doing so with multiple piles might be very time consuming because one has to go through every single file of each pile. Thereby, instead of using labels, piles provide the possibility to preserve context information. Documents can be rotated, flipped up side down, deviate from the centre of the pile pointing some particular direction. All this context information may be used for easier re-finding.

To summarise, this means that storing is fast and doesn't require any effort because the newly added document doesn't have to be placed on a particular position according to the overall ordering pattern and re-finding is not that difficult because of the spatial awareness and recognition. However, re-finding information in one pile might be easy, using multiple piles makes it very challenging.

Mixing



(a) Digital mixing in Mac OS

(b) Physical mixing with separators

Figure 2.4: Mixing structures in different information spaces

An easier re-finding of information is the main reason for organising stored documents. Regardless the used strategy, this PIM activity has to be very performant. Actually, people do not follow strictly only one organisational pattern.

Mostly they mix the strategies and depending on the situation try to get the advantages of both approaches. Mixing can be considered as a combination of the filing and piling strategies in order to compromise their advantages.

2.1.3 Re-finding

From the previous sections, one can conclude that adopting an organisational structure in your personal information space surely improves the re-find activity. However, not everything is flawless. Filing structure requires lots of effort at organising phase, but it doesn't guarantee that the applied classification is the one that is needed at every re-finding phase. A document may belong to different classes which may be different at organising and re-finding phases. Piling bypasses this issue by not requiring any of such prescience while organising. However, having multiple piles or big piles may result in passing through every single document in every pile in case one really cannot trigger the spatial memory. Mixing structures exist in different varieties and combine advantages and disadvantages of the former two organisational approaches. Another huge issue which is very common, is data fragmentation. Although different piles or files can make part of one single information space, they can be located on different workplaces and devices. It is an extremely challenging task for application designers to provide the most desirable system which will implement the best of the different organising approaches.
In the following sections different services will be discussed more deeply, but lets first sketch the current situation. Disregarding which type of organisation someone prefers and uses the most, the preferences will apply not only on physical files but will be transferred to the digital information space as well. For example for files, there exist solutions like Dropbox which will synchronise the files between one's devices. If devices are considered as distinct small instances of one's digital information space, the it can be said that applications such as Dropbox combine the different information spaces. However, this 'combination' is not flexible and basically synchronises an organisation structure between the devices disallowing the user to have different organisational structure on different devices. This is the so called *System method*. To solve this issue there exist applications such as the Maistacker [30] and the Personal Information Dashboard [1] which makes the synchronisation a more flexible manner by targeting the more unified PIM system implementation. However, as we shall see in the following sections, all those solutions are database based. Database based means that these solutions are a form of centralised storage place where all the files are stored. Another drawback of the most existing PIM applications is that they provide the possibility to manager local files only. At the current moment, none of the widely known systems include user's information space from third party services such as Facebook for example. Though, Facebook is part of one's information space since it stores the photo's. And memorising where the user has his photo's stored and re-finding all the 'copies' is a serious challenge for a human brain. Nevertheless, as already has been said,

those problems will be discussed more deeply in the following sections.

2.2 Current PIM Systems

In this section a description will be given of the main concepts about how PIM systems work on a desktop. Beginning with Vannevar Bush and his Memex machine, which lies at the root of all current PIM systems, we will proceed to some recent solutions. A brief explanation of their strong and weak sides will lead us to the conclusion that some functional disabilities result into cloud use.

2.2.1 Memex vision of the PIM concept

Indexing eases searching and re-finding information significantly but it also has its own disadvantages. We are all familiar with the situation of encountering a familiar term (e.g. description of a process, phenomenon, formula) while reading a paper which requires deeper explanation. This formula is not new, it is been explained in the math course few weeks ago. But remembering this fact, doesn't guarantee the retrieval of information. Was it mentioned in the math textbook or in the slides? Or maybe it is been explained by the course assistant and the formula is somewhere in your own notes? Knowing that this is a mathematical formula is not enough. Organising the personal information space is of course essential for information re-finding, but in general it doesn't solve everything.

In 1945 Vannevar Bush described [14] the idea of a system suited for structuring and storing information with possibilities for its replenishment, rewriting and sharing with other terminals. This machine, which he called *Memex* (i.e., MEMory EXtender) was completely mechanical and as history shows was never implemented. Vannevar Bush explained the colossally differences between the decomposition of information in subclasses for index based storage and the human associative memory. Our memory manages information using associations by building a network of paths from one concept to another. Memex looked as a writing desk with projectors for displaying information. Memex was equipped with a subsystem for dry photography which could be used to make new shots and add them to the microfilms storage and make new text documents. Figure 2.5 is a sketch of the system with all its components.

All the information was stored on a microfilm and could be accessed through a keyboard, buttons and levers. The main advantage of using micro-film for storage is its size and therefore the portability. Users could easily share information by interchanging their micro-films. Another interesting component of the Memex system is the dual projector with the ability to cross-reference the text. Bush proposed to make annotations to the documents which could make it possibly to switch between different documents. In fact, this was a system of cross-references, the prototype of the modern hypertext, as we know it today. The principle of hypertext is to

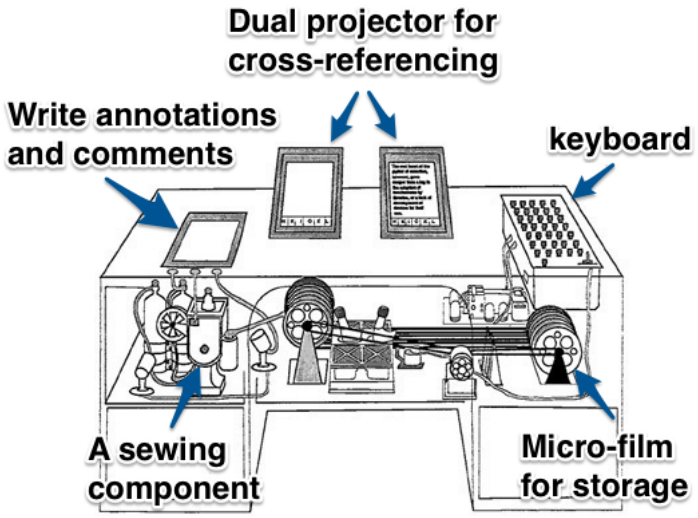


Figure 2.5: Memex – A hypothetical information retrieval and annotation system.

associate information through "links" into a coherent organisation. However, the current hypertext implementation on the internet is not how Bush described it. His intention was to have *bidirectional* references between documents. The entire idea of memex was based on easier information re-finding through associations. As Bush stated: "The natural means of item binding is by association". It is not the case with the current implementation of the hypertext system, see Figure 2.6. There is no information on a webpage that can tell which and how many other pages contain references to it, there is only one-way traffic. This approach renders such a trivial task as information re-finding by association almost impossible.

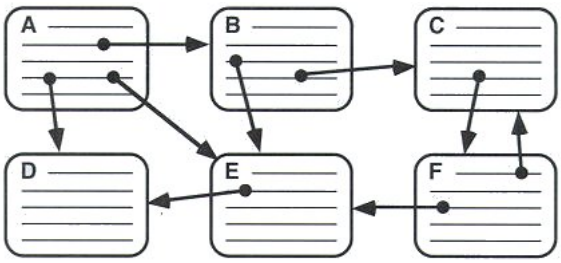


Figure 2.6: Current implementation of hypertext system on the internet.

Memex was never built as described by Vannevar, but its ideas knew some realisations and are very actual nowadays. Especially for the PIM topic. The study on how people organise their personal information, is the main goal of descriptive PIM

research field. We are overwhelmed with information and the documents have often contextual interconnection. For example, I received a powerpoint presentation via email, created an "appointment" in the calendar application, created a reminder in my reminder application to make some corrections in the powerpoint and already made a note in Evernote with a draft of these corrections. All the information in these documents is interconnected and yet it is created in different applications, stored on different places and possibly on different devices.

Memex was a solution for easier re-finding of previously stored information pieces by providing associative links between them. As we will see in the next sections, the personal information gets fragmented over different devices, but is also cluttered over different places on one single device. Therefore, providing links between documents is a very challenging task. The description of the main challenges of information organisation will be followed up by a discussion of the core activities involved in personal information management.

Inspired by the Memex, many researches tried to reincarnate it in many different ways. With the popularisation of the Internet, the idea of Memex lead people to develop something like Semantic Web [10], an idea of Tim Berners-Lee who designed the World Wide Web. This idea manifests the easier usage of the web content by semantic links which make the re-finding easier. Links are created using the Resource Description Framework (RDF). Later, the ideas of Semantic Web were transposed to the desktop, which resulted in the Semantic Desktop [44]. And one of the widely known implementations of the Semantic Desktop is the Haystack [30, 31]. Haystack is an extensible application which uses RDF to create links between information parts. It can be extended by plugins, in order to include content from different applications. However, cloud services are not supported yet. The plugins basically scan the content of an application and import in into the unified triple repository, which makes Haystack an layer above all the installed applications.

There also other PIM approaches, for example, using the time scale [19]. This PIM approach is based on the idea to organise information and documents in a chronological list, instead of classifying it in categories and is called LifeStreams. Moreover, this approach supports the automatic document sorting in the timelines, so that the user does not have to do it manually. There has been lots of research on how personal information can be organised into timelines [5, 42]. As always, theoretical research results into practical applications. On of the best known implementations of this idea is the MyLifeBits [20], which was developed in 2002 Jim Gemmell et al. So far, this implementation of the LifeStream can be seen as the most succesfull since no other application has been so widely known.

2.3 Information Fragmentation in PIM

Firstly, a description will be given of why there exist such a variety of devices. It will be shown that the majority of the tasks get partitioned over different (mobile) devices because of their characteristics. Then, some observations will be made about the fundamental issue concerning usage of multiple devices i.e., data fragmentation. Finally, other issues will be discussed that lie at the origin of data fragmentation.

There exist a wide variety of devices: smartphones, PDA's, tablets, e-readers, music players, notebooks, etc. We use them on a daily base for several activities such as: reading mail, messaging, personal bookkeeping, entertainment, etc. They are all differing in physical characteristics, size, weight, interaction and computing capabilities, which makes one device more suitable for a particular task than another. For example reading an email is easier on a tablet PC due to responsiveness of its capacitive touchscreen, while an e-ink enabled e-reader is more suitable for reading with less frequent scrolling and/or page flipping. But even here, an e-reader with bigger screen is easier for inspecting technical drawings and scheme's, while its smaller variant is better for reading books and articles. The shortcomings of one device, like very small screen for watching movies is a huge advantage in concerning portability. The point of all this is to show that people do not use one single device for everything. There exist different tools, which people do buy and use.

Market segmentation is very powerful marketing mechanism, it helps the companies to increase their revenue and consumers to fulfil their needs [59]. But here exist not only segmentation upon personal characteristics like gender, age, lifestyle and revenue. Observation and analysis of the customer needs, which defines their behaviour, lets companies develop market segmentation strategies. These consumption attitudes can be measured [4, 3] and used in market segmentation, which results in a whole variety of mobile devices. In general the sales are driven by (1) utilitarian and (2) hedonic purchases. A utilitarian motivation incites to buy out of a fundamental need to survive, mostly it concerns goods that are used daily (e.g., electricity). Hedonic motivation is based on desire which is artificially created by marketers (e.g., the newest version of iPhone, while you already posses one of the previous generations).
Those utilitarian and hedonic motivations to purchase, are defined by the product as a 'whole' [59]. This mean that not only the physical form factors of the devices are important, but also by the supported services. However the combination of these two do not completely define the usage of the device, but in contrast to a stationary desktop, mobile devices offer an extended context space [45], for example, geoposition, touchscreen, device orientation, etc. Besides the home environment, a mobile device can come in handy on other locations or even on the move, when one want to check some previously stored information. A study on the usage of mobile

services [11] shows that people spend nearly an hour a day using mobile services (on average) and the usage is determined by different contextual patterns [56]. For example, a file manager application is mostly used at home, and listening radio is mostly done while being on the move. Most *unexpected* tasks, such as reading an important document before going to a meeting or showing a holiday photo to your friend(s), are mostly done using mobile devices. All this information requires to be always available and thus most likely not only accessible from one particular device. Therefore it is organised and maintained via different devices. In fact, almost 70% of the tasks get partitioned over multiple devices [32].

Usage of multiple devices encourages people to start synchronising their data [41]. In the end people want to have access to all their information regardless the device they are working on. For example, while surfing, it would be nice to have access to their bookmarks which were created on the desktop [29]. It is not hard to share browser resources [29] between mobile and PC (i.e., bookmarks, browser history, search history, etc.). But having your bookmarks synchronised is just a fraction of the device-independent challenge. Things get more complex when trying to organise the personal information space (PIS). The biggest arising problem is that most devices and current services are not user-centric. They just do not 'know' that they are only a fraction of a bigger puzzle [37], so the implemented systems are doomed to be in isolation. It requires a lot of organizational effort to manage information across devices. Oulasvirta and Sumari [41] and Dearman and Pierce [17] have done the most significant research on this topic, i.e. issues that people meet using multiple devices (desktops, laptops, mobile devices).
In reality, almost 70% of the tasks get partitioned over a desktop and mobile devices [32], and even more, users not only perform different tasks on different devices, they split one task and solve the subproblems on different devices.

The described isolation problem implies information fragmentation over multiple devices [8] and consequently organisational challenges [17]. To decompose this problem, different synchronisation issues have been resolved. For example, synchronising resources of a web browser [49, 16], synchronising contacts and address books, automatic sharing information between mutually held devices and complementing each other's interaction interfaces [23]. But such applications were mostly designed for ad-hoc problem solving and do not support PIM related activities. Therefore besides data fragmentation over different devices caused by the multi-device paradigm of modern life, data can also be cluttered over different locations on one device.
For example, email clients were not designed with PIM in mind, but are heavily used for managing personal information. People send themselves emails with important information as reminders [28, 60], because while checking the mailbox, one will see the (flagged) mail – reminder. The email clients are often used for storing and managing the contact lists [61]. The mails are even organised into folder structures [6] for easier retrieval. But despite the fact that email can be used to store

and manage information, it is not an all-round PIM solution. All the information stays inside the email. For example, even if an email client provides a built-in calendar, switching form client applications means loss of all the appointments.

Email is just an example, but most applications that try to unify the fundamental PIM functions of keeping, organising and re-finding, do not provide support for PIM activities and handle in *isolation*. Such application-centric approach results in data fragmentation over different applications.

To illustrate the described fragmentation problems, consider the following example. An office worker receives an important email with description of some project. Firstly he creates a reminder, this even might happen inside the email client. Then the whole project gets decomposed into atomic to-do tasks. Those to-do's were created as a note in Evernote, because that's the program specially designed for such tasks. Thirdly, now that all this is done, the project can be started and initial document files in the file system are created.
All this means, that the next time that the worker wants to continue his work on this project, he has to recall at least the following things: checking the mail client for reminders, location in Evernote where stored the to-do's are stored, location on the file system of the first versions of the document. When the worker uses multiple devices and wants to resume his work somewhere else than his office computer, he can foresee synchronisation. For example Dropbox synchs the file system, Evernote also has the possibility to synchronise, the emails can be retrieved via SMTP or IMAP from the mail server. But what about the reminder inside the mail client? Mostly they do not get synchronised. Besides that, if any of these applications is not installed on one of the devices, it will be impossible for the worker to resume his work on this device. To conclude we can say that the challenges concerning information re-finding in one's personal information space are augmented with other issues related to multi-device usage such as information fragmentation over devices. Mainly, there are two fragmentation problems. (1) Physical fragmentation. The devices form in fact standalone information spaces, instead of extending the existing one. In order to re-find something, one has also to remember on which device it is stored. In case of a mobile device, spatial awareness might be neglected since it does not preserve any contextual information. Therefore it is called a mobile device, since it is not related to any location which can trigger cognition in terms of refinding. To solve this issue people synchronise their devices through cloud services, see Figure 2.7.

Figure 2.7: Different devices in the cloud.

This approach should implement one single digital information space that spans over all the devices. However, the synchronisation happens on the level of applications and the fact of having different applications synchronised does not solve all the problems. (2) Digital fragmentation. The consequence of the described isolation problem of each application, is that only the data inside this particular application will synchronised over different devices. Applications do extend an existing information space on a particular device, but they do not have access to the whole information space on the device. They also do not know which information is stored in which application since they are all isolated from each other. Therefore, the synchronisation happens only inside each application, synching its own documents, see Figure 2.8 for illustration. For example, synchronising a contact list over multiple devices mostly will not provide the possibility of accessing all the emails that are received from that person.

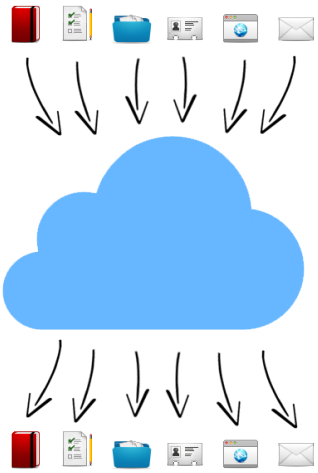


Figure 2.8: Defragmentation over applications.

2.4 Information Fragmentation in the Cloud

The phenomenon that modern technology influences the way that people approach their daily challenges [54] is widely described in section 2.3. People use multiple devices for different purposes which leads to information fragmentation over devices. Cloud services try to target this problem. In this section, we will discuss how people use cloud service. The section then proceeds with a survey and evaluation of several cloud services and PIM systems.

2.4.1 Users' Behaviour in the Cloud

As discussed in section 2.3, people tend to use different devices for different purposes. For example, portable devices are used for different purposes than a stationary desktop. While in case of devices, task separation happens based on the physical characteristics of the device and the easiness of the performance of the task, in case of cloud services it mostly happens based on the available functionality of the service. It is for example simply not possible to use Facebook as a synchronisation tool for your file system. It was not designed for that purpose and does not provide the needed functionality for this. Marshall and Tang performed a research on that field and found that people have five different use cases for cloud services [36]. As they state, people mainly use cloud service as a: "personal cloud repository, as a shared cloud repository, personal replicated store, shared replicated store, and synchronisation mechanism". This topic is more deeply discussed in section 1.2. Considering their work in combination with other major researches performed in this area, such as why [43] and how [33] people use social network services, cloud storage services [18, 36], and any other cloud service in general [58], user's behaviour in the cloud can mainly be classified into following categories. People use cloud services for organisation of their personal information. For this organisational goal, different individuals use cloud services in a variety of different ways. Some people try to use it a single storage point, as a synchronisation tool, as an extension to their local storage. This is goal is can be achieved by services such as Google Drive, iCloud, Dropbox etc. Then there is the communication goal. Social network services are cloud services and are used to communicate upon the documents and shared information, but also to extend each others personal information space by text messages. And the final main goal of using cloud services is the reflection aspect. The best example of this is the timeline of Facebook. People proactively search reflection [48] and therefore visit for example, Facebook or Twitter service in order to reflect on content and events in their life.

2.4.2 Types of Cloud Services

Now that we know three main behaviour patterns of users in the cloud and what they need, we will discuss some widely known cloud services that fulfil those needs.

Dropbox is one of the most popular and widely used cloud services. As previously described in section 1.2, Marshall and Tang state that it is mostly used as a file repository, a tool for sharing content and as a synchronisation tool [36]. Dropbox is a service that synchronises the folder structure and its content across the devices. Dropbox also makes it possible to manage files without 'syncing' any device by providing a web interface where users can create folder hierarchies, upload and manage files and file versions. So, it can be said that Dropbox is not such a pure file system syncing application, but more a hybrid solution between a cloud service and file system application. When installed on a device, Dropbox provides the possibility to selectively synchronise folders, detects file modifications and keeps everything in sync with the storage space on one of the Dropbox servers. Therefore, it can be classified as a unified repository.
Another very very interesting cloud service is Evernote. In comparison with the previously described service, Evernote implements a *pile* file structure. In Evernote, users can create notes, include files in them and categorise all this in notebooks. So as in pile structures, we have the notebooks (piles) and notes (documents) in them. However, besides that, Evernote also makes it possible to assign several types of tags to notes, to makes the re-finding easier. Evernote also provides client applications for almost every popular operating system including for mobile devices. However, there is no intervention into the actual file system of the device, since Evernote does not target to store or sync local files. Though, if needed, files can be included in notes. But those files will be duplicated and uploaded to the Evernote service, which means that there will be no further tracking of modifications to the original file content. Files inside notes have to be updated by the user, if the content changes. Which means, that users have to remember which files, and in which notes are used. Since there is also no possibility to gather a list of all files imported into Evernote.
In comparison with the general document synchronisation tools such as previously described Dropbox and Evernote, there exist cloud services which are targeted to a more specific file types. For example, Flickr is a cloud service for storage and management of user's images and photos. Flickr officially provides client applications for operating systems used on mobile devices (i.e. iOS, Android) but also a Windows and Mac application. Because Flickr provides an API for developers, there multiple third party applications available to interact with Flickr. Besides its main purpose of storing and sharing photos, Flickr also implements a social aspect by providing the possibility to make comments on photos.
However, there are cloud services which mainly target that social aspect, such as Facebook. It is also worth to mention that Facebook only provides client applications for operating systems used on mobile devices (i.e. iOS, Android, and Windows Mobile). Of course, Facebook is a social network and does not position itself as a PIM system. However, its content becomes part of someones personal information space. People can upload images and photos while others users can make comments and rate those photos by 'liking' them. Besides the photos, albums,

calendar, contacts, favourite movies and music, geographic tags and visited party events, those comments and 'likes' become part of someones information spaces as well. The main property of Facebook which makes it so different in comparison with the non-social PIM systems, is that it is designed to let users modify (and actually be part of) someone's else personal information space. In order to position Facebook in the research field of PIM, we can rely on the feature of (endless) scroll with all the events of a user. This makes Facebook a *timeline* PIM system, firstly described in 1996 by Freeman and Gelernter [19], long before the launch of Facebook in 2004. Another rather unique feature provided by Facebook is the 'year summary'. Around the period of New Year, Facebook summarises all the major events of a user and reminds about them. This promotes and stimulates reflection on friendships and relations [50].

2.4.3 Solutions for Information Fragmentation

In what follows, a survey of some existing free cloud systems (free usage without direct payment) will be described. The emphasis of the following systems lies on the fact that they are *private* systems. They behave as cloud services, however, the user preserves the full control and the ownership of the documents. Meaning that the application has to be installed on one of the user's devices which will from then on act as a central server and repository. The documents are accessible from outside of the local network, depending on the fact if the service supports multiple devices of course. The users decides and knows where the files are physically stored, owns and controls the information and documents, that is why those cloud services are called the private cloud services. Note that those are file management systems, so they refer to *filing* which is described in subsection 2.1.2.

Developed by Novell company, iFolder[2] also implements some core functionalities of a private cloud – distribution over multiple servers, synchronisation of the clients and web-interface.
The largest drawback of iFolder is that in the Linux world, only openSUSE offers a full support. Only this Linux distribution gives the possibility for one-command installation. The reason for this is that Novell company acquired SUSE Linux in 2004. Of course, there exist several manuals to perform the installation on other distributions [39]. However, unfortunately I was not able to install iFolder on Ubuntu distribution. The specific package versions, which are needed for the system to work are not available anymore on the official Novell website. Probably, someone with more technical background or experienced Linux user would be able to handle the issue.

[2] `https://www.novell.com/products/openenterpriseserver/features/online-file-storage-ifolder.html` Last accessed on 08-06-2015

The second discussed system is called OwnCloud[3]. First impressions of using this cloud system are good: modern web interface, possibility for online viewing (txt, pdf, odf) and modifying (txt) documents. Management of tasks, calendar, contact book, etc. Searching for content in documents. Synchronization happens through WebDAV protocol, and the most important – here is possibility for mounting local folders and external storage devices through FTP, Samba, etc. Despite the fact that there are some minor bugs and instabilities, the project development is very much ongoing. The whole OwnCloud project is open source, which is its main advantage. All the issues can be checked on the Github account, and every release shortens their amount. Besides that, OwnCloud provides different client applications for almost every operating system, including mobile devices. There is also possibility for server-side file encryption.

Thirdly is the SparkleShare[4]. According to the official website, it has been developed by couple developers who complained about the lack of a good collaboration tool. That is the reason why this tool not only supports the privacy of the information in the cloud, but is also intended for frequent collaboration. SparkleShare is an open source [12] cloud application based on the Git versioning system, and in fact, is a wrapper around Git. Therefore, the used approach is very similar. A repository has to be created, and then a SparkleShare client connects to it. The clients perform the automatic synchronization, which is in contrast to Git very similar to Dropbox, and maintain the versioning between themselves and the remote repository. Due to its Git base, arise several drawbacks which are typical to such version systems. For example, each client has a complete local copy of the repository, which is in case of lots big files may be not suitable. To solve this issue and not to duplicate the data locally, there exist so called "lazy" access using *git-fs*, but it supports in this case solely read-only access to the repository. Also, this assumes certain knowledge about Git, and can be completely inappropriate for users not familiar to software development.

Then there is Syncany[5] which is a very promising open source [22] Cloud system supporting FTP, IMAP, WebDAV, Windows, NetBIOS/CIFS, SFTP/SSH, data encryption, etc. But this project started in 2010 and is still in development phase. On the official website
(www.syncany.org) the developers report: "19 Oct 2013: Believe it or not, we are still working on Syncany. We recently moved to Github for the main development." A the moment of writing, the latest commit on Github was 3 hours ago, so the development is still very much ongoing.

[3] `https://owncloud.org` Last accessed on 08-06-2015
[4] `http://sparkleshare.org` Last accessed on 08-06-2015
[5] `https://www.syncany.org` Last accessed on 08-06-2015

The intention was to mention only the free systems (no payment needed). The AeroFS[6] provides fully private deployment for a fee, but is still worth to mention. The free variant has just the same functionalities as the paid variant, but allows only three clients and requires user authentication, while private cloud supports unlimited number of clients and doesn't require any authentication at all.
In fact, AeroFS is a peer-to-peer network, which in contrast to other described private cloud services, stores the files not only on one central storage server. The system is completely distributed and implements complex algorithms for data replication. Though, it is possible to assign the role of a central server to one of the clients. This provides one single fundamental advantage, that is, extra data duplication in case that all the other clients get disconnected and you have to access a files which is actually stored on other device. Besides the fact that it only supports 3 clients for no charge, the are no client applications for mobile devices, which is unacceptable for personal private cloud, however corporations that are used to have private cloud do not expect their employees to work from mobile devices.

It is also worth to mention the Pydio[7] which is a reincarnation of the former AjaXplorer, with new interface design and an extended feature set. From a technical point of view Pydio can be described with the following properties: online opening en viewing files (txt, pdf, zip, images and multimedia), editing files (txt, doc, xls), assign different access right to different clients, adaptive user interface for mobile devices (iOS, Android), file search engine. External storage devices can be connected through FTP(S), Samba, WebDAV, IMAP, POP.
Pydio is very similar to OwnCloud. Though, it provides some more extensive set of document manipulation features via the web-interface, even relatively serious image editing. Such functionalities are aimed to provide not only a cloud storage system, but all-round cloud computing environment, which may be considered as overkill. Though, such broad variety of functionalities makes Pydio very greedy for computing resources of the server it is deployed on. It is worth to mention that, in contrast to OwnCloud, it is not open source. However, it provides an API for easy integration with third party plugins, including external file versioning systems. In fact, there are lots of external plugins which extend the core functionalities in every possible way.

Finally, Amahi[8] is actually a platform for a home server in the concept of a smart home. Therefor, its media orientation is very noticeable because it supports: Squeeze server, DLNA server, Gallery 2, UPnp server uShare, media streamers Jinzora and Ampache, media libraries OpenDB and VCD-db, torrent clients, etc.

[6] `https://www.aerofs.com` Last accessed on 08-06-2015
[7] `https://pyd.io` Last accessed on 08-06-2015
[8] `https://www.amahi.org` Last accessed on 08-06-2015

It also provides possibility for functionality extension through plugins. From technical point of view, Amahi supports VPN, Samba, WebDAV (Outlook, iCal), etc. Nothing extraordinary here.
Although the installation happens through the terminal, which expects certain IT-knowledge from the user, Amahi will be persistent and very unceremoniously. It will install a graphical configuration panel, *change the IP-address of the server, enable the DHCP, restart the server*, and in general "make itself at home". Of course, it can be easy for users with very little familiarity in setting up a server, but the terminal installation indicates that Amahi expects it to be otherwise. Such behavior can be very annoying, especially if you do not know when and how will Amahi interfere the next time.

2.5 Summary

To summarise all this together, it can be said that the popularisation of mobile devices is a recent phenomenon. Most research has been done within the last decade, and is still very much ongoing in this active field. We discussed that people can organise their information in different ways, such filing, piling and mixing. Those three organisational structures can be applied on both, physical and digital documents. We also discussed that in the daily life, people use multiple devices to perform different tasks. While this task separation between devices is mostly based on the physical characteristics of the devices, people try to keep them in sync through unified repo applications. Task separation over devices implies information fragmentation over those devices, which people try to solve in different ways. Mainly by using unified repositories, which are inspired by the Memex machine. People try to keep the devices in sync in order to ease their process of re-finding information and documents. However, the unified repositories may help to solve the cross-device information fragmentation, they augment the problem of cross-service information fragmentation since most of those services form an isolated information space without being able to interact with each other and exchange information. Oulasvirta and Sumari [41] wrote a paper about how people manage multiple devices and inspired Dearman and Pierce [17] to perform significant study of this topic (almost 100 citations). There exist solutions for managing and synchronising information (bookmarks, address book, etc.) and related activities which require re-finding of this information (browsing, reading emails, listening to music, etc.). However, so far, there is not yet any ultimate solution for the cross-device and cross-service information fragmentation problem. Moreover, while investigating this subject in general, and in particular user's behaviour in the cloud, we found out that besides the information storage goal, people use cloud services for at least two other main reasons. Namely, for communication and reflection. The communication process extends the personal information spaces of the communicating

individuals, while reflection helps people to remember events, refresh friendship relations and is commonly desired by people.

3 EverSync Application

One of the contributions of this thesis to the PIM research field is the development of an ubiqutious PIM application which is called EverSync. EverSync is developed with the intention to solve several issues with the current PIM systems. In what follows, requirements for the EverSync application will be discussed along with functionalities the application will have and a description of the architecture.

3.1 Requirements

3.1.1 User Requirements

From the user perspective, EverSync has to implement the following required characteristics and features. First of all, multiple fragmented pieces of personal information space have to become accessible with easy re-finding possibilities. By using multiple devices and multiple (cloud) services, one's personal information gets scattered across all those services and devices. In terms of PIM, different devices and cloud services where personal information is stored can be associated with different contexts. Hereby, we have already discussed in section 2.3 that most cloud services make their own local copies of documents and therefore one particular document can occur multiple times in one's personal information space. Hence, a document can be available in different contexts without any overview or possibility to track where it is stored. EverSync application has to solve this issue through linking all fragmented copies of duplicated document files. Not only linking the duplicates should be possible, but also linkage of the related documents. For example, given

a files, the user want to know which other files are used together with it in one of the cloud services. Hence, EverSync has to track all devices and services in one's personal information space, create links, and has to do it automatically. Hereby, Victoria & Edwards state that user considerations and the degree of involvement in automatic systems are crucial to make the system's behaviour predictable [7] for the user. They discuss the challenges and importance of finding the right level of user control and involvement. So, whenever a link cannot be created automatically by EverSync, there should be a possibility to fall back on user's input. For example, in order to guarantee the correctness of the automated process, EverSync has to require user intervention for linkage of which loose any original metadata, which is the case with Facebook service. Once the user upload a photo, none of the original metadata of names or any other "hooks" are left by the Facebook which makes the automatic detection of the duplicate impossible unless we perform image recognition. This intervention not only assures the coverage of more use cases which otherwise would be unsupported, but also gives the users more sense of control and provides linkage correctness.
The EverSync application should also provide not only an intuitive user interface, but an interface which is familiar to the most computer users. Moreover, the interface should make discovering of the information space more easy, which will improve information re-finding as well. Hereby, the user should be able to reflect on the content. In other words, it should be possible to triggers user's memorised mnemonics links while discovering the information space.

3.1.2 Technical Requirements

From the technical point of view, EverSync should be extensible by plugins. Plugins are integration units, pieces of glue between EverSync and an external (cloud) service. EverSync should be flexible enough to let the plugins decide themselves how data from that plugin should be represented. Visual representation in the user interface, but also the internal entities representation should be decidable by the plugin. Besides the plugins, EverSync should support multiple devices and aggregate information from both, user's devices and cloud services. Aggregation and document linkage should happen on the fly and in an automatic way. Note that supporting multiple devices implies that also, that EverSync preferably should be a cross-platform application.

3.2 Iterative User Interface Design

Despite the described variety of ways to store and retrieve information (i.e. filing, piling and mixing), people tend to associate a computer with the *filing* approach [9]. This can be explained by the fact that mainly all operating systems only provide access to files and documents through folder navigation. As Bergman et al. describe [9], people are more familiar with this approach on an electronic device,

people find it more convenient to have folders on a computer and navigate through them. The authors also state in their work that computer users almost do not use the embedded search engine of the operating system. Therefore, folder navigation is one of the key requirements for the EverSync PIM application. EverSync has to provide a hierarchical view of folders and documents since it is the most convenient way for computer users.

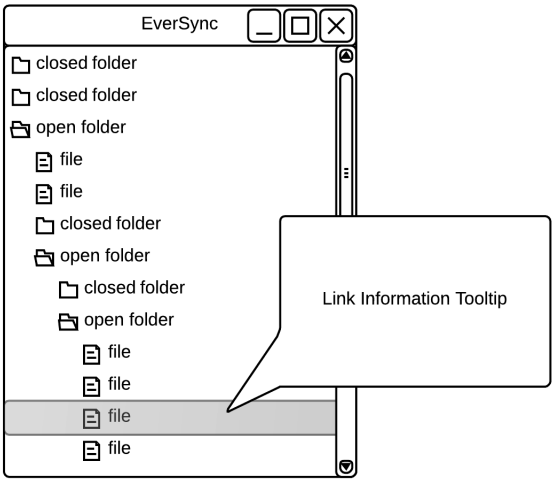


Figure 3.1: Prototype with single window and tooltips

In order to fulfil requirements from the previous section, an iterative design process has been used and different prototypes of the user interface were developed. The user interface for the EverSync application has been developed in an iterative way. Evaluation of each intermediary sketch during the iteration steps, eventually made it easier to implement the final and most applicable interface.
The very first design choice was a single window with a hierarchical tree structure of all documents on a device, which in fact is very similar to the standard "Explorer" in Windows operating system or the "Finder" from Macintosh. This very first solution was suggested by the fact that on most devices, most operating systems by default provide only the possibility to organise document in a hierarchical folder structure. The only difference with the standard file explorer is that EverSync should provide tooltips with additional information about documents. Those tooltips would contain information about where the selected document is used, to what it is linked and on which device it is stored, or any other plugin specific information. Figure 4.1 depicts the first prototype. For example, in case if EverSync had a plugin for Facebook installed, when hovering over a local photo in the file tree, the tooltip would appear and show that the photo was used in Facebook, in which album, which comments it had, and so on.

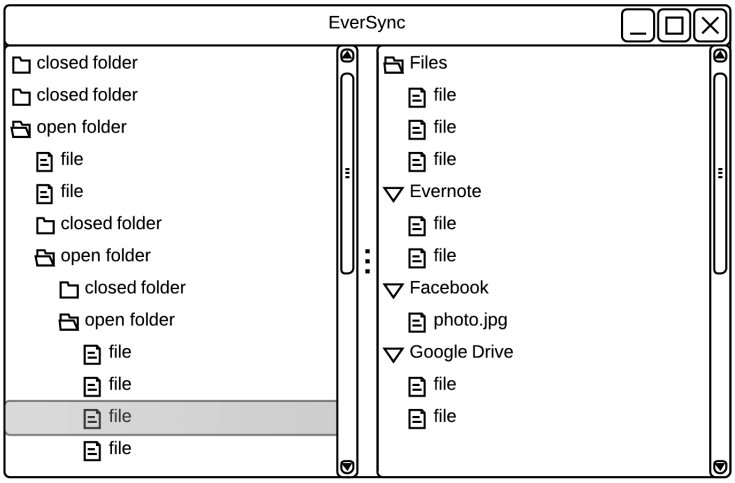


Figure 3.2: Prototype with two windows

Such a straight-forward design is very simple and intuitive for the user computer users are familiar with file explorers. However, this approach does not provide any sophisticated interaction with the user except for hovering over a file in order to get the tooltip containing information about that file. Moreover, hovering over implies that the cursor has to be kept on the file in order to read the file. This can be problematic if the user has multiple applications open and implies even more problems on mobile devices with a touch screen which do not have a cursor at all. Furthermore, considering that this tooltip should be used for all types of linked information, and thus contain information provided by multiple plugins and EverSync clients, it would suffer from overload and become inappropriately oversized.
Therefore, a second prototype has been developed which can be found in Figure 4.2. In this case, the interface divides the application window vertically into two parts with in fact two hierarchical file trees. The idea was to have in the left part all the files on the local device (i.e. where the EverSync client application is running) and the other connected devices which have EverSync client installed. The right part should be initially empty on startup of the client application. Having files from local device and from other user's devices together in one file tree at the left, should blur the boundaries between all user's devices. By selecting a document in the left window part, the user should get in the right sub-window a list of linked documents by plugins. This should give the information in which cloud services the document is used, where it is duplicated. In fact, this right window part is nothing more than a replacement for the tooltip so that there is no need for hovering over. This second design iteration solves the problem of potential oversizing and overloading of the tooltip. The problem with that approach is that the right window part itself limits the interaction with the content to one single layer of links. This leaded to

a re-evaluation of the designed prototype and the next design iteration.
So far, it was only possible to look up a document in the left file tree and on the right get a list of documents which are linked to that file. For example, selecting a photo in the local file tree, and getting a list in which cloud services it is used. Hereby, because plugins for cloud services are free to define their own hierarchies which will be displayed, it can be the following case. This selected photo is used in a note in the Evernote service, and the plugin for Evernote which is installed in the EverSync, shows three level hierarchies of the following type: {notebook; note; note content}. So, on the right sub-window, the result should be a hierarchy of the notebooks which contain the file, respectively dependent notes of those notebooks where the selected file is used, and then finally, all the files per note. By selecting *file.jpg* left, the user gets on the right among others the Evernote with the list of notes. Selecting one of these notes will show the collapsed content of that note, which yields the list of notes of inside that note which have the desired file. Then again, user can select on one of those notes to see all the used files in it, not only the one was originally selected in the left sub-window. Of course, as already mentioned several times, the plugins are free to create their own display hierarchies if needed, which makes this example even more a valid use case. This way, user would be able not only to see in which services the originally selected file is used, but also which related files it has inside a note in Evernote. And it would be desirable to be able to select one of the related documents in one of the notes in the right sub-window, and explore the interrelations of the information space.

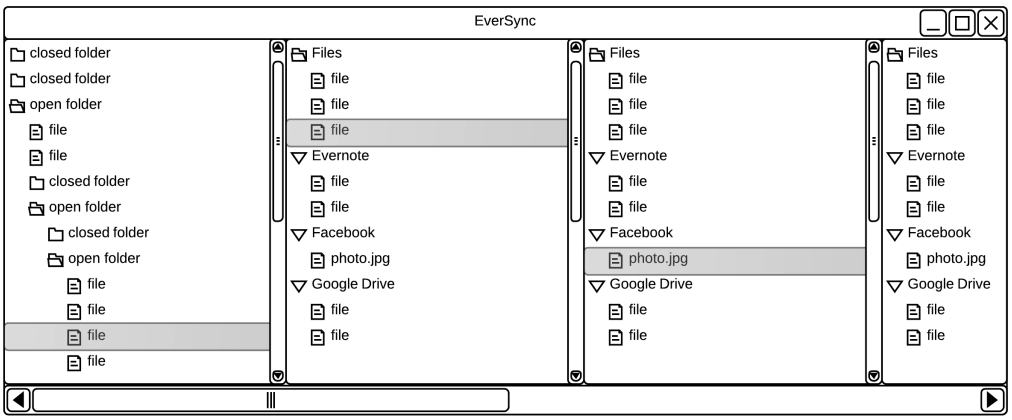


Figure 3.3: Prototype with sliding windows

The designed solution was, that whenever a user selects a document in the right sub-window, the client interface would create an additional sub-window at the right, applying the same logic of related files. This endless window creation would make

it possible to be able to endlessly following links and discover related items and documents. Of course, if the user would select some other file not in the far most right sub-window, all the windows at the right of it would become irrelevant and would be pruned. This design approach is called a *sliding window* approach. This prototype creates additional dependent file trees on demand, as can be seen in Figure 4.3. Selecting a document in the first file tree, will result in a list of linked files in the second file tree. Selecting a document on the second file tree will trigger creation of a third file tree with linked files and so on. If the user reselects a file in one of the previous trees, the subsequent file trees will be replaced by one single tree with files linked to the newly selected file. This flexible approach offers an intuitive and clear overview of link dependencies. The browse path through linked files is persistent and can be tracked back through the file trees. Moreover, this approach is easily supported by mobile devices. However, the evaluation of this design choice showed that his endless approach has its drawbacks. By drilling further more to the right, user is able to see related documents, but completely looses the context to the local file system of the device, where are those related documents located locally. If the user is interested in getting a list of documents which are linked to one of the files from the right window part, the user has to remember location of that file and restarting the whole process by looking up that document in the left window part. In other words, the user can get a list of linked files of only one document at a time. To solve that issue, the next prototype has been enhanced with dynamic content selection of the window parts. This approach can be seen as a combination of the previous two. The improved mechanism implies selection of a document in one of the window parts (i.e. one of the file trees) which results in rendering the linked files in the other part of the window. Hence, selection of one of the documents from the left sub-window will trigger a rendition of the file tree in the right sub-window. There, the file tree contains the first level of the related documents which are provided by the plugins. Selecting one of the items at the right, dependently if the plugin has more levels of hierarchies, EverSync will create a new sub-window for each hierarchy. Up until the point, where the user selects a file which is also stored on the currently used device. If that is the case, then the selection happens of that file happens in the very left sub-window which contains only the local files, and all the right sub-windows get pruned. Selection of the related document in the right, will make the selection "jump" to the local file. This way, the EverSync client interface implements the flexible endless sliding window and still preserves the context of the local files making it a ubiquitous interface.

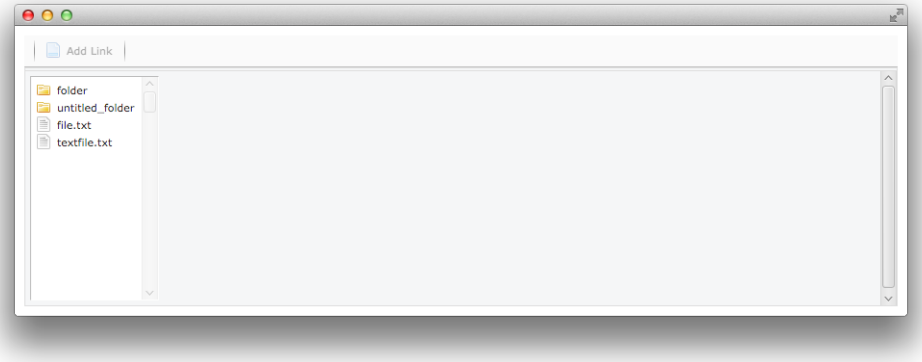


Figure 3.4: Initial application window

The user interface of EverSync is a direct realisation of previously described interface prototypes. This section demonstrates the working principle of the final implemented user interface. Figure 4.4 shows the initial window when the application client starts up. It contains only one vertical column (i.e. file tree) with local files, which are files on a particular device.

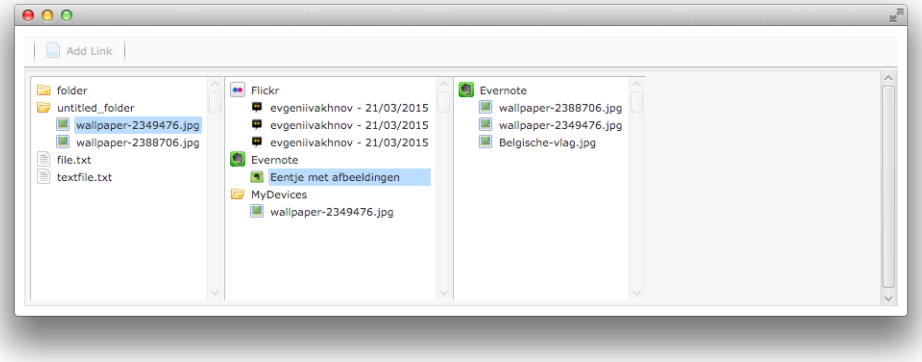


Figure 3.5: Browsing with sliding windows to third level

In fact, this window is just like a common file browser of the file system. Clicking and selecting a file from that list will send a request to the server for linked documents. The server collects all linked items and sends them back to the client. This results in an additional vertical columns, but now only with items linked to the selected one. The linked documents are separated based on the cloud service they

are from and represented in drop-down folder. Different cloud services work with different types of documents, therefore those items cannot be only text documents. Those can be multimedia documents or even comments in case of Facebook for example. Once the new column with linked items is rendered, the user can again select a file from the those results. Figure 4.5 illustrates this process to the third level. However, those steps can be repeated many times and the only limitation are memory and computational capabilities of the device.

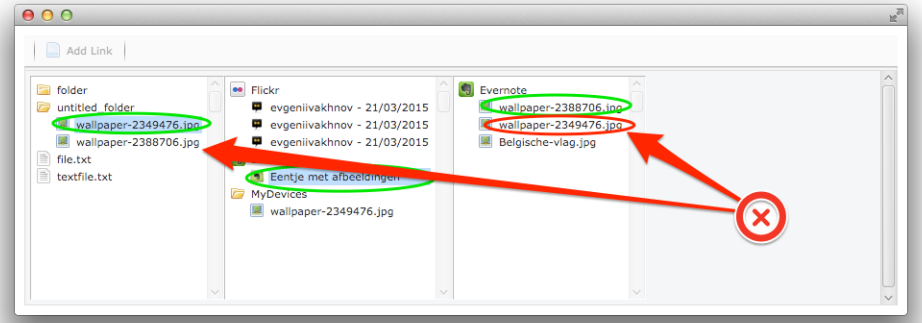


Figure 3.6: Reselecting another file, indicated with red cross.

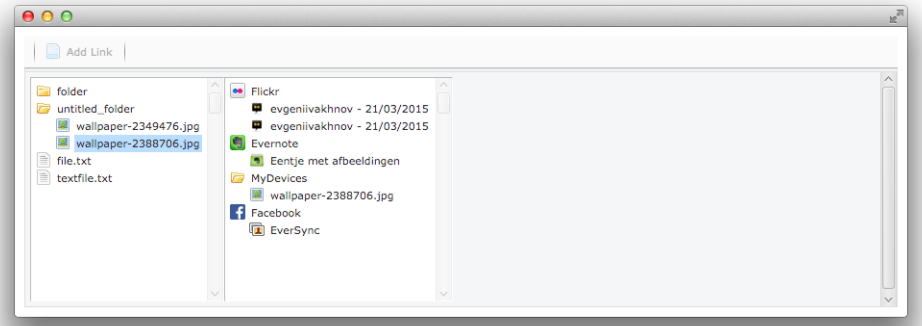


Figure 3.7: Result of reselecting a file

Of course, reselecting a file from one of the columns will influence the consequent file columns. The ones that are no longer relevant will be deleted, which is illustrated in ?? where the "reselection" happens in the very first column with files.

Which is intuitively logical since the newly selected item has potentially nothing in common with the previously rendered documents. Note that everything happens automatically and the application window becomes very interactive. The result of the reselection is depicted in Figure 4.6. The implemented user interface of the EverSync completely satisfies the described requirements. The application requires from the user a certain level of intervention, which gives the user sense of control. The application implements a standard way of representation of file hierarchies, which is important for new users since they do not have to adopt themselves to the application. And thirdly, it provides a great overview of the linked files. This makes the application easy and convenient to use.

3.3 Conclusion

The chapter started with the discussion of the main requirements for the EverSync application. From the user requirements perspective, first of all the application has to support the re-finding process. It has to make it possible to discover one's information space fragmented across multiple devices and cloud service. However, one's information space can consist not only of documents and files on a desktop or mobile devices, but also of other *entities* such as in comments, events, etc. Some of those entities can stimulate reflection, such as for example, photos from last holiday. In fact, people access their information space in order to reflect on things. Therefore EverSync has to ease this process as well and make it more ubiquitous, so that people to not have to perform additional steps but can reflect while exploring the their information space. Technically, EverSync should be able to detect different instances of the same file in different isolated parts of one's personal information space, such a device or a service and store them in a unified repository to foresee persistency. Those file duplicates should be synchronised. This means that EverSync should support multi-device usage and in order to be able to detect files in a third party service, EverSync should be extensible with plugins for services. All this should be accessible through a user interface where most users are familiar with, and which provides ubiquiitousity. This interface has been designed in several iterations and implements a sliding window mechanism. Selecting a local file will open a tree of places and services where this file has been used. Moreover, plugins will show which information inside the service where the plugins interacts with, is related to the selected file.

4

EverSync in Depth

In this section the implementation of the EverSync application will be explained together with a brief discussion of the Object-Concept-Context framework which lies at the base of the EverSync application. Then, the server, client and their communication will be reviewed. Some interesting implementation details will be discussed, together with some challenges and their solutions. The section will end with explanation of how plugins are used to extend the application to let it support different cloud services. But firstly, an explanation will be given of how the user interface has been designed.

4.1 Architecture

In this section, the main concepts of the EverSync overall architecture are introduced and explained. The differences between cloud services, plugins and client is described. Also, the main components of application are discussed, followed by the explanation of how they communicate.

In comparison to the most existing PIM systems which are mostly database oriented and focused on one's local files, EverSync application also targets digital information spaces situated in third party services. While most PIM systems act like individual databases with personal files, people also make use of the (online) services which create an additional information space. For example Flickr allows people to upload and store photo's. Clearly those photo's were uploaded by the user from a personal device and so it can be said that Flickr in a certain way

extends one's person information space. However, Flickr extends this information space not only by acting as an online photo storage place, it makes it possible to add an additional dimension to one's information space by letting users to make comments for example. For example Facebook goes further in this direction and provides features such as the famous 'Like' and 'Share' functionalities. From now, lets introduce a new term **asset**, which stands for an item from one's personal information space which is not by definition a file. As has been made clear in the example, both, a photo and a comment upon it, are related items and are equal parts of a personal information space. The photo itself was placed by the user into the third party service, it was uploaded, it is representable in the users devices since it originates from it, and it has a file extension after all. While the comment upon it is created by some other user in the service itself and is not something that has been transferred. Of course, if needed, a comment can be locally represented in the form of a text document, but this approach does not hold for all items created in third party services. Think of the 'like' from Facebook service which is certainly not text. Therefore, from now on all the items from a personal information space will be called more generally **assets**. It is the goal of EverSync application, to consolidate not only fragments of personal information space from different devices, but also to include information stored and created by third party services, in other words, all user's assets. All those described services can be integrated into the EverSync application through plugins. It can be seen as an integration tool of a third party service to into the EverSync application.

EverSync application has to support multiple user's devices. This requirement implies that the application has to support multiple clients. The decision has been made to implement a client-server architecture. The server is the control and storage point for the data. It manager clients and cloud services integration. The plugins periodically check the third party service for updates such as file modifications, creations and deletes. It it the plugins responsibility for keeping the content up to date. Plugin installation only happens on the server side of the application. However, when adding a new client, the server will push some information about the installed plugins for better support. An example of such information is an icon which will be rendered when representing linked files from the corresponding cloud service. This information push to new clients happens automatically and does not require any intervention of the user. While the plugin is responsible for connection with the third party cloud service, clients on their turn are only connected to the EverSync server via a TCP/IP socket connection. The server distinguishes between new clients (i.e. clients which are connecting for the first time ever) and already known clients. Hence, new clients have to go through an initialisation process whereby the server assigns a unique id number and pushes plugin information. Once a client is added to the network, it becomes part of the personal information space and indirectly connects to other devices, which is illustrated with a dotted line between clients in Figure 3.1.

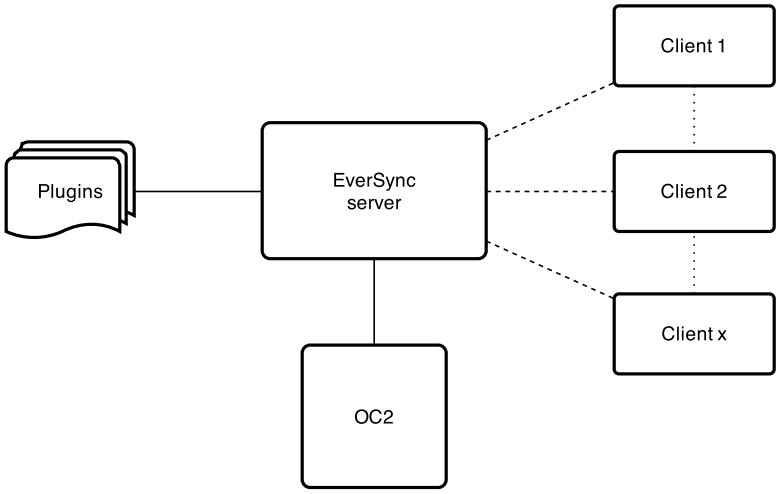


Figure 4.1: Schematic representation of EverSync architecture

The overall client-server architecture of EverSync is illustrated in Figure 3.1. The server of EverSync is the main core of the application and in fact serves as a central communication point with the Object-Concept-Context (OC2) framework. In fact, the whole server can be separated in three main components: (1) interaction layer with the OC2, (2) layer connecting server and the client and (3) logic to communicate with plugins. The OC2 framework lies at the base of the application and is discussed in next section of this chapter. Briefly, OC2 is responsible for creation and persistence of links between different resources. Those resources are different types of documents and files used and stored in multiple cloud services and user's devices. Those assets are stored in the OC2 as *Digital Object* and linked together through *Navigational Links*. For the connected clients, the server will be in charge to create links between related files, but for the content from cloud service, its the responsibility of the plugins. Moreover, the EverSync application is flexible enough to let the plugins create their own structures using the navigational links. This way, the Flickr plugin for example, uses only one level of hierarchy. The Flickr plugin is in charge to create links between user's photo's and comments upon those photos. So that when the user selects a photo, from the OC2 the related digital objects are retrieved, which will be links to comments. There is only one level defined by the Flickr plugin: {photo - comment}. But for example Evernote can have three levels: {notebook - note - file inside the note}. All those hierarchies are flexible enough to be plugin specific.

4.2 Object-Concept-Context Framework

As stated by Trullemans [52], there are three classifications of PIM systems. The first type of PIM systems are developed to track changes in the personal information space such as addition, deletion or updates of the documents. Therefore, the re-finding mechanisms of these systems are also based on the tracked changes. The second type of PIM systems are mainly focused on associating items by providing possibilities to link different types of documents. The created links however, are not always easy accessible of re-findable since such applications give less attention to interaction functionality. Finally, the third type of PIM systems, in contrast to the second one, is mainly focused on user interaction. By implementing an extensive user interface, such applications extend the set of possible user interactions in order to provide an easier process of information re-finding. However, Trullemans [52] also concludes that the problem with the most PIM systems is that the developers tend to focus either on the *organisation* of files at low level, or on the *interaction*. Where in the first case there is almost no interaction possible with the information space and in the second case the functionality for linkage is hardcoded al low level. This has lead to development of the Object-Concept-Context conceptual framework [53] which provides the possibility to combine the advantages of the two designs. In EverSync, the OC2 is used to link documents in different cloud services and on different devices.

In this paragraph, the OC2 framework will be described. At the base of the OC2 lies the Resource-Selector-Link (RSL) meta-model for hypermedia systems which was developed by Signer and Norrie [47]. The Figure 3.2 depicts this OC2 framework, and illustrates that **Entities** collection contains sub-collections of **Selectors**, **Resources** and **Links**. Hereby, **Resources** represent information pieces which will be linked, such as digital and physical documents, multimedia items, web pages etc. **Selectors** are intended for linkage of certain parts of a resource instead of linking the whole document. The **Links** collection contains bidirectional many-to-many links. Note that every instance of the **Entity** sub-collection can be linked with another one. As can be seen in Figure 3.2, the OC2 framework the **Resources** collection is extended with **Concepts** and **Objects**, which in their turn can **Physical Objects** or **Digital Objects**. A resource can belong to only one of these three sub-collections which means that a digital object cannot be physical or vice versa and an object is not a concept. Hereby, **Concepts** are items internal to the memory, while the **Objects** are external. Moving on to the **Links** collection, we will see that the OC2 distinguishes between the **Extent Links** for linking objects with concepts and **Associative Links** which are used as navigational links. Finally, there are two pairs **hasExtSource** & **hasExtTarget** and **hasAssocSource** & **hasAssocTarget** for defining the source and the target of a link.

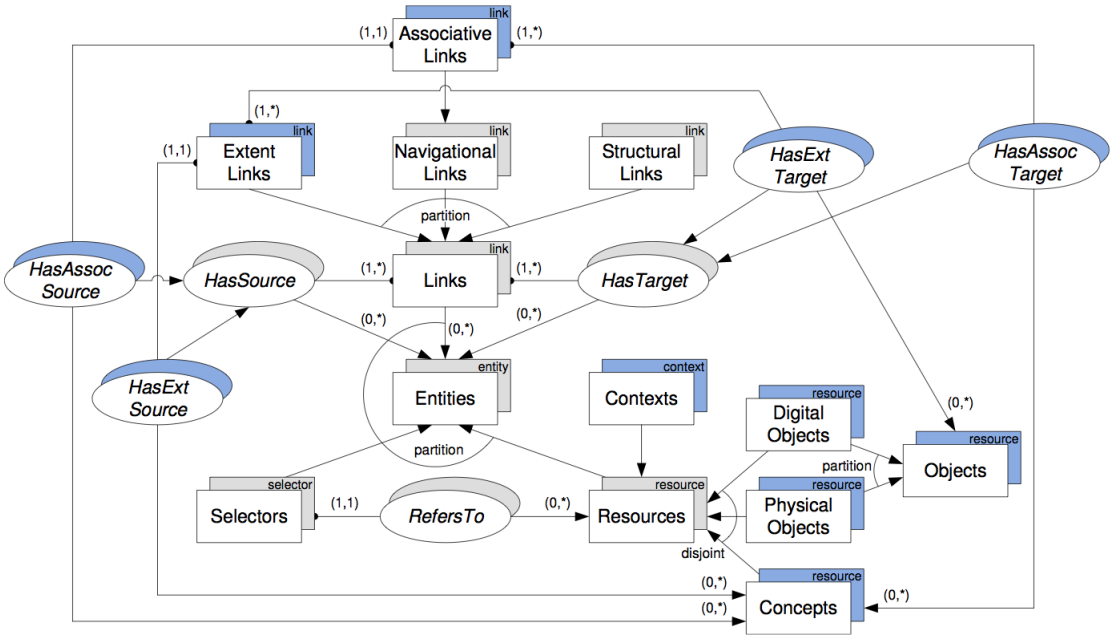


Figure 4.2: OC2 framework as domain-specific RSL application as cited in [53]

4.3 Server-side

The server-side of EverSync is implemented in Java programming language. In fact, the whole server code can be separated in three logical components which are: (1) code responsible for the interaction with the OC2 framework, (2) connection layer and (3) implementation of the logic for plugin support.

The server "listens" to the socket for the incoming connections. Detecting an incoming connection will start the handshake process. After that, for each new established connection, the server creates a new thread. Note that in case of a network interruption or client shutdown, the server detects the disconnection and stops the thread. Hence, each client has its own thread on the server where the exchanged messages are read and interpreted. While the **Server.java** creates the threads, the **MessageReflect.java** class is responsible for the interpretation of the massages. As the class name already suggests, the messages are reflected to server functions. Usually, those functions are variations of the CRUD (create, read, update and delete) methods for files and their linked items. Message which are reflected to that kind of methods are passed through an additional layer of abstraction, the **FileEventHandler.java**. Those file events are forwarded to the OC2 framework which is responsible for storage of links between documents.

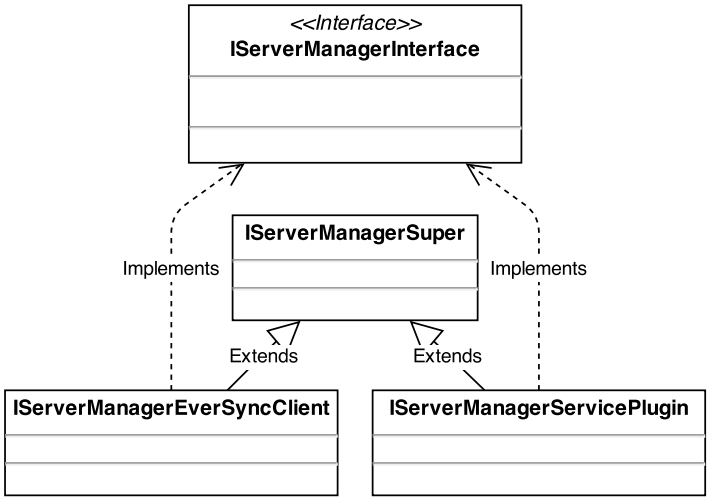


Figure 4.3: Schematic representation of the IServerManager implementations

The interaction with the OC2 framework with the server is one of the crucial points of the EverSync application. The implementation of the **IServerManager** is responsible for the actual functionality of interaction. Figure 4.8 illustrates schematically how the IServerManager is implemented. As can be seen, there are actually two types of the IServerManager. This can be explained by the fact, that there two types of documents. The documents that are actually stored on one of the devices and (copies) documents that are stored in one of the cloud services. The **IServerManagerEverSyncClient** is responsible for storage of resource instances representing the documents from one of the clients. The **IServerManagerServicePlugin** consequently contains functionality to manage documents from the cloud services. In fact, most of the functionality of those two manager types overlaps. Therefore, their intersection is implemented in the **IServerManagerSuper**. The **IServerManagerInterface** guarantees that the two manager types implement the distinct functionality that is required.

4.4 Client-side

The client side of the application is mostly implemented in JavaScript and partially in Java. The main reason for that is the pursuit of the high portability of the application. The whole implementation process of the client has been started with the TideSDK framework. This solution offers the possibility to develop a desktop application using web technologies such as JavaScript, HTML, CSS, etc. and "bake" it in a Chrome web browser. In fact, the compiled standalone application is a browser with the webpages, which represent the actual application with its logic and inter-

face. However, the TideSDK has some limitations in functionality for interaction with the actual device on which the client application is running. For example, EverSync has to be able to detect file addition and deletion. TideSDK offers an API for interaction with the file system, but a "watcher" for the file system is not provided. Moreover, an SQLite database is almost not supported by the JavaScript world. After encountering those two and several other problems a decision has been made to redirect the implementation and use an alternative for TideSDK. Instead JavaFX has been chosen, which is basically a concurrent of TideSDK. This solution offers the same possibilities and even more, JavaFX can integrate Java code into the "web app" which solves lots of problems. Hereby, JavaFX offers more convenient ways for interaction with underlying operating system. Since the whole client code was implemented in JavaScript as a web application, migration went fluently. On one hand, implementing some parts of the program in Java and others in JavaScript brings some distortion in the project coherence. On the other hand, it makes it modular and portable since the implementation of the core logic remains in JavaScript, and the modules for interaction with the operating system are in Java. Using solely Java for implementation of the whole client side is not an option since it wouldn't produce an application supported by all devices and operating systems (e.g. iPhone). Moreover, the web approach for implementation of an application offers a wide range of powerful libraries and tools (e.g. jQuery). Furthermore, JavaScript is easy debuggable at runtime via the webbrowser console. For the interface, some jQuery plugins were used such as **w2ui.js** and **jQueryFileTree.js**. The latter one is in fact the workhorse of the client since the main interaction points with the user are the file trees.

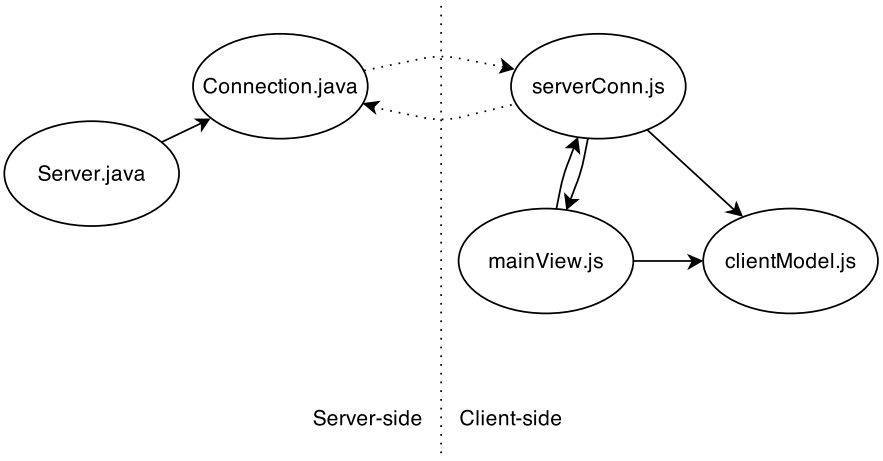


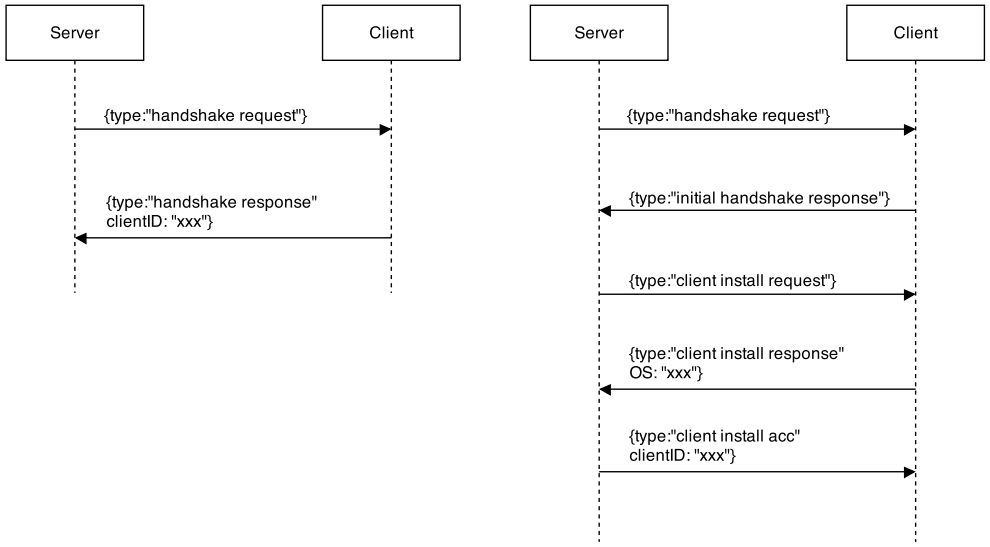
Figure 4.4: The MVC architecture of the client

The main client architecture follows a MVC pattern, which is depicted in Figure 4.9, where the **clientModel.js** is the model, the **mainView.js** is the view and the **serverConn.js** is the controller. As explained earlier, by selecting a document in the file tree, the user gets another file tree with linked items. In fact, the **mainView.js** indirectly sends requests to the server via the **serverConn.js**. The latter one is the main communication point, receives the responses and is in fact the controller of the whole client.

4.5 Client-server Communication

With modularity and portability in mind, the server and client architecture of the EverSync application have been developed in distinct programming languages. While together they form one application, they are completely independent and communicate with each other via TCP sockets. The reasons for such implementation approach will be discussed. Some encountered problems will be explained and their solutions will be evaluated.

The information exchange between the server and clients happens through a TCP/IP socket connection. Sockets are advantageous when messages come from both directions, client and server. As with most client-server applications, in EverSync usually the client requests some information and gets a response. However this is not always the case. For example, if the server cannot automatically detect a link between two document instances, it will prompt the user to define the link manually via one of the clients. Hence, when the user sets a link between two documents via one of the clients, then the server will broadcast that those documents are already linked, which will delete the prompt windows in other clients. This broadcast message is initiated by the server and all clients will receive it at the same time, without poking the server. Hence, the communication between clients and the server is bidirectional in EverSync. TCP/IP sockets also make it possible to have a connection with very little overhead. The WebSockets alternative solution has been rejected since the WebSockets are actually built on top of the normal TCP/IP socket. Hence, the overhead of WebSockets can be disadvantageous when it comes to the performance in mobile networks. More overhead usually also means higher cost for the user for the sent data, since mobile internet is often more expensive in terms of energy, money, etc. than normal Internet connection on a desktop. The messages that are actually sent, are formed in the connection layer on top of the plain sockets. The messages for information exchange are JSON's. Each message has a type and parameters, if there are any. The module for interpretation of messages on the server side is called **MessageReflect.java** because most messages received from a client are requests associated with a function or method. Hence, the messages from clients are reflected to server methods. The alternative on the client side is called **ServerConn.js**, which is actually the controller in the MVC pattern of the client.



(a) Already initialized client (b) New client

Figure 4.5: Messages exchange during a handshake with two types of clients

Any client-server communication is only possible after a handshake. Note that the very first message of the handshake process is sent by the server. This is due the fact that the server has to receive an incoming connection before starting to respond it. Hence, connecting to the server will trigger the handshake process initiated by the server which detects the incoming connection. There are two types of handshake, the one with a client which connects for the first time ever (i.e. new client) and a regular handshake. Both handshake types and the corresponding message exchange is illustrated in Figure 4.7. A new client has to be "installed", meaning that it has to receive a unique id number and become identifiable by the server. During the installation step, also the application root folder will be assigned to the client. Different operating systems have different system folder hierarchies, consequently each operating system has its own root folder for the application. Also the plugin information such as the service icons are pushed to the client. Note that those messages are not illustrated in Figure 4.7b.

4.6 EverSync Plugin Implementation

In order to support a third party cloud service, the EverSync application requires a plugin for that service. Of course, the EverSync application with its client-server architecture fully synchronises multiple devices (i.e. clients) but in order to be able to integrate a third party service, a plugin for the service has to be implemented. And so, plugins form a crucial part of the application since none

of the services is supported by default. A plugin in fact implements the facade pattern for the program that interacts with a cloud service. Plugins have access to certain methods in order to foresee persistence, creation of links between documents and the possibility to create a plugin-specific structure, and on the other hand, the plugin itself also has to implement a number of methods for the EverSync base application. For example, the base application periodically loops through all installed plugins and calls the **pollForChanges()** method. Also, each plugin has to have a **init()** and **run()** methods implemented. Clearly, a plugin implementation for a certain service is just an implementation of an interface which can be inspected in Listing 4.1. This gives a clear overview of what has to be implemented.

Listing 4.1: Plugin Implementation Interface

```
public interface PluginInterface {

 public String getPluginName();

 public void init(FileEventHandler fileEventHandler);

 public void run();

 public HashMap getInstallationFiles() throws Exception;

 public void replaceFile(String fileName, String fileUri, byte[]
     fileByteArray);

 public void pollForChanges();

 public void handleOpenOnClientRequest(EverSyncClient client, String uri);
}
```

Clearly, a plugin has to have a name which will be visible for the user. It is up to the programmer of the plugin to decide which name will be used, but most likely this name will be identical to the name of the service for which the plugin is implemented. In context of this thesis, three plugins for three different services were implemented: Facebook, Flickr and Evernote. The second listed method in the interface is the one to initialise the plugin. A plugin has to have access to the base EverSync application via the file event handler in order to report file changes. This method can also be used for any initialisations needed, such as connection to the third party service, creation of a internal database etc. Thirdly, there is also a 'run' method. This method is called whenever the whole EverSync server has been initialised, the plugins have been initialised and everything is ready to run. Mostly this method will be leaved empty, but this gives the possibility to execute any custom code if needed. The fourth method listed in the interface is the one to get the installation files. The actual plugins for the third party services are installed server side, however when a new EverSync client connects to the server, it will receive installation files for the plugins installed on the server. Those installation files are basically icons and a CSS file in order to be able visualise properly files or

any other assets internal to the service. For example the implemented plugin for Evernote, besides the personal files also tracks notes in which those files occur, and has therefore an icon in the installation files to visually represent those notes. The next and fifth listed method in the interface is used for replacing files (i.e. updating the file content). It is used whenever a file itself is updated or changed on one of the clients or services. Remember that EverSync application is not only implemented for an easy way of linking files, one of its goals is also to maintain a certain data integrity by keeping the linked files synchronised and up to date with each other. The data (i.e. files) persistency is by default built into the clients which means the file changes, updates, deletions and file creations are detected and sent to the server. As described there, the clients populate a small database with the correspondent creation and last update dates of files, which is necessary to ensure data persistency in case the client application (accidentally) goes down while the client device itself is still in use. So when the client application starts up, it first checks updated files based on this so called 'file system snapshot'. This approach is also enforced by the fact that the server-side of EverSync does not check or prevent data duplication. In other words, uploading the same file with the same properties from the same device is allowed by the server. Since this can cause discrepancies, EverSync client implementation is foreseen with the needed safe checks, and the same is required from a plugin implementation. It is the clients and plugins that have to decide which are new files, which files are updated and which are deleted. This approach considerably reduces load on the back-end server and therefore improves overall performance of the system as a whole. The sixth method in the interface is, as the name indicates, used for polling changes from the third party service. The EverSync periodically loops through all its installed plugins (using the *PluginManager*) and calls this method. Of course, it is not an obligation to implement this method. For example, if plugin designers decide to implement the actual connection with the third party service using sockets, then there is no need anymore in polling the service, hence there is no need to implement this method. In that case, the method can be left empty and the creator of the plugin can use its own approach to feed the data to the EverSync using the provided mechanisms. Finally, the last provided method in the interface is the *handleOpenOnClientRequest*. Besides providing the possibility provide an overview of the personal files as a list of assets along with their relations, the EverSync application makes it possible to actually interact with these files. Hence, listing and providing an overview of the files is not the one single use case of EverSync. Therefore, in order to open a file, or lets call it more general an **asset**, since the third party services not alway operate with files (for example, a comment on a Facebook photo is not a file), we cannot rely on the device and its installed applications to open it. EverSync relies to the plugin which represents the service from where the asset originates. When opening an asset via the context menu (by right-clicking on it), the EverSync application will make a distinction between the local assets which are files on the device(s), and assets represented by one of the plugins. The local files will be opened using

the default application associated by the operating system of the device with the respective file extension, and opening the 'remote' assets will result in a trigger of the described *handleOpenOnClientRequest* method. The plugin will get an object which represent the calling client, and a URI of the corresponding asset (which was assigned by the plugin itself). Currently, the client API provide the possibility to open a URL link in the default web browser and a byte array representing a file by the default application associated with a given file extension. This creates enough flexibility for the plugins to be able to handle such requests. For example in the implemented Facebook plugin, whenever the user tries to open a comment upon a photo, the plugin generates a web link to that comment, based on the given URI, and using the client API makes the client open it in the web browser. Of course, an alternative solution could be to create a temporary text file with the comment message in it and to let the client open it with the default text editor, which is perfectly possible since all the needed functionality for it exists. The behaviour is completely defined by the plugin designer, while the functional base exists.

4.7 Conclusion

The implemented EverSync application has been designed to satisfy the requirements from the previous chapter. Firstly, we discussed the architecture of the application. EverSync has a client-server architecture in order to support multiple devices. On each user's device, a client application has to be installed. The clients are implemented in JavaScript in order to make them cross-platform using the JavaFX. The server of EverSync is implemented in Java and can be considered as a unified repository with references and relations of user's documents and files. In order to store all those entities, which are called *digital objects* and the *navigational links* between them, the server uses the OC2 framework. EverSync can be extended with plugins for certain (cloud) services where user has personal files, in order to incorporate those files in the unified ubiquitous information space which EverSync tries to provide. A plugin is a communication interface for the third party service it is implemented for. Those plugins are installed on the server and if needed, the server pushes certain system files to the clients in order to properly visualise the plugin items in the interface. In order to support third party applications.

5

Proof of Concept – EverSync at Work

The personal information space can contain lots of different types of resources, document files are probably the most fundamental items. Therefore, in order to point up the actuality and usefulness of the EverSync, a plugin for the Evernote service has been developed. Evernote is a great solution to include different types of document files in personal information space, however multimedia files are also supported. Hereby, personal information space is not limited to only storage places for personal files and documents. Social media and social network can become a huge part of one's personal information space. Most popular example of social network is of course Facebook. Facebook can contain different types of personal information, including photo's and their comments. Note that the comments are actually added to someone's personal information space by other people. In this section, the practical relevance of the EverSync application will be illustrated. The description of the three implemented plugins for the application will be followed by an explanation of the clients usage.

5.1 Plugins for Cloud Services

5.1.1 Evernote plugin

The main idea of Evernote is to help people to remember everything. This means that the point of the application is to give the possibility to store notes, ideas, photo's, etc. in order to use it later, which is the main principle of a PIM application. As the name of the service denotes, Evernote is designed to make notes. On the first sight Evernote looks like a text editor, however it provides functionality

to add all kind of multimedia files to a note. Since Evernote also provides client applications for mobile devices, it makes it very convenient to add to a note photo's or audio notes recorded with the mobile phone. Actually, Evernote has three key features:

- **Synchronization**
  This is the main principle of Evernote. The user installs a client on his desktop and on his mobile device. From then on, notes created on one of the devices will also be synchronized with client on other devices.

- **Storing notes in different notebooks**
  On the first sight, all the notes in Evernote do not follow any order and are placed in a chaotic order. Nothing could be further from the truth. Notes can be stored in different notebooks and can be tagged (one note can have multiple tags). While searching a note, user can use this information.

- **Possiblity to add multimedia files**
  It is very convenient in Evernote to add different kind of media files to a note. This can be some audio recorded with a mobile phone of a photo or even a document (i.e. pdf, text, etc.).

Developing an Evernote plugin for the EverSync in fact does not mean that the EverSync will replace the Evernote client application. A plugin is intended to interact with the Evernote server in order to get information about existing notes, not to create notes through the EverSync application. This is important to understand, the notes still have to be created using the Evernote client. After having created a note using the Evernote client, this will be detected by the Evernote plugin of EverSync. As a matter of fact, the plugin is responsible for service polling in order to detect changes. As discussed in **??**, the plugin implementation as per plugin interface is required to implement the polling method. This method is called on each plugin every minute. Of course, is desired, the plugin cal have an internal polling loop which will poll with another frequency. Therefore, at most one minute after creation of a new note, the plugin will detect it. For that, the plugin falls back to the provided Evernote API. In this example of creation of a new note, the detection of changes triggers a whole process to add a new note and the files it contains to the EverSync and its underlying OC2 framework. The process consists of different steps. Because the Evernote API provides the possibility to request any changes (i.e. creations, deletes and modifications) of notes, the plugin stores only the timestamp of the last polling moment. Unless poll happens the very first time ever, the plugin makes a call to the Evernote API to get changes since the last synchronisation time. However, as an example for illustration, let us consider the initial poll. When started up, the plugin connects to the API and makes a request to get a collection of all the notebooks of the user. In Evernote, there are three levels of dependencies: notebooks, notes which are in those notebooks and files which are in those notes. Because of the flexibility of EverSync, each plugins

is allowed to define its own hierarchies, the Evernote plugin can make use of it. However, the implemented version for this thesis only uses two levels, namely *notes* and *files* in them. So, to continue with our initial polling process, the plugin loops through the collection of the user's notebooks and for each notebook it searches for notes with files. If a note contains files, the plugin requests some additional information for those files such as, file name and file size. The next step is to add all this information to the EverSync.

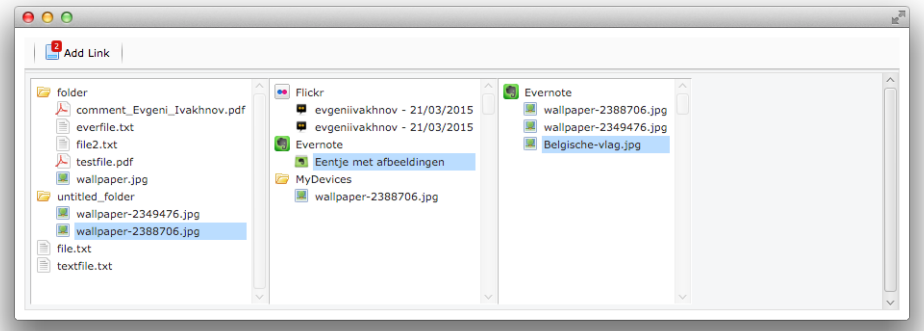


Figure 5.1: Browsing through Evernote content

The plugin asks the EverSync server to create a *digital object* for each note that contains files. Then, for each file inside this note, the plugin asks EverSync again to create a digital object for each file and link it to the note that just has been created. In this *navigational link*, the note can be seen as a parent and the file as a child. However, the plugin will ask the EverSync to create the digital object for the files using another server method, in order to notify EverSync that this is something that can contain duplicates on user's devices. This is needed because a note is something specific to Evernote, it is created and maintained inside Evernote, it cannot be something that a user uploaded to Evernote from one of the devices, while a file in a note actually originates from of the user's devices. So, because it was found on Evernote, most likely it will be duplicated in user's information space, on one of the devices. By using this special method, the EverSync will create a digital object in the underlying OC2, and search for digital objects from the devices which can be linked. The search happens based on the file name. So, the files with the same names will be considered by the EverSync as duplicates. When the duplicates are found, on that point the EverSync linking component has two digital objects per iteration to link: one representing the file on one of the devices, and the one that just has been created by the plugin. If the EverSync would link those two by a navigational link, the *note* should be left unused and not discoverable though the user interface. Therefore, the linking components creates

a link between the digital object representing the 'local' file from a user's device, and the so called **root taxonomy item** of the hierarchy created by the plugin, which in this case happens to be the note.

5.1.2 Flickr plugin

Flickr is a cloud service for storage, organisation and sharing of images and photos. However, it also has a certain social aspect since users can make comments. As with Facebook, comments become part of someone's personal information space. Conceptually, Flickr allows users to create albums with photo's which can have comments and stars (rank points). So basically, we could have again three level hierarchy: *album – photo – comment*. The Flickr plugin is implemented in implemented in a slightly different way than the Evernote plugin to illustrate the flexibility of EverSync. The implemented plugin only uses comments, which are linked by the EverSync to user's local photos. So, the plugin neglects the albums and considers only user's photo's stored in the cloud. Then, for each photo that has a comment, the plugin asks the base EverSync to create a digital object for it and immediately link it with all instances of that photo originating on user's devices.

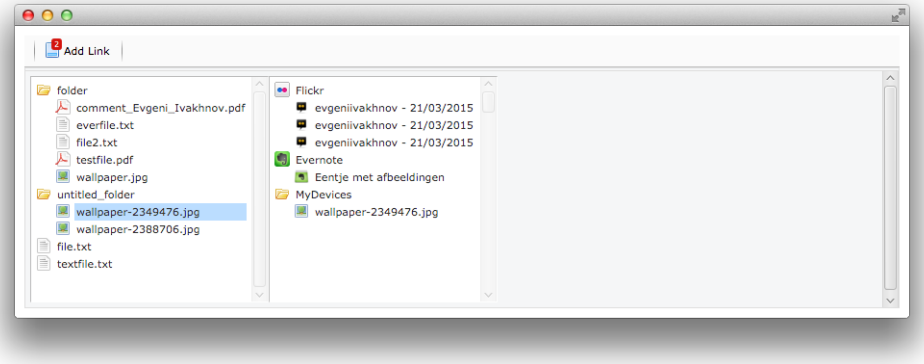


Figure 5.2: Viewing the Flickr comments upon an image

Again, the search and linkage of duplicates happens based on the file name. This is absolutely not the best solution for linkage, however, this is a limitation of EverSync and can be improved without any modifications to the plugins. Hence, the resulting structure build by the Flickr plugin has a one level hierarchy whereby the user can select a photo on one of his devices and then be able to see who commented this photo in Flickr. Again, by right clicking on the comment, user can open the comment in Flickr itself, hence by going to the page in the web browser.

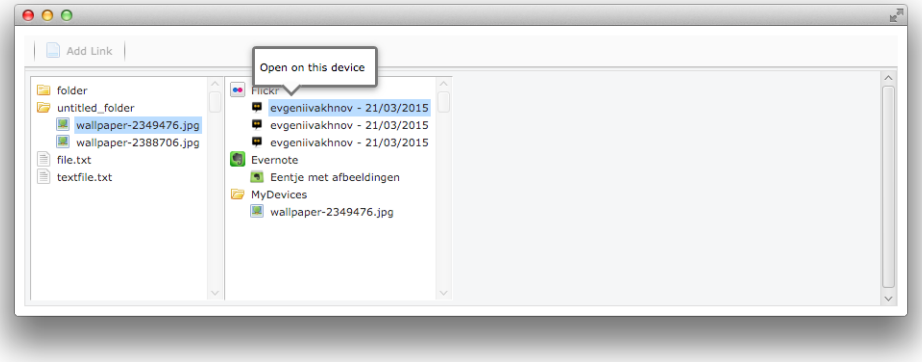


Figure 5.3: Right clikcing on the comment to view it in browser.

By right clicking on the comment, as with other plugins, it is possible to open the comment right in Flickr itself, as is illustrated in **??**.

5.1.3 Facebook plugin

Facebook is one of the most popular social network services and does not need much introduction. The last statistical reports state that in April 2015, Facebook counted 145,308,764 unique visitors. Facebook provides different kinds of social interaction and photo sharing is one of them. Users can upload and share photos while their friends can leave comments. Most likely those photo's are uploaded from a device and therefore are copies of local files. Facebook becomes part of users' personal information space and extends it with new items, comments in this case.

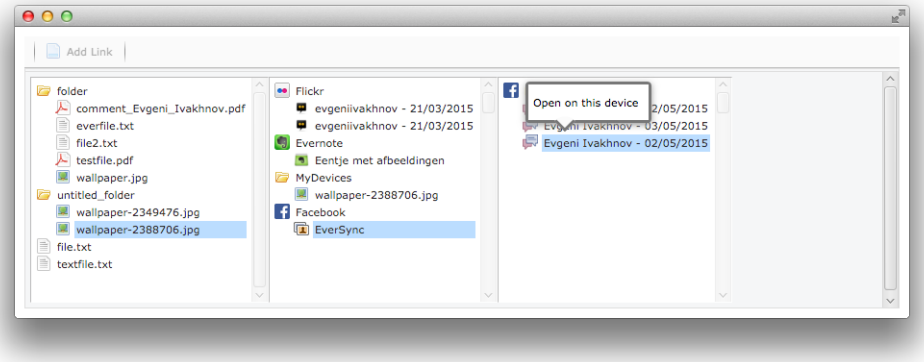


Figure 5.4: Browsing through Facebook comments.

In comparison to the Flickr plugin, this implementation of Facebook plugin for EverSync supports a two level hierarchy for comments and albums. This can be motivated as follows. The user has to select a local photo in order to see where and how it is used. So by selecting the photo in the 'local' file tree, there is no particular need to 'confirm' this selection. However, in Facebook this photo can be used in several albums. Therefore, when selecting a local photo, user will get a collection of albums on Facebook where this photo is used. By selecting one of these albums, the next hierarchy level are the comments upon that photo. So at the end, the user has the following hierarchy: *photo – album – comment*. This structure is realised in similar way as in the Evernote plugin. Plugin makes requests to the Facebook API to get all albums. For each photo in those albums, it asks the EverSync to create a digital object for this album. Then, for each comment upon a photo, it asks the EverSync to create a digital object as well and immediately calls the EverSync method to create a navigational link between the comment and the album of the photo. After all this, the EverSync method is called for detection of the local occurrences of the photo, and an automatic linkage of the photo to the *root taxonomy item* of the Facebook plugin hierarchy, which is the album. This is illustrated in **??**

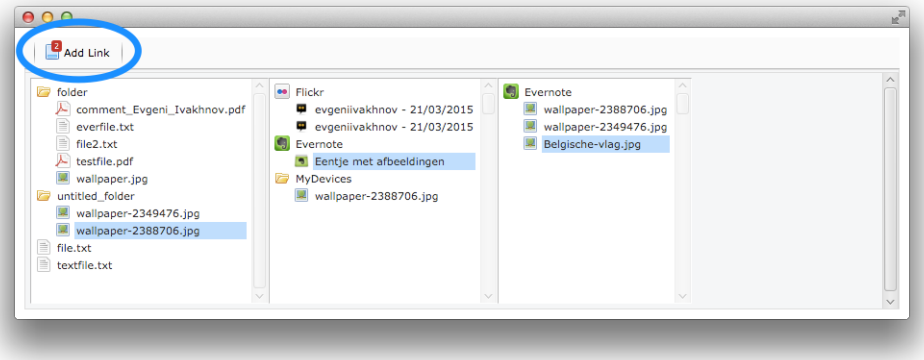


Figure 5.5: Link queue indicates two files to be linked by user.

In comparison to the other two plugins, Facebook plugin bumped into the EverSync limitation which had to be solved. The limitation consists of the already described problem of linkage mechanism which is based on the file names. Files on different cloud services and the local files with the same names are considered as different instances of the same file. The problem is that Facebook wipes all file information when it gets uploaded to Facebook. Hence, there is nothing left to rely on, no file name and no metadata in the EXIF fields. The solution for this problem is the so called **link queue**, which is indicated in **??**.

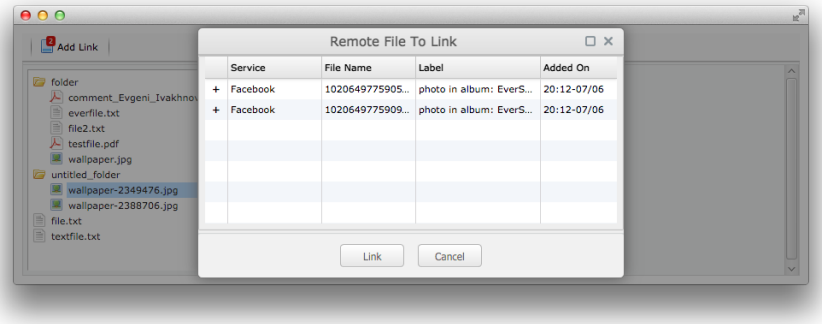


Figure 5.6: Initial pop-up of the link queue when the 'Add Link' button is clicked.

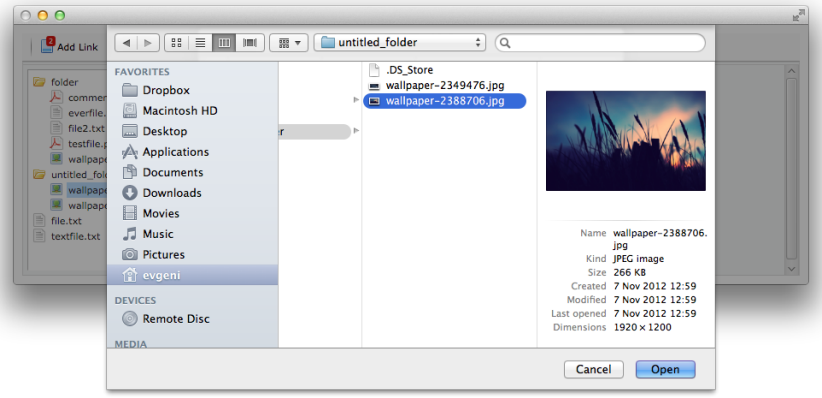


Figure 5.7: User selects a local file.

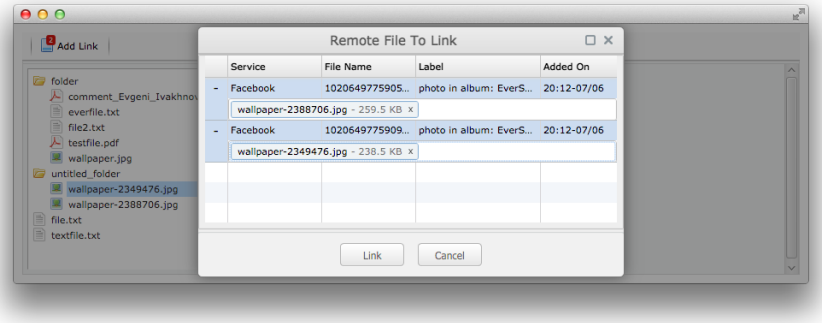


Figure 5.8: User has to confirm his selection by clicking the "Link" button.

Since Facebook assigns new names (Facebook uses id's instead of file names) to the uploaded photos which means that automatic file linkage with local original does not work. The EverSync user has to manually link Facebook photo with the local original file. Every file detected on Facebook is added to the link queue. This queue is synchronised across the EverSync clients. The number of currently pending items to link, is displayed on the button. When clicking on this button, user gets a pop-up with all the pending items, as illustrated in a sequence of screenshots, **??** where the user gets the popup, **??** where the user selects a local file to link with and **??** where the user has to confirm everything by clinking on the 'Link' button. Of course, the link queue is a generic solution and is not limited to the Facebook plugin. Therefore, the user gets a list of files and their locations. In there, user can select one of his local files to let the EverSync create the actual link between those items. While selecting a local file, the client can only access the files on that particular device. That is why the queue is synchronised across the devices. The user has actually to link the original file from one of the devices that have an instance of that file.

5.2 Multi-Client Integration

Now that the plugins are explained, let's focus how the EverSync behaves using multiple clients. In comparison to the technical perspective, from a conceptual point of view a client is a special kind of a plugin. While plugins are implemented to interact with third party services, clients are implemented to interact with user's devices. Hence, they do not implement the same interface and they don not polls to the services. Though, in the interface they are represented in the same way as any other plugin.

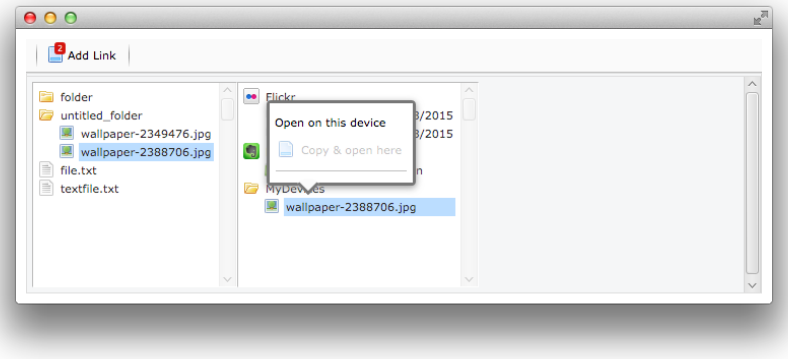


Figure 5.9: Context menu options for file on a single device.

A user select a local file in the most left file tree, which results in a rendition of an additional sub-windows with a file tree next to it. Along with all the plugins, there is a folder which is called **My Devices**. There, this folder mainly will contain only one single file. This indicates that the selected local file has also another instance on one of the users devices. It does not mean that there is only one single instance of that file. The interesting part comes when a user right clicks on that file in 'My Devices'. There, the user will be able to see on which other clients the file is replicated. The clients that are currently not connected to the EverSync server will be greyed out. For the ones that are available, the user can choose to open the file remotely. Of course, opening a file locally is also possible. This is also applicable for the cases when via a plugin, user sees that a local file is used together with other remote files (files on other devices) on the same location in the cloud service. As for example, in Evernote a file which is discoverable locally (on the client the user currently operates) is used in a note A. When clicking on that note, the user will be able to see other files inside that note. And some of them might be not replicated on the current devices. So clicking on such a file, will result in another additional sub-window being rendered with an indication where this file is used. Again, we can see there a 'My Devices' folder. By right clicking on the file inside it, we can see where this file is stored, and we can open it locally.

6 Future work

6.1 Discussion

While working with personal files and documents, people apply three main organisational approaches. There are three main organisational approaches which are called filing, piling and mixing. Organisation and re-finding documents requires time and effort. Depending of the applied organisational structure which can be filing, piling and mixing, the most effort is respectively required for adding something to the information space, re-finding in the information space or the load is equal for both processes. It is a true challenge for people to be able to organise their personal document in such a way, that re-finding and storing are as optimal as possible. And as we have seen in chapter 2, besides the physical documents, folders and papers, nowadays people also use electronic devices which extends people's personal information space with digital documents. This means that the digital files users operate on and which they organise, share the same organisational challenges as the physical documents. Moreover, people tend to use multiple devices and separate the tasks they perform on it, based on the characteristics of the device. More portable device such as a smartphone, is much smaller than a desktop computer which leads to the fact that people do not use it for the same set of tasks. The very first problem is that one's personal information space get fragmented across multiple devices. This problem has been targeted by several

applications, which were inspired by the ideas of the Memex which was described by the Vannevar Bush in 1945. Those applications are unified repositories of one's personal documents which are scattered across multiple devices. Conceptually, a unified repository can be compared to a central database with all documents or at least references to them. Some of those unified repositories are cloud services by them selves, such as Dropbox. This brings us to another problem typical to the last decade which has been solved yet. Those unified repositories operate on people's devices without incorporation of one's documents in the cloud. Along with multi-device usage phenomenon, the mobile Internet knew a great popularisation. This has lead to a promotion of a wide range of cloud services because disregarding the device, people can use the functionalities that are provided by a cloud service. There exist different cloud services for different purposes such as, Evernote simplifies the creation of quick notes on the way, Flickr provide the possibility to manage and share photo's, Facebook also makes it possible to manage photo's but mainly targets the social networking aspect and reflection via the timeline. The problem with those cloud service is that each one of them, forms a small piece of one' personal information space since it contains personal documents, but in an isolated way. For example, from the point the a photo gets uploaded to Facebook, the photo gets actually duplicated to Facebook. And there is no single way to track which personal photo's, from which devices are uploaded to Facebook, unless the person recalls it or goes and compares the content of his device and Facebook albums. This not only holds for Facebook but for most of the service. Creating a note in Evernote for example, also implies document duplication to an isolated part of personal information space. This is the second problem, information fragmentation not only across devices, but also across different cloud services. Thirdly, people access their information spaces not only at the moment of storage and the need to re-find something. Over time, one can have the desire to reflect on the content. For example, in case of a calendar, to recall how some event was scheduled some time ago, or again, in case of photo's reflect on how great the holidays last year were. People proactively search for reflection, and there exist PIM systems which are actually timeline based, such as Lifestreams. In the cloud, the best example is Facebook which is basically timeline based. Moreover, once in a year, around the year change, Facebook aggregates one's major events and happenings to create a personal year survey. Hence, people not only their personal information space for storage, but also for reflection. Al those three challenges were discussed in the requirements **??**. Those requirements were realised in the implementation of an application. From the technical point of view, the requirements contained the following points. The application has to extensible to support different (cloud) services. The user interface has be ubiquitous, in the sense that the application has to provide an aggregated overview of all users documents, scattered across multiple isolated information spaces. The reflection which is provided by the services, has to be made possible for visualisation. Moreover, the user interface has to be familiar to most part of the users ti minimise the adaptation process. As it turns

out, people build certain conceptual models about how an application should work based on their familiarity with certain interface type. Previously unseen and unfamiliar interfaces prevent usage of the application. The application also has to be ubiquitous in the sense that it has be support multiple user's devices, so it has to be cross-platform with multi-client architecture.

6.2 Conclusion

The resulted application, which is called EverSync, targets all those requirements. It is based on the underlying OC2 framework for creation of digital object (entities which represent the documents) and relations between them through navigational links. The application implements a client-server architecture whereby the client is implemented in JavaScript and is cross-platform. A client is the application installed on one of the devices which monitors files of that device. It is the application with user interface on which user can operate. The server is implemented in Java and is by itself a unified repository for the clients and plugins. A plugin is a integration layer for some (cloud) service to the EverSync. A plugin for a particular cloud service is responsible for monitoring of that service in order to detect file modifications and manipulations. Hence, plugins are installed on the server in order to communicate with third party services. A plugin has its own visualisations on the client, though. Therefore, the developer of a plugin has to provide a set of icons which will be pushed to each EverSync client. In other words, plugins installed on the server will be supported by the clients automatically by pushing some of the installation files. Clients and plugins notify the EverSync server about any modifications, which creates links between file duplicates across all the included pieces of one's information space. Each of the clients can then be used to explore the file relations. By this approach, when opening the client application the user sees a file tree with all his local files where this particular client is operating on. As request by the requirement, a file tree is the base representation model in most of the operating systems and is certainly familiar to all computer users. By selecting one of this local files, the user get another file tree rendered in the user interface, with an aggregated information where this file is used and duplicated. By default in EverSync, the user will see at least in which service this particular file is used and a list of user's devices which contain this file as well. Moreover, in between clients, user is able to see which of those clients are online and disregarding this connection status, user can manage the file across devices. A file can be opened on the current device, opened remotely and can be copied to some other client. Hereby, each plugin is free to choose it own representation hierarchy, since the default user gets to see is in which plugin a file is used. As an illustration of the EverSync extensibility, three plugins were implemented, for Evernote, Facebook and Flickr. Each of them realise different internal hierarchies, consider an example when user select a photo, then for other placed where this

file is stored, following hierarchies will be displayed per plugin: Evernote has *note – this file and all other files inside the note*, Facebook has *album – comments for this photo* and Flickr has only one level, namely *comments for this photo*. Again, each plugin is free to implement is own structures, while the document duplicates are linked automatically by the EverSync server. This makes the EverSync application in general, and the user interface in particular ubiquitous and extensible. The reflection requirement is realised through that explained ubiquitous plugin integration. Whenever a user selects a document in the file tree, all the information available on the services about that document will become visible. Depending on the plugin realisation and the service it is designed for, this can be a comment from a social network, a calendar date, a reminder, a note, etc.

6.3 Future Work

Despite those great characteristics, the application still requires validation in real world usage. It is expected that a validation test with users of several cloud services will provide much more insight in aspects that have to be improved. The current main drawback is its approach of linkage of files. For linkage, the search for file duplicates on different cloud services happens based on the file name, and on the devices based on the file path. This of course drastically limits the freedom of file management since the user won't be able to assign another name to the same file on different device or services. Moreover, this limitation has lead to the implementation to the so called *link queue*, explained in the **??**. However, despite the fact that this limitation is very drastic in the way how the application behaves, this is relatively easily solvable but requires additional implementation time which was not left for this thesis. The very best solution for this should be the usage of hashes generated based on the file content, instead of file names. This would provide the 'freedom' of file naming and relocation on the devices. However, this also would have implications of the performance of the application since each modification to the file content would require a hash recomputation. Moreover, while with the current approach, for the plugins it suffice to know only the names of all the files used in a service, the hashes approach would imply that the plugins should download the file content in order to compute the hash for the file. This is applicable for several low size document, but for several GB of content this can become challenging.
Another improvement for the EverSync would be is to extend it to one's physical information space. This is not realised in the current implementation, however is easily achievable thanks to the extensibility though plugins. An implementation of a plugin which could somehow integrate physical files would make EverSync a truly ubiquitous PIM application. However, this is only possible when the previous limitation is solved, since physical papers mostly do not have file names as digital files do. Using a hash code based on the file scan, could make that possible.

Bibliography

[1] Joao Aires and Daniel Gonçalves. Personal information dashboard-me, at a glance. In *PIM 2012 Workshop*, pages 1–8, 2012.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[3] Barry J Babin, William R Darden, and Mitch Griffin. Work and/or fun: measuring hedonic and utilitarian shopping value. *Journal of consumer research*, pages 644–656, 1994.

[4] Rajeev Batra and Olli T Ahtola. Measuring the hedonic and utilitarian sources of consumer attitudes. *Marketing letters*, 2(2):159–170, 1991.

[5] Gordon Bell, Jim Gemmell, and Roger Lueder. Challenges in using lifetime personal information stores. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 1–1. ACM, 2004.

[6] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, Ian Smith, and Rebecca E. Grinter. Quality versus quantity: e-mail-centric task management and its relation with overload. *Hum.-Comput. Interact.*, 20(1):89–138, June 2005.

[7] Victoria Bellotti and W. Keith Edwards. Intelligibility and accountability: Human considerations in context-aware systems. *Human-Computer Interaction*, 16(2-4):193–212, 2001.

[8] Ofer Bergman, Ruth Beyth-Marom, and Rafi Nachmias. The project fragmentation problem in personal information management. In *Proceedings of the 2006 Conference on Human Factors in Computing Systems, CHI 2006, Montréal, Québec, Canada, April 22-27, 2006*, pages 271–274, 2006.

[9] Ofer Bergman, Ruth Beyth-Marom, Rafi Nachmias, Noa Gradovitch, and Steve Whittaker. Improved search engines and navigation preference in personal information management. *ACM Trans. Inf. Syst.*, 26(4), 2008.

[10] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[11] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage. In *Proceedings of the 13th Conference on Human-Computer Interaction with Mobile Devices and Services, Mobile HCI 2011, Stockholm, Sweden, August 30 - September 2, 2011*, pages 47–56, 2011.

[12] Hylke Bons. SparkleShare Github Repository, April 2010.

[13] Marilynn B Brewer and Wendi Gardner. Who is this" we"? levels of collective identity and self representations. *Journal of personality and social psychology*, 71(1):83, 1996.

[14] Vannevar Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.

[15] Irene Cole. Human aspects of office filing: Implications for the electronic office. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 26, pages 59–63. SAGE Publications, 1982.

[16] Leandro S. G. de Carvalho, Raquel F. do Valle, Alexandre Passito, Edjair de Souza Mota, R. Novellino, and A. G. Penaranda. Synchronizing web browsing data with browserver. In *Proceedings of the 15th IEEE Symposium on Computers and Communications, ISCC 2010, Riccione, Italy, June 22-25, 2010*, pages 738–743, 2010.

[17] David Dearman and Jeffrey S. Pierce. It's on my other computer!: computing with multiple devices. In *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008*, pages 767–776, 2008.

[18] Idilio Drago, Marco Mellia, Maurizio M. Munafò, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: understanding personal cloud storage services. In John W. Byers, Jim Kurose, Ratul Mahajan, and Alex C. Snoeren, editors, *Proceedings of the 12th ACM SIGCOMM Internet Measurement Conference, IMC '12, Boston, MA, USA, November 14-16, 2012*, pages 481–494. ACM, 2012.

[19] Eric Freeman and David Gelernter. Lifestreams: A storage model for personal data. *SIGMOD Record*, 25(1):80–86, 1996.

[20] Jim Gemmell, Gordon Bell, Roger Lueder, Steven M. Drucker, and Curtis Wong. Mylifebits: fulfilling the memex vision. In Lawrence A. Rowe, Bernard Mérialdo, Max Mühlhäuser, Keith W. Ross, and Nevenka Dimitrova, editors, *Proceedings of the 10th ACM International Conference on Multimedia 2002, Juan les Pins, France, December 1-6, 2002.*, pages 235–238. ACM, 2002.

[21] Erving Goffman. *Encounters: two studies in the sociology of interaction.* Bobbs-Merrill, Indianapolis, 1961.

[22] Philipp C. Heckel. Syncany Github Repository, November 2012.

[23] Ken Hinckley. Synchronous gestures for multiple persons and computers. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology, Vancouver, Canada, November 2-5, 2003*, pages 149–158, 2003.

[24] John Horrigan. *Use of cloud computing applications and services.* Pew Internet & American Life Project, 2008.

[25] Judith A Howard. Social psychology of identities. *Annual review of sociology*, pages 367–393, 2000.

[26] Jeff A. Johnson and Austin Henderson. Conceptual models: begin by designing what to design. *Interactions*, 9(1):25–32, 2002.

[27] William Jones. *Keeping Found Things Found: The Study and Practice of Personal Information Management: The Study and Practice of Personal Information Management.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[28] William P. Jones, Harry Bruce, and Susan T. Dumais. Keeping found things found on the web. In *CIKM*, pages 119–126. ACM, 2001.

[29] Shaun K. Kane, Amy K. Karlson, Brian Meyers, Paul Johns, Andy Jacobs, and Greg Smith. Exploring cross-device web use on pcs and mobile devices. In *Human-Computer Interaction - Interact 2009, 12th IFIP TC 13 International Conference, Uppsala, Sweden, August 24-28, 2009, Proceedings, Part I*, pages 722–735, 2009.

[30] David R. Karger, Karun Bakshi, David Huynh, Dennis Quan, and Vineet Sinha. Haystack: A general-purpose information management tool for end users based on semistructured data. In *CIDR*, pages 13–26, 2005.

[31] David R. Karger and Dennis Quan. Haystack: a user interface for creating, browsing, and organizing arbitrary semistructured information. In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *Extended abstracts of the 2004 Conference on Human Factors in Computing Systems, CHI 2004, Vienna, Austria, April 24 - 29, 2004*, pages 777–778. ACM, 2004.

[32] Amy K. Karlson, Shamsi T. Iqbal, Brian Meyers, Gonzalo Ramos, Kathy Lee, and John C. Tang. Mobile taskflow in context: a screenshot study of smartphone usage. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, Georgia, USA, April 10-15, 2010*, pages 2009–2018, 2010.

[33] Ohbyung Kwon and Yixing Wen. An empirical study of the factors affecting social network service use. *Computers in Human Behavior*, 26(2):254–263, 2010.

[34] Mark W Lansdale. The psychology of personal information management. *Applied ergonomics*, 19(1):55–66, 1988.

[35] Thomas W. Malone. How do people organize their desks? implications for the design of office information systems. *ACM Trans. Inf. Syst.*, 1(1):99–112, 1983.

[36] Cathy Marshall and John C. Tang. That syncing feeling: early user experiences with the cloud. In *Conference on Designing Interactive Systems*, pages 544–553. ACM, 2012.

[37] Tara Matthews, Jeffrey Pierce, and John Tang. No smart phone is an island: The impact of places, situations, and other devices on smart phone use. *IBM RJ10452*, 2009.

[38] I. C. McIlwaine. The universal decimal classification: Some factors concerning its origins, development, and influence. *JASIS*, 48(4):331–339, 1997.

[39] Jared Norris. iFolderInstall - Community Ubuntu Documentation, October 2011.

[40] William Odom, Abigail Sellen, Richard H. R. Harper, and Eno Thereska. Lost in translation: understanding the possession of digital things in the cloud. In Joseph A. Konstan, Ed H. Chi, and Kristina Höök, editors, *CHI*, pages 781–790. ACM, 2012.

[41] Antti Oulasvirta and Lauri Sumari. Mobile kits and laptop trays: managing multiple devices in mobile information work. In *Proceedings of the 2007 Conference on Human Factors in Computing Systems, CHI 2007, San Jose, California, USA, April 28 - May 3, 2007*, pages 1127–1136, 2007.

[42] Jérôme Picault, Myriam Ribière, and Christophe Senot. Beyond life streams: activities and intentions for managing personal digital memories. In *the International Workshop on Adaptation, Personalization and Recommendation in the Social-semantic web (APRESW 2010), CEUR-WS*, volume 585, pages 25–32. Citeseer, 2010.

[43] Craig Ross, Emily S. Orr, Mia Sisic, Jaime M. Arseneault, Mary G. Simmering, and R. Robert Orr. Personality and motivations associated with facebook use. *Computers in Human Behavior*, 25(2):578–586, 2009.

[44] Leo Sauermann. The semantic desktop-a basis for personal knowledge management. In *Proceedings of the I-KNOW*, volume 5. Citeseer, 2005.

[45] Albrecht Schmidt, Michael Beigl, and Hans-Werner Gellersen. There is more to context than location. *Computers & Graphics*, 23(6):893–901, 1999.

[46] Abigail J Sellen and Richard H. R. Harper. *The myth of the paperless office*. MIT Press, Cambridge, Mass., 2002.

[47] Beat Signer and Moira C. Norrie. As we may link: A general metamodel for hypermedia systems. In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *ER*, volume 4801 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2007.

[48] Donghee Sinn and Sue Yeon Syn. Personal documentation on a social network site: Facebook, a collection of moments from your life? *Archival Science*, 14(2):95–124, 2014.

[49] Henry Song, Hao-Hua Chu, Nayeem Islam, Shoji Kurakake, and Masaji Katagiri. Browser state repository service. In *Pervasive Computing, First International Conference, Pervasive 2002, Zürich, Switzerland, August 26-28, 2002, Proceedings*, pages 253–266, 2002.

[50] Victoria Schwanda Sosik, Xuan Zhao, and Dan Cosley. See friendship, sort of: how conversation and digital traces might support reflection on friendships. In Steven E. Poltrock, Carla Simone, Jonathan Grudin, Gloria Mark, and John Riedl, editors, *CSCW '12 Computer Supported Cooperative Work, Seattle, WA, USA, February 11-15, 2012*, pages 1145–1154. ACM, 2012.

[51] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *CHI*, pages 415–422. ACM, 2004.

[52] Sandra Trullemans. Personal cross-media information management. Master's thesis, Vrije Universiteit Brussel, 2012-2013.

[53] Sandra Trullemans and Beat Signer. Towards a conceptual framework and metamodel for context-aware personal cross-media information management systems. *Proceedings of ER 2014*, 33rd International Conference on Conceptual Modelling, October 2014.

[54] Sherry Turkle. *The second self: computers and the human spirit*. MIT Press, Cambridge, Mass., 20th anniversary ed., 1st mit press ed edition, 2005.

[55] Jacob van Kokswijk. Granting personality to a virtual identity. *International Journal of Humanities and Social Sciences*, 3(8), 2008.

[56] Hannu Verkasalo. Contextual patterns in mobile service usage. *Personal and Ubiquitous Computing*, 13(5):331–342, 2009.

[57] Amy Voida, Judith S. Olson, and Gary M. Olson. Turbulence in the clouds: challenges of cloud-based information work. In Wendy E. Mackay, Stephen A. Brewster, and Susanne Bødker, editors, *CHI*, pages 2273–2282. ACM, 2013.

[58] Jiaqiu Wang and Zhongjie Wang. A survey on personal data cloud. *ScientificWorldJournal*, 2014:969150, 2014.

[59] Michel Wedel and Wagner A Kamakura. *Market segmentation: conceptual and methodological foundations.* Kluwer Academic, Boston, 2nd ed edition, 2000.

[60] Steve Whittaker. Supporting collaborative task management in e-mail. *Hum.-Comput. Interact.*, 20(1):49–88, June 2005.

[61] Steve Whittaker, Quentin Jones, Bonnie A. Nardi, Mike Creech, Loren G. Terveen, Ellen Isaacs, and John Hainsworth. Contactmap: Organizing communication in a social desktop. *ACM Trans. Comput.-Hum. Interact.*, 11(4):445–471, 2004.