Vrije Universiteit Brussel

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

Re-finding Physical Documents: It is Like a Dream Come True

Graduation thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in de toegepaste informatica

Tim Reynaert

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

Academic year 2014-2015

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN
VAKGROEP COMPUTERWETENSCHAPPEN

Re-finding Physical Documents: It is Like a Dream Come True

Masterproef ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de toegepaste informatica

Tim Reynaert

Promotor: Prof. Dr. Beat Signer
Begeleider: Sandra Trullemans

Academiejaar 2014-2015

Abstract

On a daily basis, people experience issues with keeping, organising and re-finding their personal information. Keeping, organising and re-finding are also known as the KOR activities. Most of these issues are caused by information overload, information fragmentation and the burden of classifying digital and physical documents. In the Personal Information Management (PIM) research field, these three main issues are investigated and tools are been developed to support users during the three KOR activities. In this thesis, we dig deeper into the re-finding activity of physical documents. Most applications focus only on spatial indications to show where a document is located. Nevertheless, descriptive PIM research has shown that the spatial cue is not the only re-finding cue that users apply in their re-finding activities in physical space. By only indicating where a document is, the overall context of the document is not given. Such extra information can include in which tasks the documents is used or when the document was used in a timespan. This extra contextual and time-related information is as important as the spatial cue if it comes to re-finding documents. In addition to the missing cues, users often have to execute a digital search before a spatial augmentation can be invoked. At the same time, users often do not know exactly which document they are searching for. Lastly, most of the systems do not have a structured central repository to keep track of the contextual and time-related meta-information of the documents. When this data would be available, it could be used when augmenting the physical world. In order to do an initial step towards a solution for the described issues and shortcomings, the KOR framework is developed. The framework's functionality is to track and organise digital and physical documents as well as to provide developers with a mechanism to rapidly prototype re-finding user interfaces in digital and physical space. Due to the extensiveness and complexity of the KOR framework, this thesis focuses on the development of the mechanism for re-finding user interfaces. First, an extension is made to the KOR framework's core module called ReViTa. This module is responsible for the data management of spatial, contextual and time-related information of documents. The extension also includes a model of the required data of re-finding user interfaces. Secondly, a mechanism for re-finding user interfaces is developed. This component of the KOR framework allows developers to easily register and reuse re-finding user interfaces as well as use meta-information in their applications. It was our goal that developers do not need any knowledge of the underlying complexity of tracking documents and how documents are stored in the ReViTa component. Furthermore, the re-finding module is also responsible for ensuring the right re-finding cues for the right organisational

structures. For example, descriptive research indicates that users do not use any time cue for re-finding a document in a pile but rather use a contextual cue. Therefore, an augmented user interface for a pile has to contain support for a contextual cue. Finally, we present various augmented re-finding user interfaces for piles, filing cabinets and paper trays. These developed user interfaces show the potential of the KOR framework in an Office of the Future setting.

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

First of all I want to thank my promoter Prof. Dr. Beat Signer to offer me the chance to write a thesis at the WISE lab and provide feedback during the presentations. A second person I want to thank is my supervisor Ph.D. student Sandra Trullemans. She guided me through this thesis and provided useful information to be able to finish this thesis. I also want to thank my parents, my brother, all my friends and my co thesis students, Audrey Sanctorum, Jasmien De Ridder and Ayrton Vercruysse, to support me during the writing of this thesis. The atmosphere at the WISE lab helped me a lot to focus on the thesis but, at the same time, provided some spare time to support me when it was more difficult to write. Thank you, Alison Govaerts to inspire me during the writing of the use case examples of the re-finding user interfaces.

Contents

List of Figures

List of Tables

1 Introduction

1.1 What is Personal Information Management?

On a daily basis, people have issues with organising their own personal information. Already in the forties, Bush described the mismatch between how we organise our information and how the human brain processes this information [5]. Information has to be classified before it can be stored somewhere. In digital space, users use the file system whereas in physical space filing cabinets are commonly used. This is different from how the human brain processes information. The human brain keeps associations between information. Bush can be seen as one of the pioneers in PIM research. Jones described the PIM research area as the study of PIM activities in order to understand how people keep, organise and re-find personal information:

" Personal Information Management (PIM) refers to both the practice and the study of the activities a person performs in order to locate or create, store, organize, maintain, modify, retrieve, use and distribute information in each of its many forms (in various paper forms, in electronic documents, in email messages, in conventional Web pages, in blogs, in wikis, etc.) as needed to meet life's many goals (everyday and long-term, work-related and not) and to fulfill life's many roles and responsibilities (as parent, spouse, friend, employee, member of community, etc.) " William Jones, 2010 [26]

This definition shows that PIM focuses on the research of what a person does in order to, for instance, store, organise and re-find information. Apart from this research, there is also a technical part that focuses on system design.

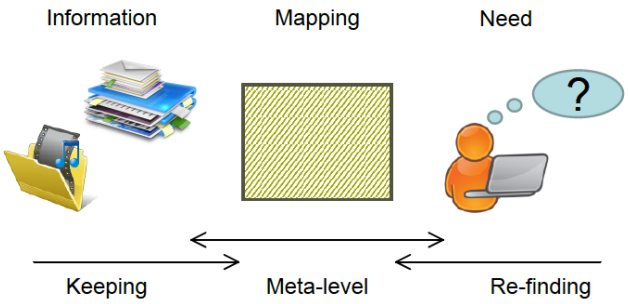

Figure 1.1: Keeping-Organising-Re-finding

The definition of PIM led to the Keeping-Organising-Re-finding (KOR) conceptual framework. Figure 1.1 illustrates the KOR framework. Keeping, organising and re-finding are the three general activities of PIM [26]. For example, documents, papers, emails, photos or videos, etc. are stored by the user in the digital as well as the physical world. Keeping focuses on the decision of integrating the information gathered, encountered or received in the user's own personal information space. For example, the user will receive a lot of emails but has to decide whether every mail is important and needs to be kept. Re-finding activities focus on the activities a user executes to re-find stored information in their personal information space. The user can re-find documents by, for example, using a search engine or navigating through the digital file system. In order to re-find information, the information has to be stored beforehand in the personal information space. In order to connect the keeping and re-finding activities, some meta-level activities are needed. The main meta-level activity consists of organising and maintaining stored information in the personal information space.

In digital as well as physical space, there are some organisational structures that the user will use to store information. Malone defined the filing and piling organisational structures [39]. This was later extended with mixing by Trullemans [51]. Every organisational structure has different properties depending on labelling and ordering. Files are used to archive documents and have strict rules concerning ordering and labelling of the individual elements. Elements have to be labelled and ordered according to a structure. Piles cannot have an internal order and are not labelled. Mixtures are everything that falls in between filing and piling. There are no specific rules about labelling

and ordering. A typical mixture structure can often be found in ring binders and paper trays where the structure itself is labelled but the elements in the structure are not. When people are re-finding documents, files, piles and mixtures provide specific cues. Depending on the organisational structure, a context, spatial or time cue is provided. These cues will help to re-find the document the person is looking for.

1.2 A Daily Struggle

The first problem is that there is too much information. The human brain can only process a limited amount of information [47]. Because of this, the retrieval of information depending on a user's needs is sometimes difficult. Important information is overlooked or will not always be remembered or recalled. As described before, Bush already mentioned that there is a problem how we process information that leads to re-finding problems. At that time, only physical information was available. There was no digital space and thus there was "less" information available. Nowadays, the problem of information overload is even worse since the revolution of the digital age makes information very easily accessible. For example, pictures are shared via social media or the World Wide Web allows people to keep relevant and most of all irrelevant information.

Another problem in PIM is the classification problem. Ordering files in a systematic order will take more time and thus the user will tend to skip the classification of documents. Users do not want to spend time to classify documents if there is no benefit [6]. The classification problem can be divided into two separate problems [10]. The first problem is that people find it difficult to label documents. The second problem is that a document can only be placed in one category in the physical space and this often also holds in digital space. Those problems make it difficult to classify the document if it has to be stored in multiple categories.

The last problem concerns information fragmentation. Nowadays most people have a laptop, a smartphone, a tablet, USB drives and cloud storage space. The laptop will give the user access to information stored on the laptop as well as the USB drives and the cloud services. The smartphone can store data like messages and emails. With the tablet, the user can access the cloud space and create documents using some of the applications. The biggest problem is that the user has to remember where which document is stored and information is fragmented over multiple devices and places.

1.3 Digital Systems in Overload

To solve the issues of PIM described in the previous section, various digital systems have been built. The goal is to try to solve the three PIM issues namely information fragmentation, information overload and lack of organisation. In digital space, we can derive three categories of tools: visualisation, augmentation and central repository tools.

1.3.1 Visualisation Tools

In the visualisation category, we can see tools that apply visualisation techniques on personal information data and tools that try to mimic the physical space by showing a 3D representation of the workspace.

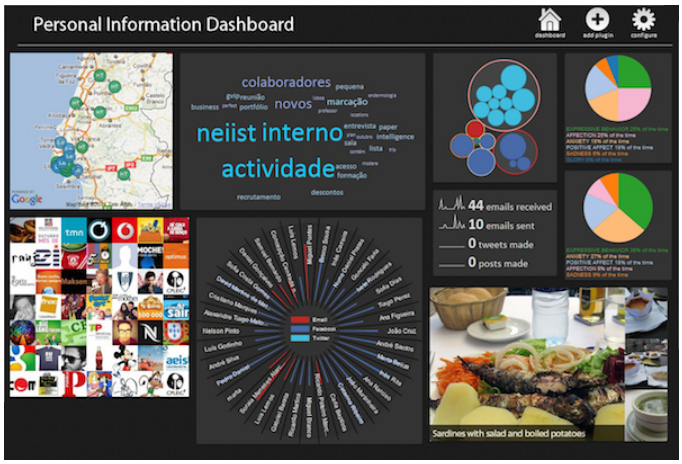

Figure 1.2: Personal Information Dashboard

The first system that provides a visualisation of the user's data and tries to solve the information overload problem in PIM is called Personal Information Dashboard [2]. The Personal Information Dashboard focuses on combining all kinds of data sources. Figure 1.2 shows an example of a configured Personal Information Dashboard. The goal of the Personal Information Dashboard is to combine different sources of personal data, find interesting patterns of the user's life and show the user what is important at that moment. Implicit refinding is used when the user is looking for data. For example, in the keyword cloud, the user can see the most important words that were received on a given day. The Personal Information Dashboard is a plug-in based system. In total 10 plug-ins were implemented, ranging from a keyword cloud, which shows the most important keywords in mail posts and status updates of social media, to Who&How which keeps track of activities with each of your

contacts and Spark Stats which gives an overview of all communication done via mail and social media. Other systems like Themail [55] proposed a way of visualizing personal information using piles of important words.

Figure 1.3: BumpTop

In a second visualisation approach, researchers try to copy the physical space. People have to categorise information when they are storing data in a digital file system. This takes more effort than just piling the documents like they can do in real life. Therefore, BumpTop [1] provides a virtual organisation tool that consists of a virtual desktop. Figure 1.3 shows an overview of the BumpTop interface. The goal is to support the pile organisational structure in digital space. Files are represented as 3D objects that can be tossed around. If the user wants to build a pile that has a meaning they can do this. Physics of the real world are added to give the virtual desktop a more natural feeling. For instance, when the user starts to toss files in a corner, a heap starts to form.

1.3.2 Augmentation Tools

As described before, users are forced to file documents in digital space. Fitchett studied ways on how to improve navigational file retrieval [14]. The focus is more on retrieval and re-finding than on how to store and organise the documents. Successful retrieval mainly depends on the user. The user will have to remember the location of the file. Nardi describes three types of information which are organised in different ways [41]. Those three types are temporal information, working information and archived information. Hierarchical structures have a direct influence on navigational time [38] [40]. In Fitchett's work, three goals are defined. The first goal is to minimise the time

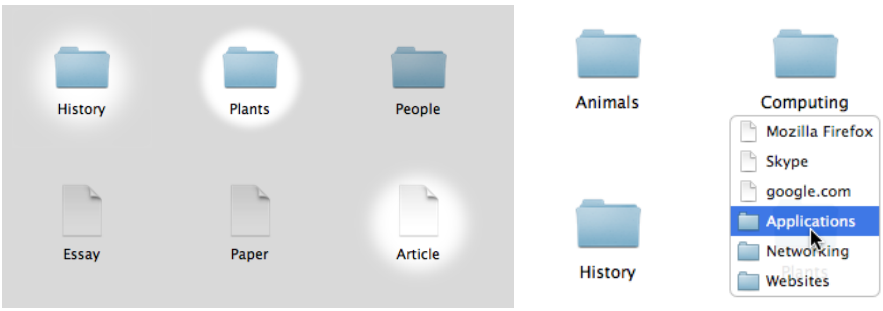

Figure 1.4: The Icon Highlights and the Hover Menu

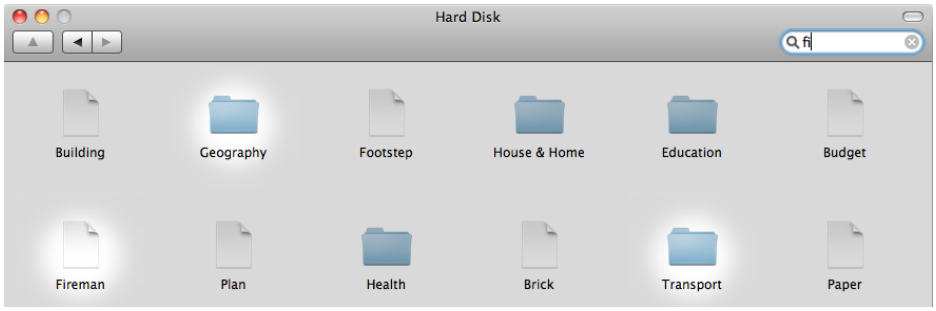

Figure 1.5: Search directed navigation

spent on each level while the user is looking for the file. The second goal is to reduce the number of levels traversed to the target file by providing shortcuts. The last goal is to promote rehearsal of the retrieval mechanics and facilitate expertise. To accomplish this, three new user interfaces which are illustrated in Figure 1.4 and Figure 1.5 were built. Icon Highlights predicts which items are most likely to be accessed using the AccessRank algorithm [13]. Hover Menus provides access to commonly accessed items. Search Directed Navigation allow the user to search for a file based on a query. Figure 1.4 and 1.5 show examples of Icon Highlights, Hover Menus and Search Directed Navigation [15].

1.3.3 Central Repository Tools

In order to overcome information fragmentation, the vision of a central repository for personal information was raised in the forties. Bush envisioned the Memex as a central system which would contain all documents a user kept. The novelty of this vision lies in the fact that the organisation of all these documents would no longer be hierarchical. Instead, Bush aimed for linking documents in order to allow users to organise their documents as they natu-

rally do in their brain. Bush described the Memex as follows:

"A memex is a device in which an individual stores all his books, records and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory." Vannevar Bush, 1945 [5]

Most of the content is purchased on microfilm but the user can also add their own content by photographing it to an empty space on the microfilm. The user can recall information by typing the code linked to that piece of information. Everything is structured in a scheme. When viewing a document, simple notes and comments can be added. The Memex can be seen as the first central repository that stores information the user wants to keep and provide easy access by indexing the information. Nevertheless, the Memex was never built.

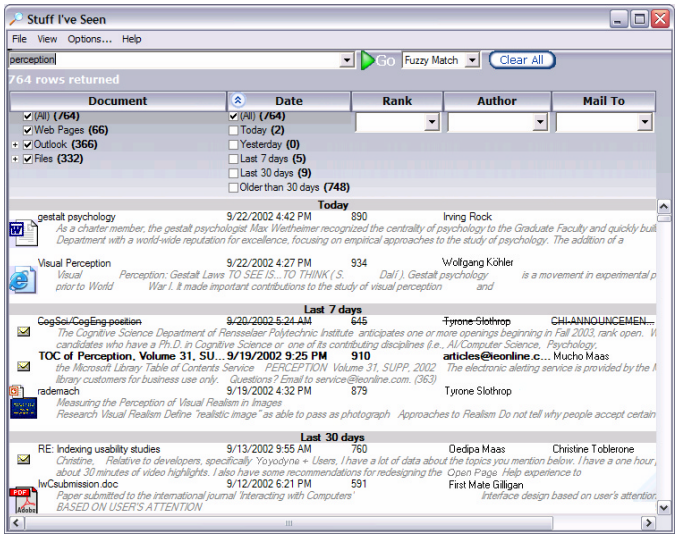

Figure 1.6: Stuff I've seen

A system that is based on the idea of the Memex is called *Stuff I've Seen* [9]. The goal is the same as with the Memex, which is to build a unified index of information seen by the user. The system is an extension of Microsoft's Search Indexing architecture and uses a rich user interface that allows the user to search through their information. Figure 1.6 shows the user-interface of SIS that consists of a search field. This search field is used to query the index. One of the biggest disadvantages of SIS is that the user explicitly has to search for something. This is not natural and sometimes the user will not find what they were looking for [50].

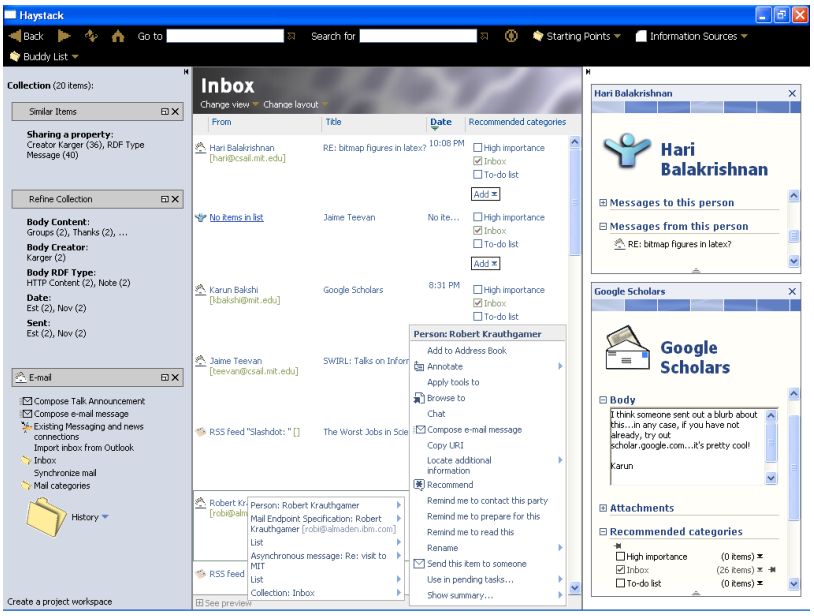

Figure 1.7: Haystack

Other systems that provide some sort of information management are Lifestreams [16], Presto [8] and MediaFinder [28]. They all use a data model to push the systems beyond a single domain and allow cross-domain information extraction. The most promising system is Haystack developed at MIT [30] [29]. Haystack is a general-purpose information management system [30] [29]. The data, as well as metadata, of Haystack, is stored in a directed graph. The user can add links to this data and recall important information. The interface of Haystack is shown in Figure 1.7.

1.3.4 Other Tools

The biggest disadvantage of digital documents is that the spatial information is lost. A system that tries to bridge between digital documents and the physical world is called BubbleFish [12]. BubbleFish is an augmented reality application that shows digital documents in the physical world. Digital documents are represented as a bubble in the physical world. The user can interact and organise the digital documents as if they were physical documents and place them somewhere in the physical world. BubbleFish consists of a Sony Glasstron[1] along with a head-mounted camera.

[1] `http://www.sony.net/Fun/design/history/product/1990/plm-s700.html`

1.4 What About Physical Documents?

Digital and physical documents each have different properties, advantages and disadvantages. Digital documents lack the tangibility and universal acceptance of physical paper. Features that are missed when working with physical paper are, for example, direct access to a wide variety of interactive functions that are available in digital documents. A solution to these problems is to provide a digital alternative of the physical document. Nevertheless, this is not always the best option. For example, research has shown that reading from a screen is, in general, slower than from paper [18]. It is also widely accepted that the Paperless Offices is more a myth than reality [47]. An alternative solution is to augment real-world objects [56] [57]. The goal of augmenting documents in the real world is to reduce incompatibilities between digital documents and physical documents by augmenting the physical documents with digital information. To allow the digital system to track the physical documents, a vision system is used [33]. This system will capture the documents and digitise, identify and locate them depending on the application. In order to enhance the office working experience, interactive horizontal displays can provide a big benefit.

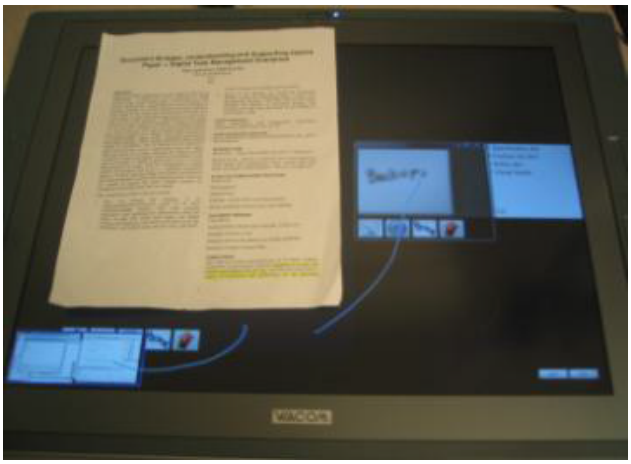

Figure 1.8: DocuDesk

A system that uses a vision system and an interactive display is the DigitalDesk [58]. The DigitalDesk is the first approach to augment digital documents, EnhanceDesk [36] is an enhancement of DigitalDesk. Marcel [42], DocuDesk [11] and PaperSpace [49] are other systems that use a vision system to track and detect documents. DocuDesk is shown in Figure 1.8.

1.5 Problem Statement

There are various shortcomings with current augmented reality applications aiming to support users in re-finding activities. First, they are mostly standalone applications which only support spatial indications of where a physical document is. Descriptive PIM research indicates that the spatial cue is not the main re-finding cue but falls next to contextual and time cues. Secondly, as discussed earlier, re-finding activities are much more complex. In existing augmented reality PIM systems, users have to use digital search or navigate through a digital pile to first find the digital version and only then the spatial augmentation can be invoked. Nevertheless, users often do not exactly know which document they search for and have to do the search activity digitally. Although, this way of interaction is already an improvement, there is still a switch between the digital and physical space. Finally, unified PIM systems are known for their central repository approach where every personal document can be linked to each other. These systems try to overcome information fragmentation and help users in better organising their information space. However, current augmented reality PIM systems do not make use of these systems to integrate additional meta information in their augmentations.

1.6 Thesis Contribution

In this thesis, we present the KOR framework. The KOR framework is based on various requirements. By fulfilling these requirements, the KOR framework tries to solve the previously mentioned problems. First, it provides implicit re-finding support for physical documents. This is done by augmenting existing organisational structures. For example, paper trays will show the context of available documents by lighting LEDs in a specific colour. Secondly, re-finding user interfaces have to integrate the right re-finding cues when augmenting existing organisational structures. As introduced before, a pile invokes the context and spatial cue when users want to re-find a document in it. Therefore, the augmented user interface should provide support that can be used by these two cues. Third, because the KOR framework should support rapid prototyping, it has to be extensible. This allows, for example, to easily add a new re-finding user interface that augments a newly purchased filing cabinet.

The KOR framework's architecture is a client-server architecture where the server is responsible for providing access to the OC2 PIM system. The clients include re-finding user interfaces, tracking devices, or visualisation

applications. The server side contains three components namely the tracking module, the ReViTa module and the re-finding module. The tracking module is responsible for tracking documents in physical space, identifying the documents and storing the locations in the PIM system. ReViTa is responsible for communicating with OC2 and also contains an extension of OC2 to provide support for organisational structures and re-finding cues. ReViTa provides a mapping to OC2 to keep track of the re-finding user interfaces. The re-finding module provides support for management and registration of re-finding user interfaces to an organisational structure and taking into consideration if the organisational structure supports the re-finding cue triggered by the re-finding user interface. The re-finding module also provides an interface to the re-finding user interfaces if specific information fromReViTa is needed. My contribution to this large framework is the re-finding component and some extensions made to the ReViTa component. In addition, we present various augmented re-finding user interfaces for piles, filing cabinets and paper trays. These developed user interfaces show the potential of the KOR framework in an Office of the Future setting.

1.7 Thesis Outline

This thesis will start with giving an overview on how people organise and re-find their documents. Organisational strategies filing, piling and mixing as well as the re-finding cues, context cue, spatial cue and time cue, are introduced. Next an overview of existing systems that augment physical documents are presented with an overview of their problems. Now that we know what the problems are with existing systems, requirements of our system are presented. The focus is to build a framework that supports the KOR principles and supports implicit physical search, integrate re-finding cues in re-finding user interfaces, unified repository integration and extensibility. The KOR framework is further elaborated as well as the architecture and the three modules of the framework. Every module, tracking, ReViTa and re-finding are discussed. The extension of ReViTa and the re-finding module are elaborated in detail and an overview of the implemented re-finding cues is given. Next, every implemented re-finding module is elaborated in detail. To finish, a conclusion is presented as well as some future work.

2 Background

2.1 How People Organise Their Documents

Malone did research to find out how people organise their information [39]. His research shows that there are two categories of offices namely messy offices and neat offices. Messy offices are offices that in general contain a lot of piles. Most of those piles are stacks of things to do. The piles are not necessarily ordered in any particular of way. Because of the amount of information that is not ordered in any particular of way, less important information sometimes tends to get lost in all the piles and will never be processed. On the other hand, neat offices mostly rely on filing documents and thus limit the number of piles. If there are piles, they are organised in a particular way on the desk.

	Elements labelled	Elements ordered	Groups labelled	Groups ordered
Files	Yes	Yes	?	?
Piles	?	No	No	?

Table 2.1: Malone's definition of Files and Piles

Malones observations led to the definition of the two most common organisational structures namely *filing* and *piling*. These definitions are illustrated in Table 2.1. The individual elements of a file are explicitly arranged in some

kind of order. For example, this order can be alphabetical or chronological. The individual elements of a file also have to be explicitly labelled. A group of files, for instance, a filing cabinet, can be labelled but does not necessarily have to. The group can be arranged in a specific order but this is not required. Individual elements of a pile are not necessarily labelled. They are also not ordered in any kind of way. Piles themselves are not titled and do not necessarily have any kind of order when placed on the desk. Piles do not have a specific order but their spatial location is important to re-find them. The file structure defined by Tsichritzis matches the file structure of Malone [53]. He also defined a heap which matches the pile definition of Malone.

Figure 2.1: Filing and piling example

The definition of files and piles is not limited to the organisational structures on their own. For example, a filing cabinet is called a `file` organisational structure because the file folders in the filing cabinet (i.e. elements) are ordered and labelled. The elements are placed in an order and each element has a label. Thus, the internal structure of that filing cabinet is called a `file`. A shelf of ring binders that is not ordered and is not labelled is called a `pile`. Inside the ring binder, the structure is called a `file` if every element is labelled and follows a specific order.

Figure 2.2: Mixtures examples

Let us have a look at a file folder in a filing cabinet. When the elements of the file folder are not ordered and explicitly labelled, the definition of file or pile is not satisfied. Thus, there has to be a third organisational strategy. Every organisational strategy that does not follow the definition of filing or piling will follow the definition of this third organisational strategy. Organisational strategies, where the elements are optionally labelled and do not have an explicit order, are not following the definition of files or piles. A group of an organisational structure that is sometimes labelled but not always and also has no explicit order is neither a file nor a pile. This organisational structure is called a mixture. The mixing organisational strategy was first defined by Trullemans as:

"Mixing is an organisational strategy which is neither filing nor piling. Mixtures as an organisational unit do not match the definition for files and piles by Malone. They may contain titled as well as untitled elements and elements might be explicitly ordered. A group might be titled and groups can also be explicitly ordered whereby the titles and ordering of elements do not have to be consistent." Sandra Trullemans, 2014 [51]

Let us recall the example of the file folder in a filing cabinet, if the elements of that file folder are explicitly labelled but do not follow any order, the definition of files or piles was not satisfied. With the previously defined mixture organisational structure, the content of that file folder can be called a mixture. Another example is paper trays. The trays have no specific order and some of the trays can be labelled.

Because of the organised filing structure people tend to re-find the information they need without overlooking less important information. We can say that people with messy offices have more problems to re-find information than people with a neat office. Nevertheless, this is not totally true for every office. Spending a lot of time to organise a filing cabinet does not always have the expected result. Sometimes too much time is used to find a system that works in relation to the result it gives. One of the reasons that people tend to end up with more piles and thus a messy desk has to do with the cognitive effort needed to file a document in comparison to starting or continuing a pile. Some people have difficulties to classify a document and file in the correct file folder [6]. The difficulty of classifying the document will lead to the creation of many vaguely classified piles [10]. Another problem is that a specific order has to be followed when filing a document. This takes time to complete. Sometimes it is just easier to start a pile rather than taking the time to find out where to file the document [6]. The time gained by creating

a pile is lost when looking for a document in the pile. On the other hand, a pile provides a reminding function [47]. People tend to be reminded of the things they have to do instead of looking for what has to be done. Retrieving a document from a file folder is less difficult because the file folders are ordered and labelled. In general people tend to pile more often [6].

2.2 Re-finding Documents

In this section, the re-finding cues that help re-finding documents are discussed. Every organisational strategy provides support for specific re-finding cues. Trullemans studied these re-finding cues in detail [51]. Table 2.2 gives an overview of the available cues in relation to the organisational strategies discussed before.

	Context Cue	Spatial Cue	Time Cue
Filing	x	x	x
Piling	x	x	
Mixing	x		x

Table 2.2: Re-finding cues [51]

The use of a classification system will enhance the re-finding effectiveness [25]. Filing is best suited for cold information because of structuring. Piles provide more contextual and spatial information for hot and warm information. Despite the difficulties of re-finding documents in piles, piles also provide a reminder function based on the spatial location [47]. Piles do also preserve context. Apart from spatial information, this context information is also used when retrieving a document from a pile [39]. Research has proven that the context cue is one of the main advantages of the human memory to recall information [39] [32] [4]. The spatial position of piles is also important to improve the re-finding process [27]. The user can, for example, remember where on the desk the upcoming tasks are placed. Every time a document is placed on the pile, the time information and thus time cue is lost compared to the other organisational strategies.

The filing of documents will take a lot of cognitive effort because the internal organisation of the file has to be followed. This only will benefit in the ease of re-finding a document that is located in a file. When filing a document, the context information is gone. A label is responsible for adding context information to the file again. The label in combination with the structure of a file provide an overall context for the user [34]. Users also tend to annotate the files with reminders and add timestamps to files. When

the user starts to look for a document the last known location where the document is last classified is important to know where to start looking [39]. Files also provide a spatial cue [6]. A filing cabinet is placed somewhere in the office that the user will remember and locate easily. This will help to know where the document is located. The exact location of the file folder can be found if the user knows, for instance, the context of the document.

In mixtures, contexts are less used to re-find documents. The uses tends to annotate the mixture with a timestamp to help re-finding documents based on a time cue. Because the mixing organisational structure was only presented in 2014, more research is needed to find out more about the re-finding cues supported by the mixing organisational structure.

2.3 PIM Systems in Physical Space

The systems that provide support for PIM in physical space can be divided into two large categories, namely systems that support organising activities and systems that support organisational structures. Every system has its advantages and disadvantages and supports some kind of re-finding. Re-finding can be implicit by triggering a re-finding cue or explicit which is often based on searching in digital space.

2.3.1 Support for Organising Activities

In this section, systems that provide support for organising activities are presented. Every system will use a vision system to track and locate physical documents.

A system called DigitalDesk provides computer-based interaction with paper documents [42]. The setup of DigitalDesk is shown in Figure 2.3. A projector mounted above the desk is used to augment objects and documents placed on the physical desk. Also, a camera is mounted above the desk to track what documents are placed on the desk [37]. DigitalDesk allows the user to interact with a digital calculator where the input is done by pointing to the numbers on a physical paper. Another use case is that sentences pointed by the user on a physical document are translated and projected next to the physical document. DigitalDesk provides a basic interplay between physical documents and a digital connection, without making a complete digital copy of the document. There is only a one-to-one mapping between the physical and digital world.

Other systems that only provide a one-to-one relationship between digital and physical documents are PADD [17], The Designers' Outpost [35],

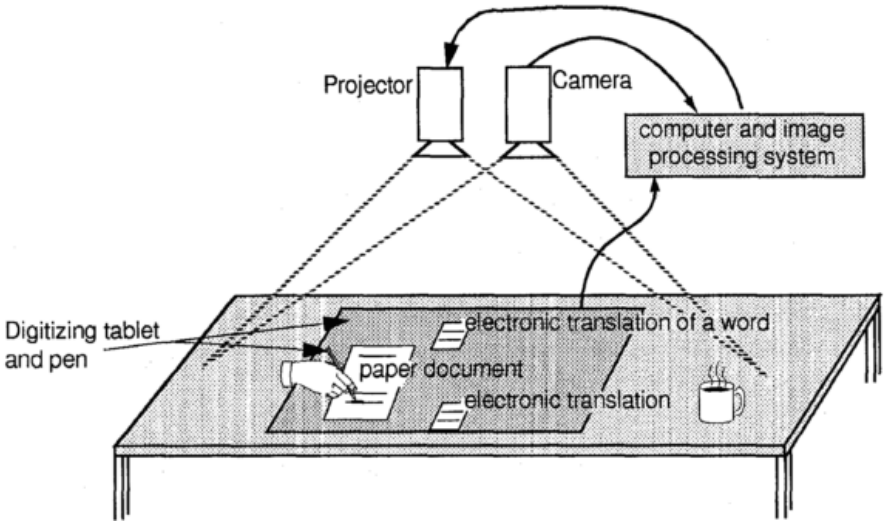

Figure 2.3: Setup of DigitalDesk

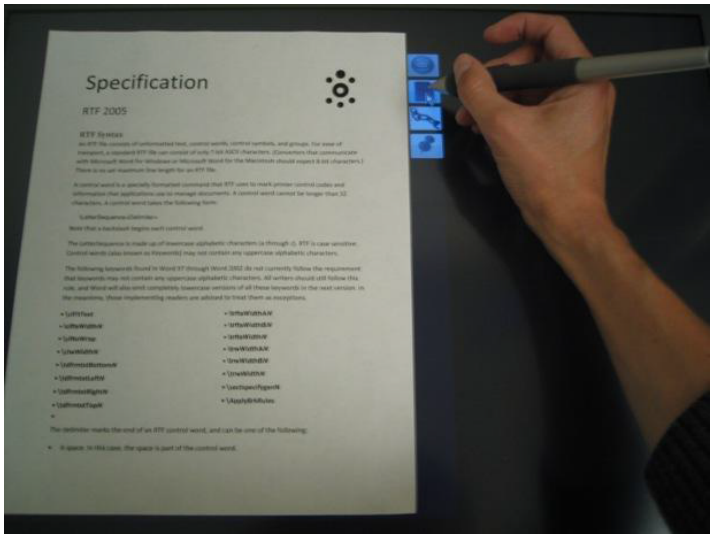

Figure 2.4: Menu of DocuDesk

PaperSpace [49], ButterflyNet [59] and Print-n-Link [43]. PADD transfers notes captured by an Anoto digital pen to the digital representation of the document [17]. The Designers' Outpost captures sticky notes on a digital whiteboard. When the physical sticky notes are removed, digital copies are shown [35]. ButterflyNet uses the Anoto pen to annotate time-stamped digital photos [59]. Print-n-Link uses the Anoto pen to create a link between physical paper and digital information and/or searches for cited documents

[43]. An interesting system that provides many-to-many links between digital and physical documents is called DocuDesk [11]. DocuDesk consists of a display laying on top of the desk. Above the display, a camera with IR-filer is responsible for tracking and detecting a document placed on top of the display. Also, a stylus is tracked and used for user input. To find out which document the user placed on the display, the system will look for a barcode printed on the document. If there is no barcode, an image of the document is captured. This barcode or image is used to look for the digital representation of the document. When the document is identified and located above the display, a menu is displayed next to it. An example of this menu in combination with a document can be seen in Figure 2.4. The menu allows interaction with that document. The menu provides support to, for instance, email the document, open the digital version of the document or store a new digital copy of the document. An option to pin the document will open the digital version and align it with the physical location of the document. Now that the digital document is recalled, the physical document can be removed. The most important feature of DocuDesk is linking. This feature allows to create links between the document and digital as well as physical documents. Useful documents can now be linked to each other and easily recalled when needed. When the user, for instance, is writing a report about a paper, the paper can be digitalised in DocuDesk. Related documents in both the physical as digital world can be linked to that paper. Using this feature, the user can built a network of information between the original paper and all the related content.

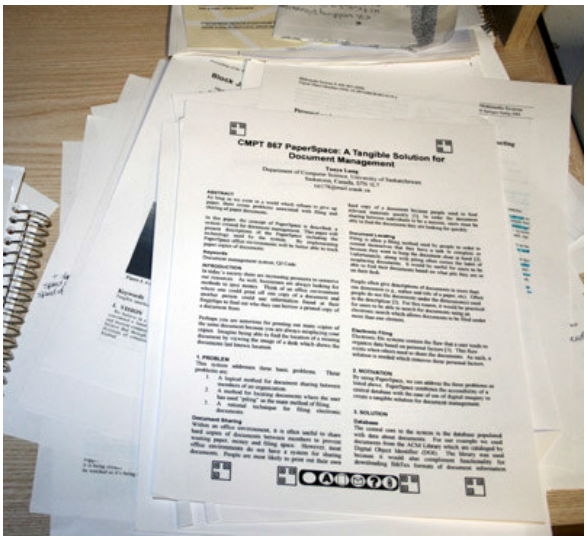

Figure 2.5: PaperSpace example

Another system that reuses the idea of interacting with a document on the desk is PaperSpace. An example of a document printed with PaperSpace is shown in Figure 2.5. PaperSpace allows the combination of digital and physical documents on a desk [49]. The goal of PaperSpace is to manage academic papers. PaperSpace wants to reduce printed documents by informing the user if the document is already printed when the user wants to print a document. When the document was already printed and misplaced, PaperSpace will show where it is. Apart from the database, the system consists of a tracking code generator, a camera system, a tracker and a user interface. The tracking code generator is used to generate DOI (Document Object Identifier) tags and store location information. When the document is printed using PaperSpace, an encoded tag containing the DOI, user ID, page corner identifier, print date, version and page number is added to the document. Also, a command bar is added. This command bar allows interactions with the document by providing a tangible interface. The physical document, as well as the command bar, are part of this interface. Interactions with the command bar are more or less the same as with DocuDesk, the only difference here is that there is no augmentation of the document using projectors of displays. The user will use two fingers to interact with the document and can annotate, link and email the document. These interactions by tracked with the camera.

2.3.2 Support for Organisational Structures

This section presents systems that provide support for organisational structures. These systems focus on the filing and piling organisational structure.

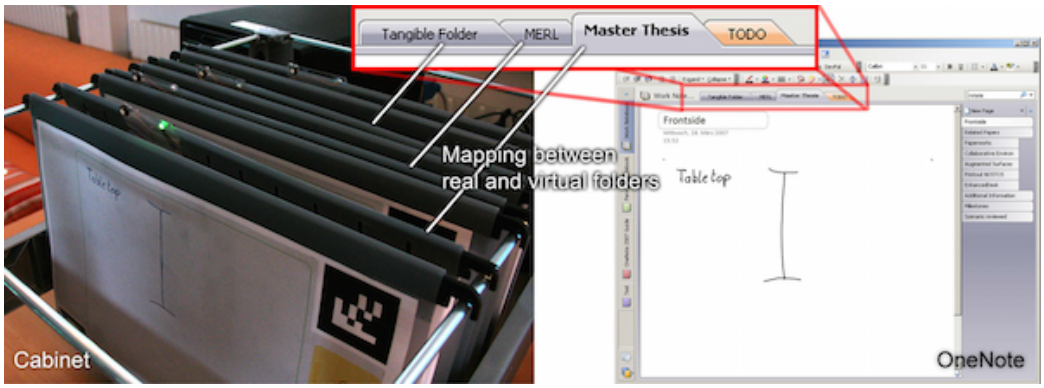

Figure 2.6: Smart Filing System

Jervis built various systems that are focussing on the filing organisational structure. Every system has some advantages and disadvantages. The dis-

advantages are most of the time solved in a new revision of the system. The first system built by Jervis is called "Smart Filing System" [46]. The goal of this system is to combine the benefits of digital documents as well as a physical counterpart of that same document or vice versa. In this system, the file folders in a filing cabinet are augmented and made intelligent so that they can be linked to a digital representation stored in Microsoft OneNote. Figure 2.6 shows an example of the file folders in the real world as well as the mapping with OneNote. The filing cabinet or vertical filing cabinet, invented by Seibls in 1898 [44], already exists for a long time. It is a good system for archiving documents but the retrieval of documents is still a problem. Because people still prefer to print out their documents [47], a system that maps the digital version with the physical one is very useful. This is where the Smart Filing System comes in handy and supports the connection between digital and physical documents organised in a filing organisational structure. Every file folder is extended with a small LED and push button. The hardware in the file folder also provides a unique ID to the file folder. Communication with the file folders is done using a bus system called 1-Wire. In this system only, one wire is needed and a ground wire. Thus, the two hooks to hold the file folder in the filing cabinet are used to communicate with the file folder. Apart from the button and LED, the front side of the file folder consists of an Anoto paper. This allows the user to write annotations on the file folder. The annotations are tracked by an Anoto pen and directly sent to OneNote. The button on the file folder allows to navigate between the virtual folders in OneNote. When performing a search in OneNote, the LEDs of the file folders are used to show where the search results are located in the physical world. The LED can also be used to show which file folder the user is interacting with. One of the disadvantages of the system is that the user has to start a digital search if they want to re-find a file folder thus this is not natural. Another disadvantage is that the location of a document is not known in a very detailed way. The system can only know if the document is in a file folder in the filing cabinet or not. A LED can show the exact location but the system does not know how the file folders are ordered or where exactly the file folder is located. Therefore, the system does not know the exact location of a file folder in the filing cabinet. Another disadvantage is that the hooks are used to communicate with the file folders. This results in a bad connection between the file folders and the filing cabinet. Because of the way the hardware is built, lots of file folders will slow down the hardware.

One of the problems with the Smart Filing System is that the hardware will start to react slow when a lot of file folders are added to the filing cabinet. A solution is to decouple the tangible hardware from the software.

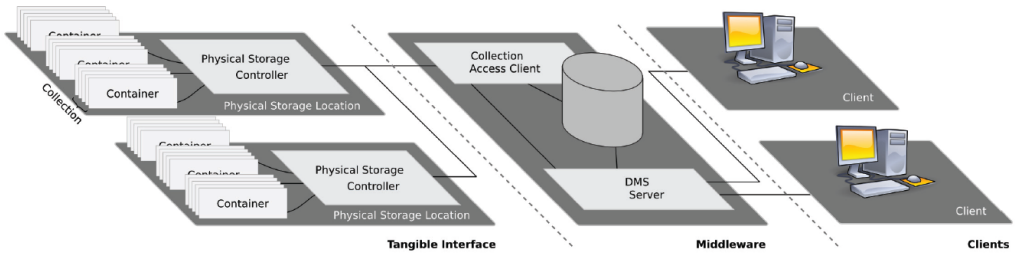

Figure 2.7: SOPHYA architecture

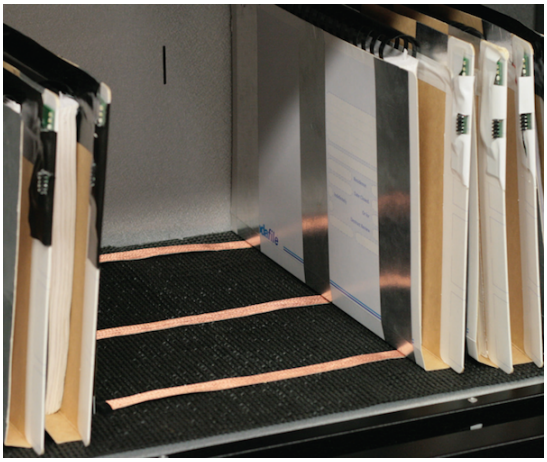

Figure 2.8: File folders in SOPHYA

SOPHYA or "Smart Organisation of PHYsical Artefacts" fixes the slowness of the hardware by placing a middleware between the tangible interfaces and the clients [22]. As can be seen in Figure 2.7, there is no more direct communication between the hardware and clients. A more distributed approach of the architecture is used to split the load of scanning for changes in the tangible part of the architecture. Because the number of devices on the bus are smaller, this results in a faster response of the tangible part of the system. The only advantage is that the file folders are represented as containers that can contain documents. These containers contain all the hardware and augmentations. Another hardware improvement compared to the Smart Filing System is the change to lateral file folders instead of conventional vertical file folders. The advantage of using lateral file folders is that the file folders are not hanging on hooks anymore. Now, it is possible to use any amount of wires needed to communicate with the hardware mounted on a file folder. A proprietary protocol was designed that gives more freedom in communicating with the file folders. Figure 2.8 shows an example of the lateral file

folders. Nevertheless, even now there still are some problems with SOPHYA such as containers are used instead of documents. Because of this it is still not possible for the system to link a document to a specific location. The exact location of a document or container is also difficult to find because the system cannot remember the order of the containers, the containers are connected to a bus system that does not allow exact location detection.

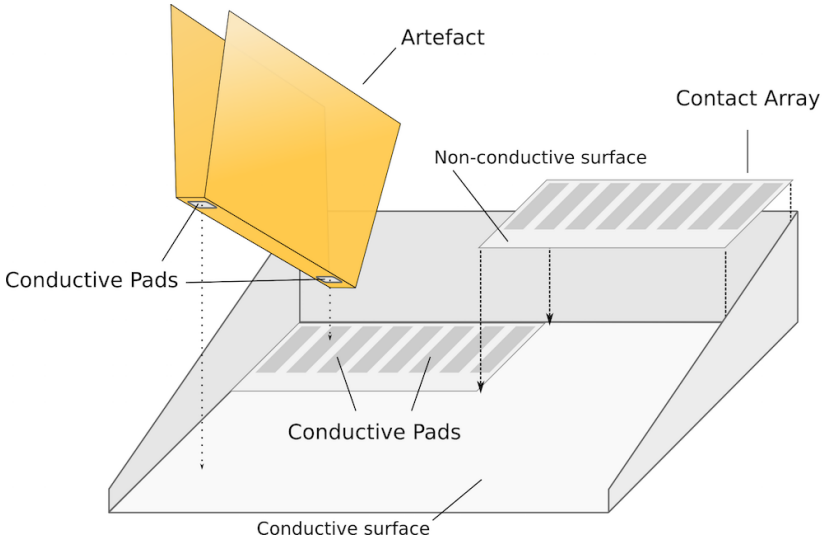

Figure 2.9: File folders in SOPHYA V2.0

To fix some of the problems existing in SOPHYA, SOPHYA V2.0 has been built [23] [24]. One of the biggest problems is that the location of a document is not very specific. Others have built systems that use RFID tags to track the location of a document [3]. Hark described a system where the RFID tag is printed onto the paper using e-ink technology [19]. The Smart Shelf system designed by Decker can locate documents depending on the position of the document above an array of RFID antennas [7]. The system developed by Hinske extends this idea and allows location of a document on a touch table [20]. Hinske also built a system with a moving arm under the table that changes the location of the RFID reader [21]. These systems provide a valid solution to locate a document on a desk. The only problem is that location of individual items in a dense area is difficult. It is, for instance, impossible for those systems to locate a document in a filing cabinet and provide an exact location. This is where the redesigned hardware of SOPHYA comes in handy. This time the containers of SOPHYA are placed on a contact array. Every contact of the array matches an ID that is linked to a known position by the system. This allows SOPHYA to find out where exactly the container is located. To allow this to work, conductive pads have to be mounted in the

filing cabinet. Figure 2.9 shows a schematic overview how this will work. At the same time, the display function of the file folders is separated from the file folders. The file folders still have to be intelligent but only have to contain some hardware to store the unique ID of a file folder. As Jervis already made a good attempt at building a re-finding system that can help the user, there are still some problems left to solve. The focus is only on filing structures without any knowledge of cues. A link between organisational strategies and re-finding cues can be the cue to build an augmented PIM system that can help the users re-finding documents.

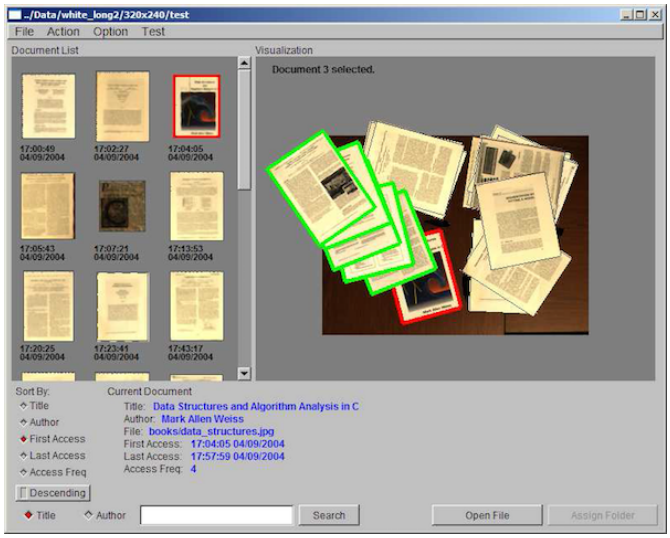

Figure 2.10: Video-based document tracking

Kim built a system to track documents placed on a pile [33]. Using a vision system, the system can keep track of when a document is added to the pile and where it is located. The user can re-find a document by querying the system and asking when a document was last accessed, last appeared or by initiating a digital search based on keywords. The user interface is shown in Figure 2.10. The system can also be used to sort photos. The user starts by printing out the photos. Next they start to form piles of the physical photos of photos they want to keep together. To finish every pile is assigned to a folder and the photos corresponding to that pile are placed in the corresponding folder. Apart from the support of the pile organisational structure, sort the digital representation of the documents based on last accessed and last appeared. Nevertheless, piles do not support a time cue making it difficult to re-find a document based on time. Digital search is also used to re-find a document based on keywords, this acts as an explicit digital search and supports a spatial cue.

2.4 Where it Goes Wrong!

There are still some issues with augmented systems that want to provide re-finding support to the user. Based on the organisational strategies and previous research, re-finding is not as simple as just showing where the document is located using a spatial cue. Context and time cues are also an important factor when re-finding documents. Another problem that is very clearly available is that the user has to start a digital search before the re-finding of a document will start. This makes it difficult for the user because they have to switch between a digital and physical space. None of the existing systems take this into account. A last problem that arises is that none of the discussed systems is using any kind of system to integrate the augmentations into a unified PIM system.

	Implicit re-finding			Explicit re-finding	Search in digital space
	Context Cue	Spatial Cue	Time Cue		
DigitalDesk	x	-	-	-	-
DocuDesk	x	-	-	-	+/-
PaperSpace	x	-	-	-	+/-
Smart Filing	-	x	-	x	x
SOPHYA	-	x	-	x	x
SOPHYA V2.0	-	x	-	x	x
Pile tracking	-	x	x	x	x
Goal	**x**	**x**	**x**	**x**	**-**

Table 2.3: Overview of PIM systems in physical space

The discussed systems can solve part of the re-finding problem but still not fix other problems with PIM. Integrating the concepts of organisational strategies and re-finding cues in a PIM system and augments the physical world can be a valid solution to provide support and make re-finding of documents easier and more natural. As shown in Table 2.3, none of the previously presented systems provides correct support for re-finding documents. The goal is that the system supports implicit re-finding based on the re-finding cues, context, spatial and time. At the same time, the system has to support explicit re-finding in a way that the user does not have to start a digital search.

3

The KOR Framework

In this thesis, we present the KOR framework. The KOR framework is a rapid prototyping framework for applications that provide support for keeping-organising-re-finding activities. In order to support such a rapid prototyping framework, the functionality of tracking documents, organising the documents by means of associations and managing re-finding user interfaces for explicit and implicit search integrations has to be provided. Due to the magnitude of the KOR framework, this thesis focuses on the re-finding component. First, the re-finding component requirements are discussed followed by the frameworks general architecture and its individual components. Finally, the re-finding component and implementation is elaborated.

3.1 Requirements for Re-finding

This section will discuss the requirements that the framework and re-finding module will have to support. As described before in the problem statement, implicit physical search, as well as re-finding cues and unified repository integration, are important requirements. On top of this, the re-finding module also needs to be extensible.

Implicit physical search Most of existing systems created let the user do a digital search to re-find a physical document [46]. The problem is that this creates a less natural user interface. At the same time, the user has to focus on keywords to search for a document. This will not always give the correct document because it is not always known in advance what document has to be found [50]. To support non-digital search, the augmented user interface has to detect if the user is, for instance, near the filing cabinet. If this is the case, the filing cabinet can be augmented to trigger the human brain.

Integrate re-finding cues in re-finding user interface Because search is not natural, re-finding has to be done another way. As discussed before, every organisational strategy has different re-finding cues. Those re-finding cues help the user to find the document based on context, spatial or time cues [51]. Each re-finding user interface will trigger one of those cues. For example, a filing cabinet is alphabetically ordered. This will result in context that gets lost [34]. The re-finding user interface will have to provide the context in the user interface. Another function that also should be available concerning the re-finding cues is management of the re-finding user interfaces. A mechanism that checks if it is possible to add a re-finding user interface to an organisational structure. For example, a pile cannot provide support for a time cue.

Unified repository integration A system is needed to keep track of linked documents and metadata such as time, location and context. The unified repository has to support context-awareness as well as extensibility for organisational structures and re-finding cues.

Extensibility Since we provide a rapid prototyping framework, it is necessary to not limit the functionality of tracking, organising and re-finding. In the re-finding component, this means that the application should be able to re-use existing user interfaces. It should also be possible to easily import new re-finding user interfaces in the re-finding component.

3.2 Architecture

As described before, the architecture of the rapid prototyping framework is based on the theoretical KOR framework [26]. For every KOR activity, the framework provides an individual module. Therefore, the KOR framework consists of a tracking module, the ReViTa module and a re-finding module. In Figure 3.1 a schematic overview of the framework is seen.

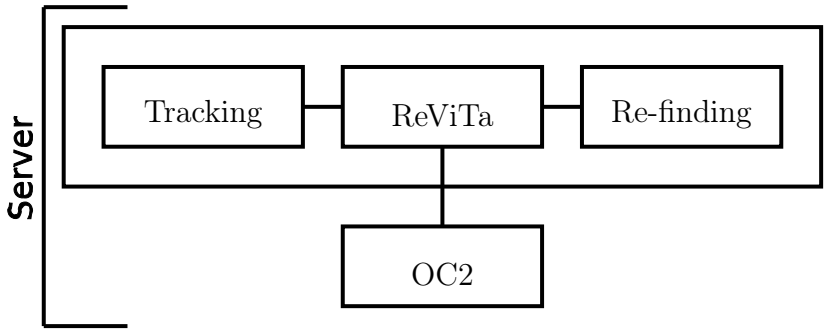

Figure 3.1: KOR Framework

The tracking component is responsible for tracking the documents and reporting back where they are placed in the physical world. ReViTa is responsible for the communication with the OC2 PIM system. ReViTa is also responsible for providing support for organisational strategies and re-finding cues on top of OC2. The third module focuses on re-finding. Re-finding cannot be done without knowing where the documents are located in the real world or knowing what re-finding user interfaces are available. Therefore, it is important to have the tracking and ReViTa modules. Every module is implemented in Java and provides a RESTful interface to allow communication from external applications. The KOR framework is implemented as a server to serve as a rapid prototyping framework.

In the next sections, every module is elaborated. As this thesis focuses more on re-finding, the re-finding module, as well as the re-finding user interfaces, are discussed in more detail.

3.3 ReViTa Module

One of the modules in the KOR framework is ReViTa. ReViTa is responsible for providing a connection to the OC2 PIM system. The OC2 metamodel is defined by Trullemans and Signer [52]. The OC2 framework was used because it provides context-awareness and support for physical documents compared to other PIM systems such as OntoPim [31], Haystack, Semantic Desktop [45]. At the same, time ReViTa provides support for organisational structures and re-finding cues by mapping data to OC2. The following subsections will provide a detailed explanation of OC2 as well as the extensions made in ReViTa.

3.3.1 Object-Concept-Context Model

The Object-Concept-Context model or OC2 is a conceptual framework for context-aware personal cross-media information management systems [52]. Besides the integration of context-awareness and physical documents, the OC2 was also designed to support both episodic as well as semantic memory [31]. Episodic memory focuses on creating links between information and perceptible properties like events. Semantic memory focuses more on the links between concepts.

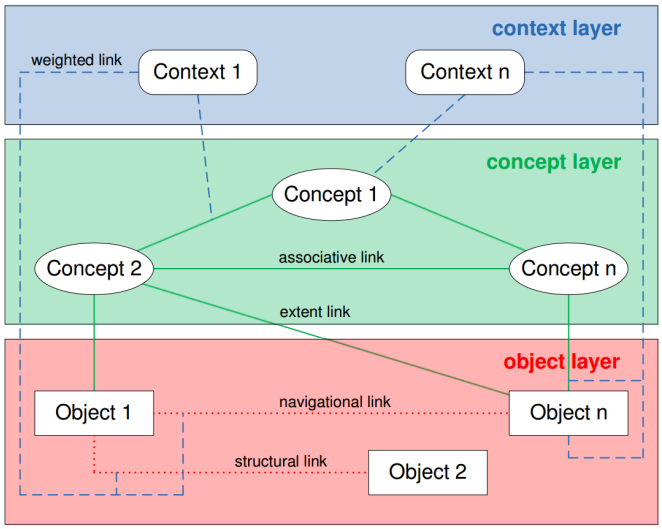

Figure 3.2: OC2 conceptual model

The OC2 conceptual framework illustrates the low-level metamodel in a more abstract form and is given in Figure 3.2. The conceptual framework is designed in three layers namely the object, concept and context layer.

Objects located in the object layer are the physical and digital pieces of information. This can, for example, be a physical document or email. Every object is uniquely identified and can contain other objects. Objects can be linked by using navigational and structural links. Navigational links are links that represent a navigational relationship between objects. Structural links are links between objects in order to express a structure. A common use case would be a tree structure of documents such as the digital file system. On top of the object layer, there is the concept layer. Concepts abstract the complexity of the real world. For example, concepts can be seen as the labels of a paper tray. Associative links allow two concepts to be associated with each other. A relation between a concept and an object is represented by an extent link. Typically this can be seen as a categorisation of objects to a

concept. For example, in terms of a file system, this is the categorisation of documents in a folder where the folder represents the concept. The top layer of the OC2 metamodel is the context layer. A context can be seen as a user's activity. Objects and concepts can have a certain relevance to a context as well as the links between them.

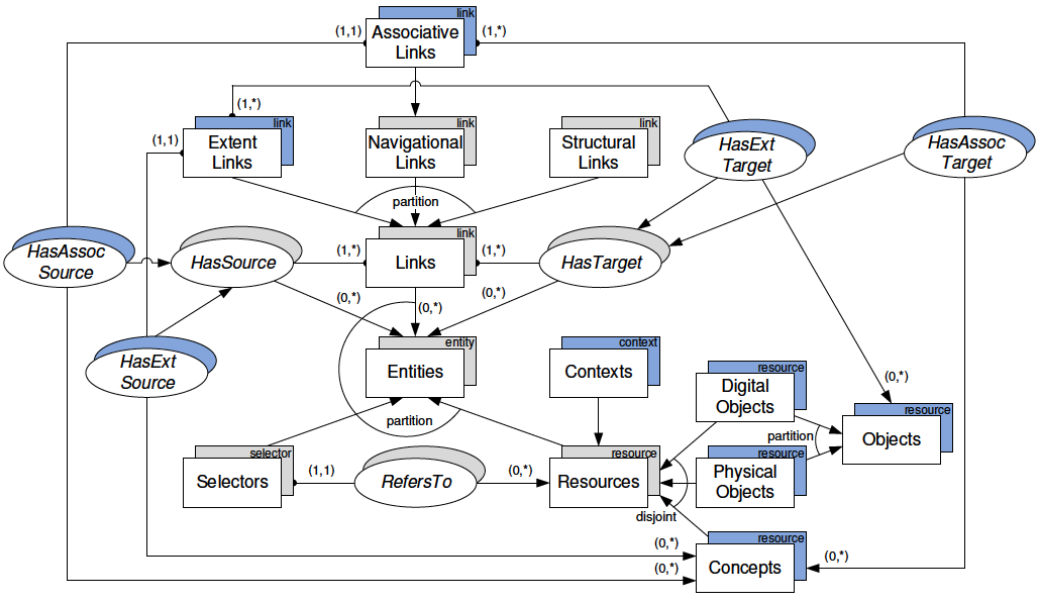

Figure 3.3: OC2 Metamodel

The OC2 conceptual framework has been translated to the OC2 metamodel. The OC2 metamodel is an extension of the Resource-Selector-Link (RSL) metamodel. The RSL model is a metamodel for cross-media information spaces [48]. `Entities` are the centre element of the RSL model as can be seen in Figure 3.3. An entity can be a resource, link or selector. `Links` represent the relationship between entities. `Resources` represent files or other information. `Selectors` are introduced to define areas in a resource. By subclassifying them from `Entities`, individual pieces of a resource can be linked. `Entities` can also have different properties. To support the OC2 metamodel, the basic RSL model had to be extended to support the different types of links. `Associative Links` allow two concepts to be associated with each other. `Extent Links` allow a concept and object to be linked. `Objects` can represent `Digital Objects` and `Physical Objects`.

3.3.2 ReViTa Extension

First of all, an extension of OC2 is needed to have support for re-finding user interfaces, organisational strategies and re-finding cues. This is provided by the ReViTa module of the framework. The extensions to the OC2 metamodel are not implemented as defined in the OM model. This model is shown in Figure 3.4. In contrast, we have used the OC2 as a metamodel. Every extension made is translated into collections, structural links, links and resources. Note that, therefore, the extended model could also easily be modelled in other languages such as ER or ORM. Nevertheless, to keep consistency with the OC2 model we used OM.

Since objects and concepts share a lot of the same associations, we first defined a supercollection `O&C` which contains both. `LocationProperties` uses property objects to store the location of an object. This property will contain the physical or digital location of the object. `LifeLines` will keep the history of an object by not only logging location changes but also interactions. The history will contain timestamps when there was an interaction with the object and is built out of Line objects. `LifeLines` are useful to check when, for instance, a document is last accessed.

Support for organisational functions and thus organisational structures is added using the `OrganisationalStructure` subcollection of `Structures`. Because `Filing`, `Piling` and `Mixing` collections have different properties, these structures are subpartitions of the `OrganisationalStucture` collection. `Filing` and `Mixing` have extra associations namely `HasFilingLabel` or `HasMixingLabel`. A label is an OC2 concept. The associations are different because the definition of a file and a mixture are different. Files always need a label whereas with a mixture this is not required. Finally, the collection `OrganisationalLinks` is added. `OrganisationalLinks` is a subcollection of `StructuralLinks`. An organisational link can only have one source of `Objects` and can only have child objects. For example, a pile needs one organisational link. The source of this link represents the root. This root is the object that represents the pile. This way, links can easily be added to structures. To add a physical pile to the system that contains two documents, three physical objects are created. Two for the documents and one for the pile. The pile object is the source of a new organisational link. The two physical documents are added as targets. A new organisational structure is created and named "desk pile". The organisational link is added to this structure. To finish, the structure "desk pile" is added to the `Piling` collection. OC2 also needs to be extended to support a re-finding functionality. The organisational functionality already provides a lot of information to refind documents but there are still missing some things. First of all, a generic

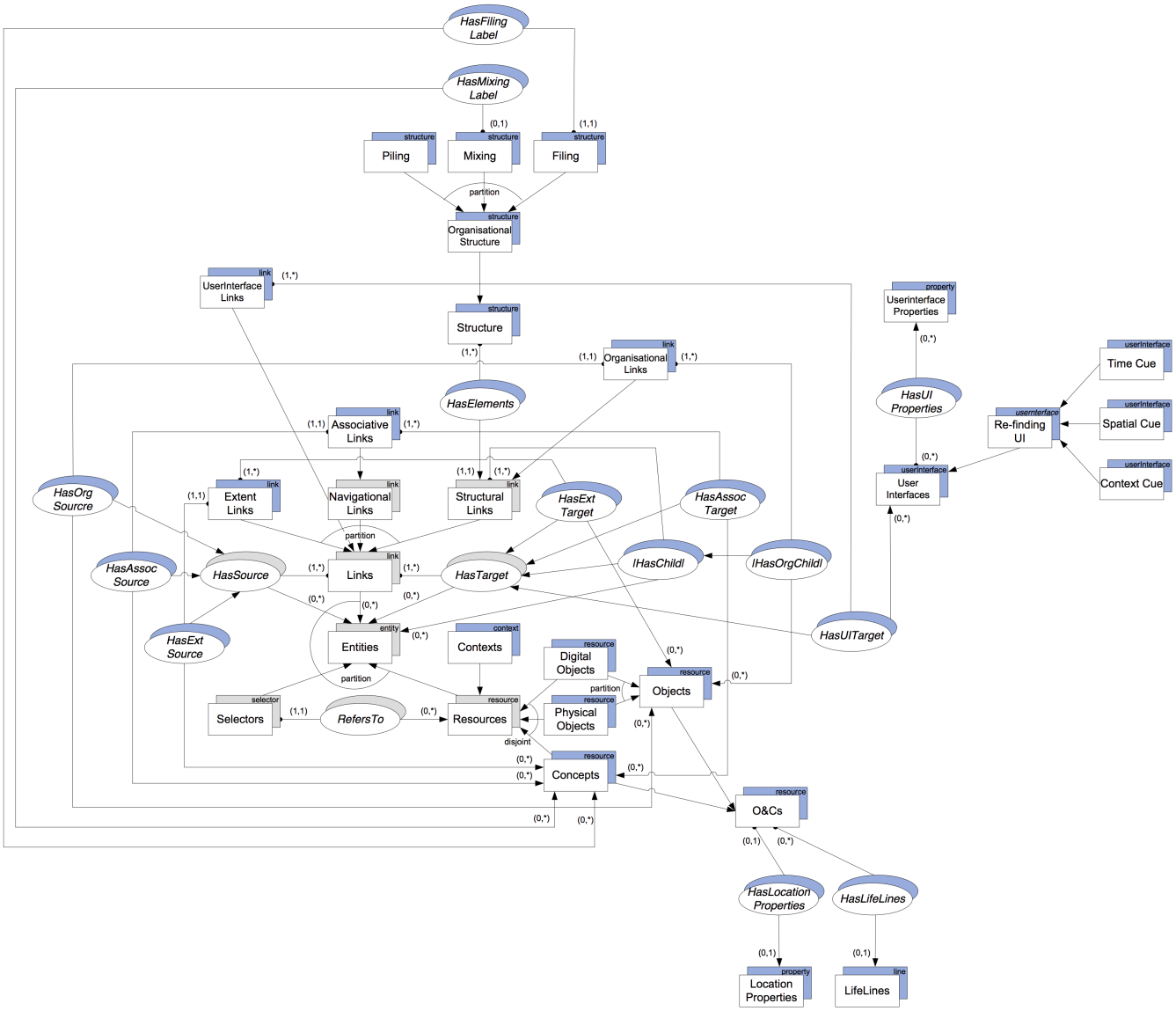

Figure 3.4: Extended OC2

collection `UserInterfaces` is added to support user interfaces. Without the `UserInterfaceLinks`, no object can be linked to a user interface thus this is also added. The target of the `UserInterfaceLinks` is an instance of the collection `UserInterfaces` rather than an `Entity`. `HasTarget` is subtyped to support `UserInterfaces` with `HasUITarget`. Properties of user interfaces are linked using the `UserInterfaceProperties` collection. Examples of properties are the required devices, available modalities, etc. To com-

plete the support of the re-finding functionality in OC2 subcollections of `UserInterfaces`, `Re-finding UI` is added. Re-finding user interfaces have the subcollections `ContextCue`, `SpatialCue` and `TimeCue` to keep track on which user interface supports which cue.

To allow easy access to ReViTa, a RESTful interface provides endpoints to communicate with the specific functions. This allows third-party applications to easily communicate with every feature of ReViTa.

3.4 Tracking Module

The tracking module of the framework is responsible for tracking the documents in the physical world and store the actual location of a document in OC2. This information will later be used by the re-finding module to locate a document when, for instance, explicit re-finding is asked by a re-finding user interface or external application. As this is not the focus of the thesis, we assume that this information is already available and can be accessed with ReViTa. It is important to know that documents are located in or at a specific resource ID. This resource ID can represent a file folder in a filing cabinet, a paper tray in a stack of paper trays or even a pile on the desk. For further information, we refer the reader to the Master Thesis of Ayrton Vercruysse [54].

3.5 Re-finding Module

In this section, the re-finding module is elaborated in detail. An overview of the implemented re-finding user interfaces is also provided. The following sections contain a detailed discussion of the re-finding user interfaces.

3.5.1 Re-finding Mechanism

The re-finding module consists of four main functions. The first one is to provide a possibility to register and manage new re-finding user interfaces in the system. A second function is to invoke a re-finding user interface when a request from a trigger is received. The third function of the re-finding module is to provide additional data to the re-finding user interface if needed. A last function provided by the re-finding module is a system to easily communicate with extra displays used by the re-finding user interfaces.

In the next sections, the four functionalities are elaborated. To explain every function in detail, a scheme of the data flow is drawn in Figure 3.5.

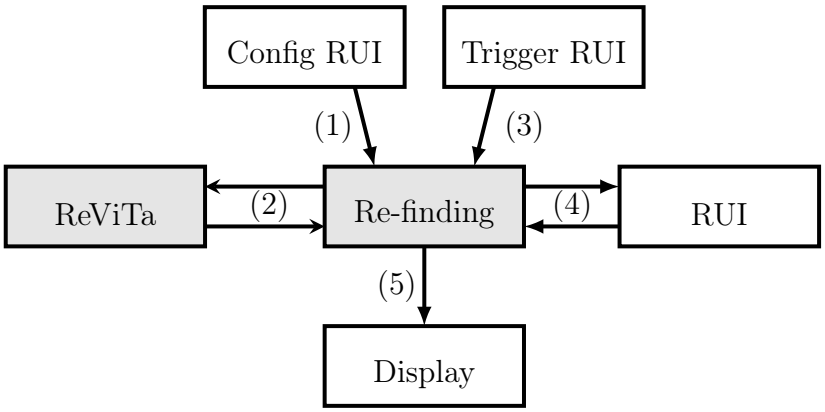

Figure 3.5: Data flow

Parts that run on the server are coloured grey. Every function is accessible using a RESTful interface.

Management of Re-finding User Interfaces

An important function of the re-finding module is to provide some kind of management of the re-finding user interfaces. This function results in a system that is extensible. At the same time, the system has to be intelligent enough to check whether or not a re-finding user interface can be connected to an organisational structure. This is based on the re-finding cues the organisational structure supports. For example, a pile does not support the time cue thus no re-finding user interface of the type time cue can be added to a pile.

As can be seen in Figure 3.5(1), the new user interface is registered by sending a data object directly to the re-finding module. This data object is sent from a configuration user interface and consists of the name of the user interface as well as the supported cue and organisational structure that is augmented. The re-finding module provides an endpoint `/register` that accepts this data object using the `POST` method. The data object itself is JSON encoded. When the re-finding module receives an object on this endpoint, the module will first check whether this re-finding user interface can be added to the asked organisational structure. A request from the re-finding module will ask the ReViTa module to get the details of the organisational structure. The request to ReViTa can be seen in Figure 3.5(2). When the re-finding module receives the object from ReViTa containing all the data of the requested organisational structure, the module will check if the requested re-finding cue is available on that organisational structure. The re-finding

module is aware of the different cues supported by the organisational structures. If everything is fine, a request is made to ReViTa to register the re-finding user interface. The response from the initial request will either be successful or failed depending on this check.

Invoking a Re-finding User Interface

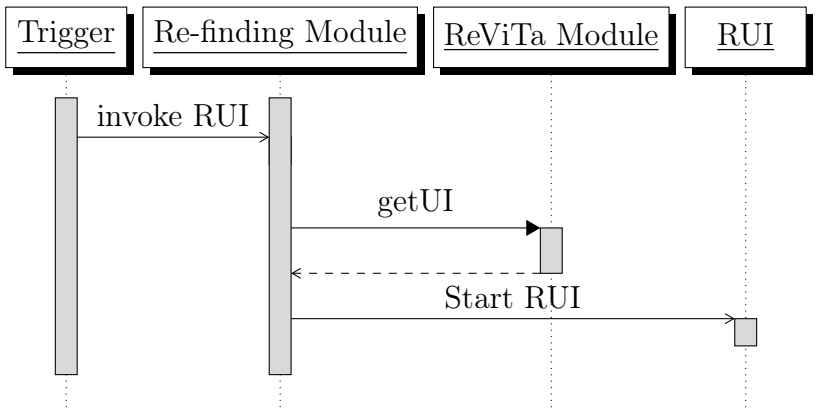

Figure 3.6: Sequence diagram: invoke re-finding user interface

After registering a re-finding user interface, there has to be a mechanism to invoke a re-finding user interface. A re-finding user interface can be triggered using a physical trigger. When the user, for instance, opens the filing cabinet, the filing cabinet will trigger a re-finding user interface to show all the contexts of the file folders in the filing cabinet. Other examples of triggers are waving a hand in front of some paper trays or detecting if a person is standing in front of a filing cabinet. In Figure 3.5(3) the interaction between the trigger and re-finding module is shown. Figure 3.6 shows a sequence diagram of the interactions. The trigger communicates directly with the re-finding module using the endpoint `/invokeRUI/{ruiName}/{recourceID}`. Variable `{ruiName}` represents the re-finding user interface's name, variable `{recourceID}` matches the organisational structure's unique ID that triggers the re-finding user interface. This endpoint understands `GET` requests to trigger the re-finding user interfaces. When extra data is needed to process the re-finding user interface, the endpoint supports `POST` requests. An example of extra data is a time span selected by the user. When a trigger sends a request to this endpoint a lot of things happen before the actual re-finding user interface is invoked. First of all, the re-finding module will check if the organisational structure `{recourceID}` is registered with the re-finding user interface `{ruiName}`. Thus, the re-finding module will start with querying

ReViTa and ask the re-finding user interfaces of that organisational structure. Next the re-finding user interface is searched in the list returned from ReViTa. The interaction between the re-finding module and ReViTa is shown in Figure 3.5(2). If the re-finding user interface is found and thus supported by the organisational structure, a request to `/{ruiName}/start/{recourceID}` is executed to invoke the existing re-finding user interface. Every re-finding user interface is required for at least offer this endpoint. This interaction is shown in Figure 3.5(4). What happens now depends on the invoked re-finding user interface. Every implemented re-finding user interface is discussed in detail in the next chapters.

Providing Additional Data to the Re-finding User Interface

Another function the re-finding module will support is providing some endpoints to allow the re-finding user interfaces to get data from ReViTa. This data is needed to execute the re-finding user interface. A re-finding user interface, for example, needs to know all the contexts of the file folders in a filing cabinet. To support this, the re-finding module will provide a number of endpoints that give access to this data. In general context, content, time and location objects can be requested. The data flow for these actions starts in the re-finding user interface, travels to the re-finding module as shown in Figure 3.5(4) and is processed in the re-finding module. The re-finding module can get data from ReViTa as shown in Figure 3.5(2). To finish, the data is sent back to the re-finding user interface. Figure 3.7 gives a sequential overview of the requests.

First of all, some re-finding user interfaces need to know which contexts are stored in an organisational structure. The sequence diagram is provided in Figure 3.7, case 1. The re-finding user interface can access this data by sending a `GET` request to the endpoint `/context/{resourceID}`. Variable `{resourceID}` represents the unique identifier of the organisational structure that is augmented. A request to ReViTa will get all the contexts of this organisational structure. In response to the initial request, a JSON encoded object is returned. This object consists of an array of the child objects of the initial requested organisational structure. A child is stored in a JSON object and contains the context of the child as well as the colour matching to this context. The unique identifier of this child is also added to the object. With this JSON object the re-finding user interface can, for instance, augment the file folders in a filing cabinet. An endpoint that looks like the previous one is `/contexts/{resourceID}`. The difference is that this time a `GET` request will generate an array of objects of the children of `{resourceID}`. For every child, an array is added that stores the five most available contexts of the

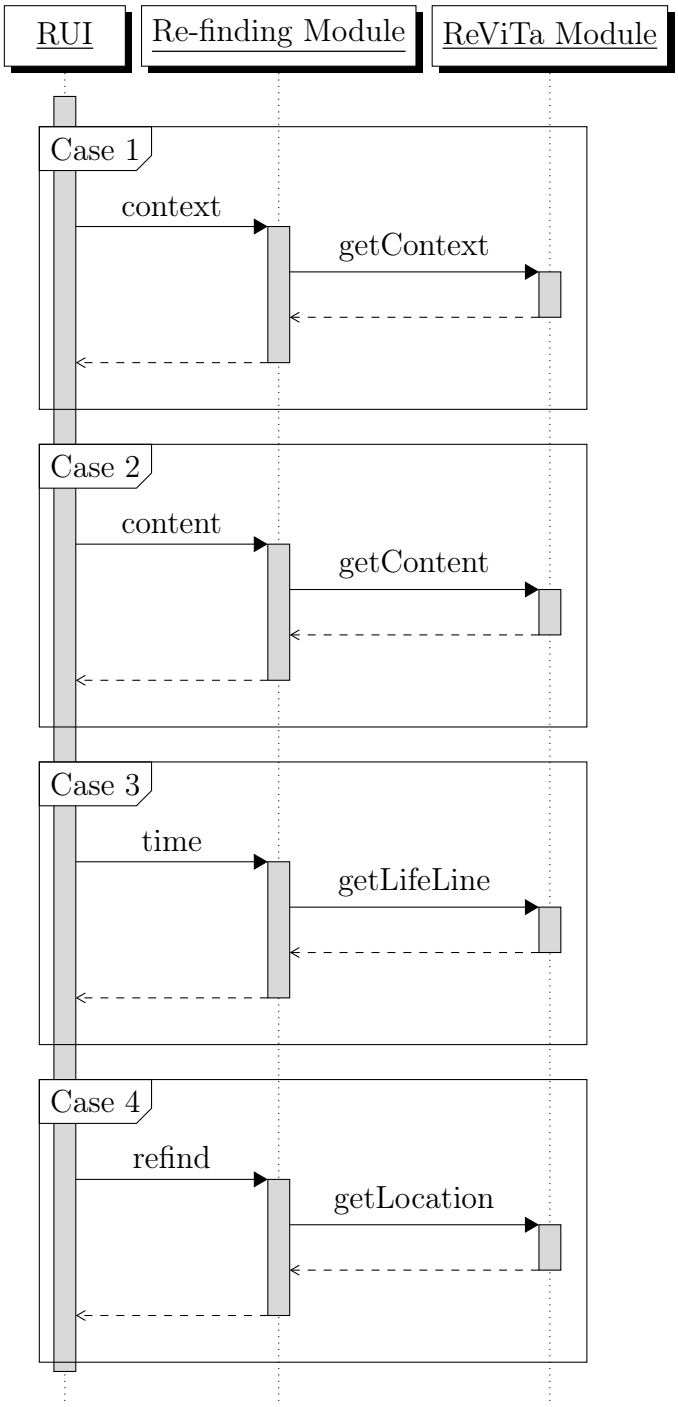

Figure 3.7: Sequence diagram: provide additional data

children of this child.

A third endpoint concerning contexts is provided as `/context/{hwID}/{resourceID}`. This endpoint only supports `GET` requests. The main difference between this endpoint and the previous one is that the return object is filtered on a context. The filter is based on the most relevant context of an organisational structure's element or the context that is available in the re-finding user interface. This time the variable `{hwID}` represents the organisational structure's unique id. The variable `{resourceID}` matches the unique ID of a child element of the organisational structure or a context id. As before, the contexts of the organisational structure are requested from ReViTa. Next, if `{resourceID}` is an ID of a structure element, the context of that element is requested in ReViTa, otherwise the provided context ID is used. To find out if `{resourceID}` is an organisational structure's element or a context, ReViTa is called. The context received from the structure or directly from the variable in the URL is used as a filter. As described in the previous endpoint, the return object will contain an array of child objects. This time only objects where the context matches the previously gathered context is returned. This endpoint is, for instance, used if the user only wants to augment a filing cabinet with a specific context.

At least one re-finding user interface wants to request the contents of, for example, a paper tray. Figure 3.7, case 2, provides a sequential diagram. To support this feature, an endpoint `/content/{hwID}/{resourceID}` is provided that understands `GET` requests. Like the endpoint that filters on context, the variable `{hwID}` represents the organisational structure's unique id, the variable `{resourceID}` matches the unique ID of a child element of the organisational structure. These two ids are used to ask ReViTa what documents are stored in that specific child element. The return of the initial request will contain an array of document objects. Document objects consist of a name, context, etc. A re-finding user interface that will use this endpoint wants to show the user what can be found in, for instance, a paper tray. ReViTa is queried to find out if `{resourceID}` is an organisational structure's element or a context. The returned context is used to filter. The array of document objects will now contain the document objects related to the `{hwID}` and the context. This time `{hwID}`, for instance, matches a unique ID of a pile.

Some of the re-finding user interfaces want to support a time cue. They need data about when documents are accessed, how frequent they are accessed and so one. A sequential representation is given in Figure 3.7, case 3. In general four distinctions are made. Thus, four endpoints provide support to get time data. The first endpoint is responsible for providing data based

on a time span the user selected. `POST` requests are processed by the endpoint `/time/{resourceID}`. The body of the request must contain a JSON object that specifies the start and end date of the timespan. The variable `{resourceID}` matches the unique ID of the organisational structure that is augmented. The unique ID is used to query for the lifeline in ReViTa. Using the selected timespan and lifeline information, the re-finding module will generate an array of child objects and add an intensity parameter depending on how long ago a child was accessed. The most recent accessed file folder, for example, will get a high intensity value whereas older accessed children get a smaller value. This array is then returned to the re-finding user interface. The second endpoint will provide support for requesting when objects are last used. The endpoint `/time/{resourceID}/last` accepts `GET` requests and provides support for this function. As before, `{resourceID}` matches the unique ID of an organisational structure. ReViTa is queried to retrieve the lifeline of the structure elements of this organisational structure. The re-finding module will filter every lifeline information to the last month. Using this data, an array is built. Every object in the array matches a structure element. To every object, an array of access dates is added. This array is sent back to the re-finding user interface. A third endpoint will generate an array that shows how frequent children of an organisational structure are accessed. This function is provided by the endpoint `/time/{resourceID}/frequent` and accepts `GET` requests. `{resourceID}` again matches the unique ID of the organisational structure. A request to ReViTa will get the lifeline for every child. The re-finding module will count in every lifeline when the structure element was accessed on the past month. This information is converted to an array. This array is built out of child objects that contain their unique ID and the access count gathered from ReViTa. When the array is ready, it is sent back to the re-finding user interface. The last endpoint concerning time is `/time/{resourceID}/recent`. This endpoint only accepts `GET` requests and returns an array of objects. The objects are the children of the organisational structure `{resourceID}` and contain an extra parameter intensity that matches when this child is accessed in the last month. More recently accessed objects will get a higher value than older accessed objects. The difference between this endpoint and the first endpoint about time is that this endpoint will always have a fixed time span from today to one month back. Depending on how the re-finding user interface wants to provide a time cue a different endpoint is used.

The last part of information that is required by some re-finding user interfaces is about where a document is located. This is only used in explicit re-finding user interfaces. Figure 3.7, case 4, shows a sequential representation of the requests. Two endpoints provide location information. The first endpoint accepts `GET` requests and is available at `/refind/{resourceID}`. The goal of this endpoint is to return the location of the document specified with `{resourceID}`. ReViTa is queried to get information of the root of `{resourceID}`. This is sent back to the re-finding user interface. The second endpoint will look for the location of, for instance, a file folder based on a specified label. Endpoint `/refind/label/` only accepts `POST` requests. ReViTa is queried to get the location of a file folder matching the label sent in the body of the request. This location is sent back to the re-finding user interface.

Provide Support for Extra Displays

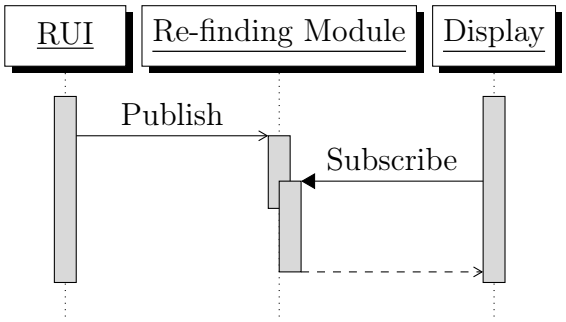

Figure 3.8: Sequence diagram: support for extra displays

A last function that the re-finding module supports is a system to communicate with displays. Some re-finding user interfaces need an extra display to make it easier for the user to show what is going on. The user wants to know, for example, what all the colours in the filing cabinet mean. The re-finding module provides a mechanism to allow easy communication with these displays. Everything is built on the publish-subscribe pattern. Devices subscribe to a topic and get updates if new data is published. This same principle is used for the displays. Displays subscribe to an endpoint on the re-finding module and new data is pushed from the re-finding user interface to the displays using the same endpoint on the re-finding module. Figure 3.5(4) and (5) show the schematic representation of this system. In Figure 3.8 a sequence diagram of the interactions is shown. The publish-subscribe system is available on endpoint `/pubsub/displays`. The displays subscribe by sending a `GET` request to this endpoint. New data is pushed to the displays by

sending a `POST` request. The body of this request will contain the data that is displayed as well as an ID of the display that is used. Every display has a unique ID. On the server, a library called Atmosphere[1] is used to provide this publish-subscribe system. On the client side, the displays show a web application built with CSS, JQuery and JavaScript. The JavaScript client of Atmosphere[2] is used to connect the web application to the re-finding module on the server. Depending on the data received from the publish-subscribe endpoint, different data is shown on the display.

3.5.2 Re-finding User Interfaces

In the next three chapters, the implemented re-finding user interfaces are discussed in detail. The re-finding user interfaces are divided into two categories, implicit re-finding and explicit re-finding. Every implicit re-finding user interface provides one or more re-finding cues. This simulates a more natural interaction compared to explicit re-finding user interfaces and digital search. The explicit user interfaces are provided to allow third-party applications to re-find documents. Table 3.1 gives an overview of the implemented implicit re-finding user interfaces. In total 13 re-finding user interfaces have been implemented. The explicit re-finding user interfaces are shown in Table 3.2. In total five explicit explicit re-finding user interfaces have implemented.

		Context Cue	Spatial Cue	Time Cue
Filing	File folders in a filing cabinet	2		1
	Ring binders on a shelf	2		1
Piling	Piles on a desk	2	1	
Mixing	Paper trays	2		2

Table 3.1: Overview of implemented implicit re-finding user interfaces

Filing	Filing cabinet	Label search and highlight file folder
	Ring binders	Highlight ring binder
Piling	Piles	Show pile and space in pile
Mixing	Paper Trays	Highlight tray

Table 3.2: Overview of implemented explicit re-finding user interfaces

[1] `http://async-io.org/`
[2] `https://github.com/Atmosphere/atmosphere-javascript`

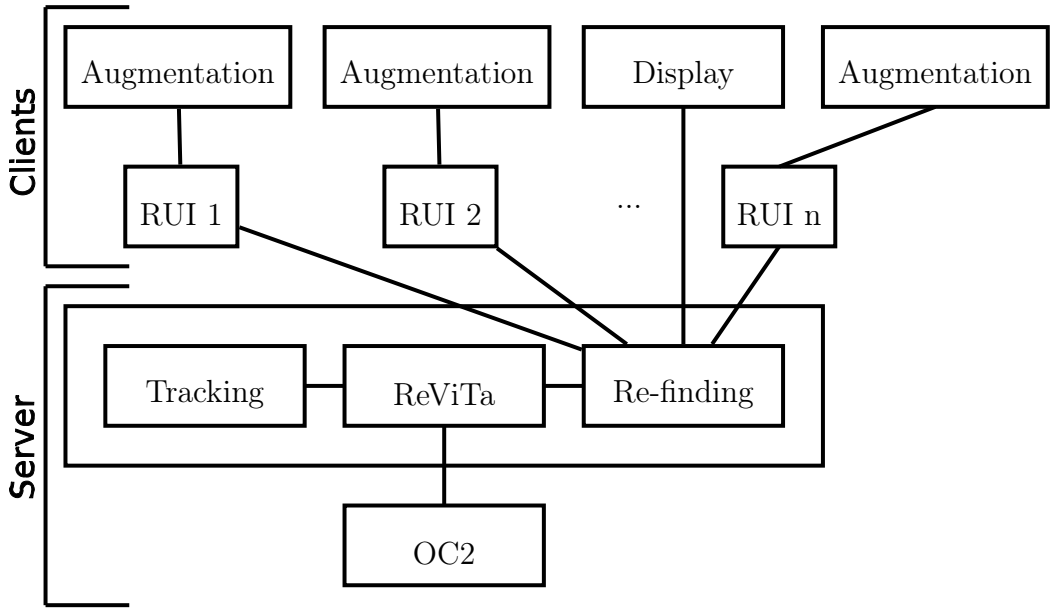

Figure 3.9: KOR Architecture

The architecture for these re-finding user interfaces will contain the server as well as potential clients. In Figure 3.9, a schematic representation of the global architecture is shown. The previously described server is shown as well as some clients. The server is accessible through a RESTful interface described before. Apart from the server some clients are shown. Three re-finding user interfaces are drawn that communicate with the server. They each communicate with hardware that augments the physical world. Also, a display is shown. Because of the way the server is built, new re-finding user interfaces and displays can be added.

4

Re-finding of File Folders

The first organisational strategy that is augmented with re-finding user interfaces will focus on the filing organisational strategy. As a reminder, the definition of filing is repeated. Every element of a file structure is labelled and ordered in a specific way. Groups of files are not necessary labelled or ordered in a specific way. The filing organisational structure can trigger a context, a spatial as well as a time cue. In total two setups are built. One setup is augmenting a filing cabinet using some LEDs and buttons. The other setup augments ring binders placed on a shelf. In every ring binder, a smartphone is mounted.

Six implicit re-finding user interfaces are implemented as well as two explicit re-finding user interfaces. As shown in Table 4.1, two re-finding user interfaces that augment the filing cabinet will provide a context cue. When the user opens the filing cabinet, all the file folders will show a colour matching the file folder's most used context. The user is also able to push a button of a file folder to filter on the context of the selected file folder. A third re-finding user interface will provide a time cue on the filing cabinet. The user can select a timespan and the file folders are coloured from bright to dark. The brighter the colour, the more the file folder has been accessed in the selected time span. To augment ring binders on a shelf, three re-finding user interfaces are implemented, two of them providing a context cue. The first re-finding user interface will show on every ring binder a stacked column

chart of the five most available contexts in the ring binder. The re-finding user interface will allow the user to select one of those contexts and filter the ring binders containing that same context. To provide a time cue, a third re-finding user interface for the ring-binders is built. The user can push a button to show on every ring binder a calendar when the ring binder was accessed.

On top of the implicit re-finding user interfaces, two explicit re-finding user interfaces are implemented. Table 4.2 gives an overview of the explicit re-finding user interfaces. The first explicit re-finding user interface will allow the user to do a label search on the file folders. The second explicit re-finding user interface will provide support for third-party applications to re-find a file folder. An example of a third-party application is a visualisation of the links between physical file folders.

		Context Cue	Spatial Cue	Time Cue
Filing	File folders in a filing cabinet	2		1
	Ring binders on a shelf	2		1
Piling	Piles on a desk	2	1	
Mixing	Paper trays	2		2

Table 4.1: Overview of implemented implicit re-finding user interfaces

Filing	Filing cabinet Ring binders	Label search and highlight file folder Highlight ring binder
Piling	Piles	Show pile and space in pile
Mixing	Paper Trays	Highlight tray

Table 4.2: Overview of implemented explicit re-finding user interfaces

In the next sections, the setups, as well as the re-finding user interfaces, are discussed in detail. Before we continue, we first would like to introduce you to someone. Alison is a biologist and during her excursions she gathers a lot of data from experiments and measurements. This data needs to be processed and analysed later on. Because of the amount of data that is generated, it is important that Alison organises the data well during measurement and when she is back from an excursion. At the same time, Alison is also planning new excursions depending on results of previous research and different locations.

4.1 Setup One: File Folders in a Filing Cabinet

This setup uses a filing cabinet with file folders. Every file folder in the filing cabinet is extended with an RGB LED and a button. In total 30 file folders are augmented. To allow more interactions with the file folders, a tablet is added to the setup. On this tablet, the system can show additional information such as the link of a colour emitted by the LEDs in relation to the context or the contents of a file folder. Also, a micro switch is mounted on the filing cabinet to detect when the user opens or closes the filing cabinet. The setup is shown in Figure 4.1.

Figure 4.1: Filing cabinet setup

Referring back to the system presented by Jervis, SOPHYA, [22]. Jervis replaced the conventional vertical file folders with lateral file folders to solve slow hardware problems. In SPOHYA V2.0, he was able to change the hardware even more to allow detailed location detection of a container [23]. Changing from conventional vertical file folders with lateral file folders is a drastic change because lots of offices still have these vertical file folders with hooks. In the setup presented here, vertical file folders are used and provide support for detailed location of a file folder. Nevertheless, there are still some problems. First of all because of hardware and space limitations, only 10 file folders per row are augmented. Second of all, the LEDs and buttons have to be placed in a fixed position, this converts the filing system to a fixed system. As a result, the file folders have to stay in the order they are placed in at the beginning. Advantages are the re-use of conventional vertical file folders with the option to locate a file folder and provide fast hardware response.

As described before, the file folders are extended with an RGB LED and a button. Also, a micro switch is mounted to the filing cabinet. To control the LEDs and react on a pressed button, some extra hardware is needed. A Raspberry Pi is used to translate all the requests from a RESTful interface to the correct protocol to talk to the LEDs. The Raspberry Pi is also responsible to react on the events from the buttons and micro switch. The LEDs are daisy chained to each other to form one bus system. This bus system is connected to the Raspberry Pi with SPI (Serial Peripheral Interface) as can be seen in Figure 4.2(1). Because the Raspberry Pi does not have enough I/O pins, the buttons are connected to two I/O bus extenders. Figure 4.2 (2) and (3) show the bus extenders in the schematic. Every button is added to one of those bus extenders. To keep the schematic clear, only one button is drawn. This button is shown in Figure 4.2(4). The same principle is used for the other 29 buttons. The bus extenders are connected to the Raspberry Pi via I^2C. The I^2C-bus of the Raspberry Pi works with 3.3V and the bus extender with 5V thus a logic level converter is used to convert the bus levels as shown in Figure 4.2(5). The micro switch, that is responsible to detect if a user opens the filing cabinet, is connected to one of the I/O's of the Raspberry Pi. The micro switch is drawn in Figure 4.2(6). Figure 4.3 gives an overview of the built prototype, LEDs, buttons and micoswitch. Furthermore, the Raspberry Pi runs two python applications. One is responsible to react on the events of the buttons placed on the file folders and the switch to detect the filing cabinet. Another application runs a web server. This web server will provide a RESTful interface to communicate with the LEDs. This same web server is also used for the ring-binders in the next section.

To detect if a button on one of the file folders is pressed, the bus extenders are monitored with a while loop. A library is provided by ABElectronics[1] to query the bus extenders. Every button on its own can send a trigger to the re-finding module on the server. In this same while loop, one of the GPIO pins of the Raspberry Pi is monitored to check if the filing cabinet is opened or closed. Depending on the state of the GPIO pin, a trigger is sent to the re-finding module on the server or the LEDs are turned off.

To provide a web server to the Raspberry Pi the python library Bottle[2] is used. This library allows easy definition of endpoints. An endpoint `/leds` will accept `PUT` requests with a JSON object to control the LEDs. This object must contain an ID and colour for every LED that has to change colour. Internally, the LEDs are controlled with an example code provided

[1] `https://github.com/abelectronicsuk/ABElectronics_Python_Libraries/tree/master/IOPi`

[2] `http://bottlepy.org/docs/dev/index.html`

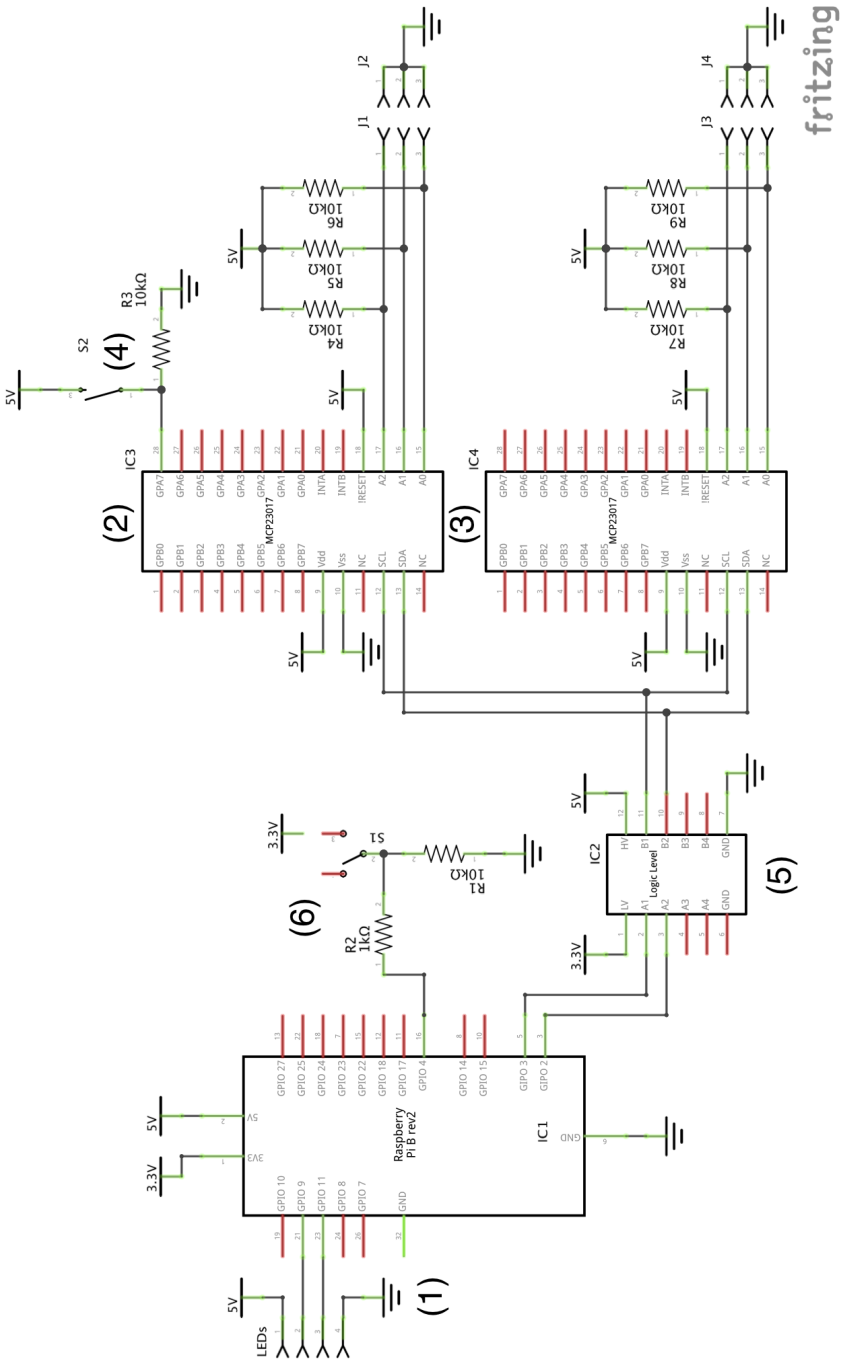

Figure 4.2: File folders hardware schema

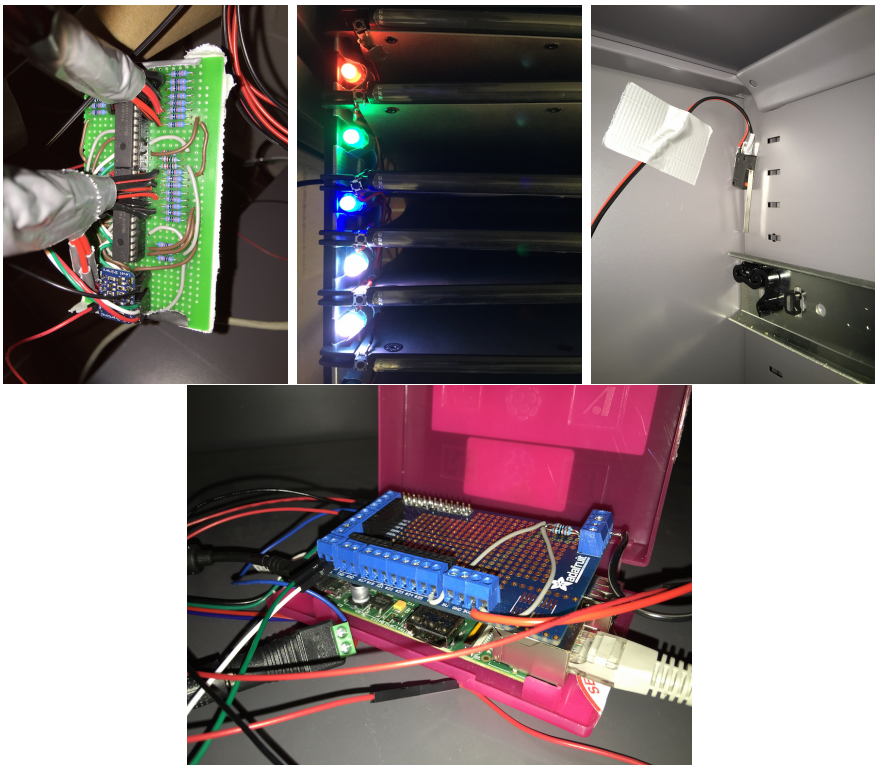

Figure 4.3: Prototype of file folders hardware

by Adafruit[3]. For every LED a colour has to be specified in an array. That array is sent over SPI to the LEDs.

Some re-finding user interfaces need an extra display to show detailed information. To provide this function, a tablet is placed near the filing cabinet. How this tablet will connect to the system was explained in section 3.5.1.

4.1.1 Implicit - File Folders that Show Contexts

Type: Context Cue

Alison wants to analyse data from one of the previous excursions. She knows that the excursion is called "South-Africa Summer 2014". She uses the filing cabinet in her office to archive all measured data and other related documents of the past excursions. When Alison wants to re-find the measurements, she opens the filing cabinet. Colours next to the file folders will show to what context most of the documents stored in the file folder belong. Because

[3] `https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code/blob/master/Adafruit_LEDpixels/Adafruit_LEDpixels.py`

this filing cabinet contains only file folders related to previous excursions, the tablet gives an overview of the excursions linked to the colour that are available in the filing cabinet. Alison knows that she is looking for the measurement data of "South-Africa Summer 2014". The tablet links this context to the colour green. Alison now knows which file folders contain documents of the excursion from last summer. In Figure 4.4 a prototype of the context view on the tablet is shown.

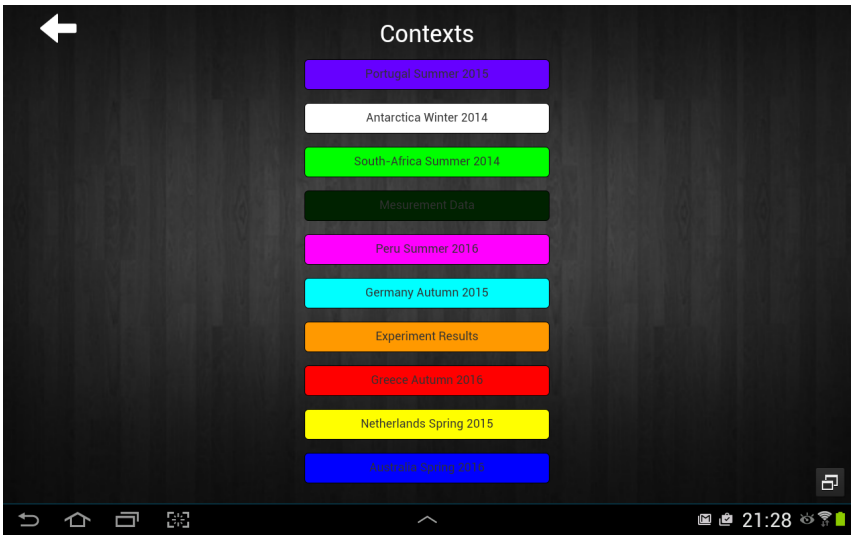

Figure 4.4: Context info on tablet

Implementation

When Alison approaches the filing cabinet and opens the filing cabinet, the Raspberry Pi will detect this because of the micro switch mounted in the filing cabinet. The Raspberry Pi will send a `GET` request to the `/invokeRUI/filesshowcontext/{resourceID}` endpoint on the re-finding module on the server. `{resourceID}` represents the unique ID of the filing cabinet. The re-finding module will check with ReViTa if this `{resourceID}` is allowed to execute the re-finding user interface `filesshowcontext`. If everything is fine after this check, the re-finding user interface `filesshowcontext` is started by a request from the re-finding module on the server to the re-finding user interface located at `/filesshowcontext/start/{resourceID}`.

The re-finding user interface must know what the contexts of `{resourceID}` are. This is done with `GET` request to `/context/{resourceID}` on the re-finding module of the server. When the re-finding user interface knows all the contexts of `{resourceID}`, a JSON object containing the contexts is

sent to the Raspberry Pi in the filing cabinet. A `PUT` request to `/leds/` is used for this. At the same time, that JSON object is sent to the endpoint `/pubsub/displays` on the re-finding module on the server to allow the tablet to show a list of contexts.

4.1.2 Implicit - Filter on Contexts

Type: Context Cue

Because it is sometimes easier to filter a specific context in the filing cabinet, Alison can push the button of a file folder that matches the context "South-Africa Summer 2014". As long as she presses and holds the button, only file folders belonging to "South-Africa Summer 2014" will show their green colour. This makes it easier to locate file folders of a specific context, in this case "South-Africa Summer 2014". When Alison releases the button, all the file folders are augmented again with the context colour of the corresponding file folder.

Implementation

When Alison presses and holds one of the buttons on the file folders, the Raspberry Pi sends a `GET` request to `/invokeRUI/filesshowfilteredcontext/{hwID}/{resourceID}` provided by the re-finding module on the server. This time `{resourceID}` is the ID corresponding with the file folder. `{hwID}` matches the organisational structure. As before ReViTa is checked to be sure that the user interface `filesshowfilteredcontext` may be invoked on this type of structure. The actual re-finding user interface is started with a `GET` request to `/filesshowfilteredcontext/start/{hwID}/{resourceID}`.

The re-finding user interface needs a list of child elements of the parent object of `{resourceID}` provided by `{hwID}`. This list can only contain objects of the same context matching the context of `{resourceID}`. The required list can be requested from the re-finding module on the server using endpoint `/context/{hwID}/{resourceID}`. The returned object is sent to the Raspberry Pi in the filing cabinet using a `PUT` request on endpoint `/leds/` on the Raspberry Pi. This will light up the LEDs of only one context.

When Alison releases the button, a `GET` request is sent to `/invokeRUI/filesshowcontext/{resourceID}` and the user interface `filesshowcontexts` is started as described in section 4.1.1.

4.1.3 Implicit - Timespan Selection

Type: Time Cue

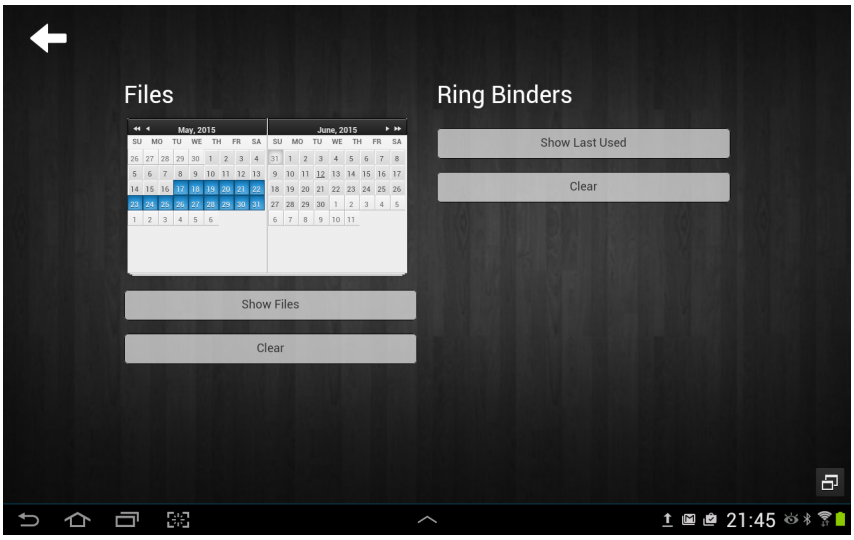

Figure 4.5: Time span selection using tablet

Alison wants to compare the data from the excursion of last summer with data of older excursions but she does not exactly remember the name of those excursions. The only thing she remembers is that she worked with the data in the spring of 2012. A calendar on the tablet near the filing cabinet allows her to select a time span, in this case from the 20th of March to the 21st of June 2012. File folders in the filing cabinet will show a colour linked to how long ago they were accessed in that time span. File folders that are not accessed will not show any colour. File folders that are accessed more recently will have a very bright colour. Older accessed file folders will have a darker colour. The interface shown on the tablet is pictured in Figure 4.5.

Implementation

As described before in section 3.5.1 the displays that are placed near an organisational structure are implemented using CSS, JQuery and JavaScript. The tablet placed close to the filing cabinet will permanently show a button to access a calendar. The calendar is constructed using the Kalendae library[4]. When Alison selects a time span, she has to push on the `ShowFiles` button to start the augmentation. The `ShowFiles` button will send a `POST` request to `/invokeRUI/filestimeselection/{resourceID}` available on the re-finding module of the server. `{resourceID}` will match the ID of the filing cabinet. A JSON object containing the selected time span will we send as body of this request. After a check in with ReViTa, the re-finding

[4] `https://github.com/ChiperSoft/Kalendae`

module will start the re-finding user interface by sending a `POST` request to `/filestimeselection/start/{resourceID}`, the body contains the previous received JSON object.

The re-finding user interface will start with a `POST` request to the re-finding module on the server at endpoint `/time/{resourceID}`. The response to this request will contain an array of objects containing an intensity parameter. This parameter will have an influence on the brightness of the colour emitted by the LED later on. A `PUT` request to endpoint `/leds/` on the Raspberry Pi with this array in the body will light up the LEDs.

4.1.4 Explicit - Label Search

There is one specific file folder Alison wants to incorporate into her research. She knows that the label of that file folder is called "Explored sites in South-Africa". The only problem is that she forgot where the file folder is located in the filing cabinet. Thus, she enters "Explored sites in South-Africa" in the label search field on the tablet. The LED corresponding to the file folder will light up and Alison can easily access the documents in this file folder.

Implementation

To provide support for re-finding a file folder using a label search, the tablet provides a search field. After pushing the search button, a `POST` request is sent to the re-finding module on the re-finding module on the server at endpoint `/invokeRUI/explicitlabelsearch/{resourceID}`. The re-finding user interface is started by sending a request to `/explicitlabelsearch/start/{resourceID}`.

The re-finding user interface will start by getting the ID of the file folder that matches the requested label by sending a `POST` request `/refind/label/` on the re-finding module. The body of the request will contain the label. The re-finding module will send a request to ReViTa. ReViTa will reply with a list of file folders that contain this label. This list is sent back to the re-finding user interface. The re-finding user interface will send this object to the Raspberry Pi in the filing cabinet on endpoint `/leds/`. This will light up the specific LED matching the specific file folder.

4.1.5 Explicit - Spatial Indication

When Alison is working at her computer, she can use a third-party application to visualise all the physical documents stored in the filing cabinet.

A function that can be useful in this third party application is to locate a document in physical space.

Implementation

To allow a third party application to locate a document, the application will have to send a `GET` request to `/invokeRUI/explicitrefind/{resourceID}` on the re-finding module of the server. `{resourceID}` corresponds with the document ID that the third party application wants to locate. The actual re-finding user interface is started with a request to `/explicitrefind/start/{resourceID}`. The re-finding user interface starts with a `GET` request `/refind/{resourceID}` to the re-finding module on the server. The re-finding module will query ReViTa and return the location of the `{resourceID}` to the re-finding user interface. This object is sent to the Raspberry Pi in the filing cabinet on endpoint `/leds/` and lights up the correct LED matching the file folder.

4.2 Setup Two: Ring Binders on a Shelf

The next setup will use a shelf in the bookcase. On this shelf, ring binders are placed. The organisational structure of the shelf represents a filing structure. In total 10 ring binders are augmented. To be able to show more information about a ring binder or a group of ring binders the same tablet as in the previous setup is used. Figure 4.6 shows the ring binders on the shelf.

Part of the label of every ring binder is replaced with a smartphone. On this smartphone, different user interfaces are shown depending on the re-finding user interface. The user can also directly interact with the ring binders. To allow the smartphones to communicate with the re-finding user interfaces, the same Raspberry Pi of the previous setup is used. This time, the Raspberry Pi will orchestrate the communication between the ring binders and the re-finding user interfaces.

To provide easy communication with the smartphones, the Raspberry Pi provides endpoints to allow a publish-subscribe pattern style of communication between the smartphones and re-finding user interfaces. The first endpoint is `/api/listen`. This endpoint is used by the smartphones to subscribe to the data feed. When a `GET` request is sent to this endpoint, no response is sent until there is new data available. The principle used here is called long polling. The second endpoint supports `PUT` requests and is available at `/gsms/`. When a re-finding user interface sends data to this endpoint, the body is sent to all the connected smartphones on the `/api/listen`

Figure 4.6: Augmented ring binder

endpoint. This happens because the endpoints `/gsms/` and `/api/listen` are internally connected using a socket. As a result, the body is sent to all the smartphones. Socket functionality is provided by the pyZMQ library[5]. The implementation of the publish-subscribe system is implemented without any other library. On the smartphones, a web page is shown that is provided by the Raspberry Pi. This web page is built with CSS, JQuery and JavaScript. Because the ring binder has to be uniquely identified, a query string in the URL is used to pass this ID to the web page.

The tablet is implemented the same way as described in section 3.5.1. Depending on the re-finding user interface, specific information is shown to the user.

4.2.1 Implicit - Show Contexts

Type: Context Cue

[5] `http://zeromq.org/`

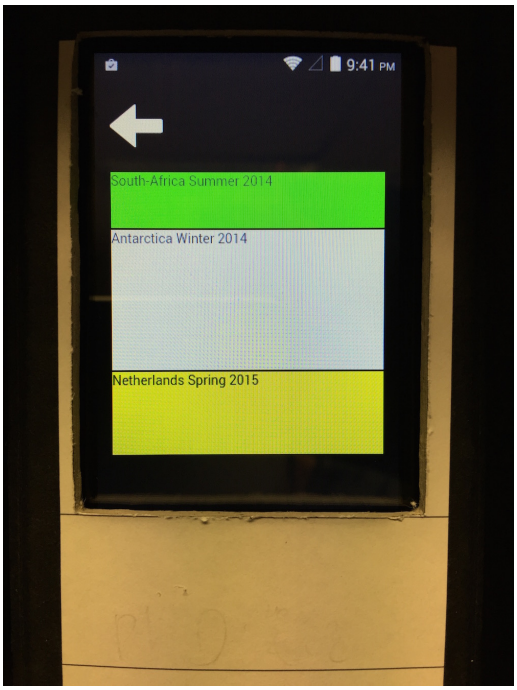

Figure 4.7: Context info on ring binders and tablet

Alison uses the ring binders to keep track of the animals she encountered in the past. Every ring binder represents one animal family. Inside the ring binder, details of the different animals are stored. To find out which animals she encountered the most during an excursion, Alison could access every ring binder and count the animals. Nevertheless, this will take a long time. Instead of doing this, she can walk up to the filing cabinet and just have a look a the information shown on the ring binders. The ring binders will show the five most common contexts available in the ring binders in a stacked column chart. The height of a context depends on the amount of documents available in the ring binder that have this context. The more a colour is represented on a ring binder, the more animals of this animal family Alison encountered during the matching excursion. In this example, the previous excursions are the contexts. Figure 4.7 shows the user interface on the ring binders in action.

Implementation

When Alison walks up to the filing cabinet, a `GET` request is sent to the endpoint `/ringbindersshowcontexts/start/{resourceID}` on the re-finding module on the server. The variable `{resourceID}` represents the unique ID of the ring binders organisational structure. The re-finding module will check if `{resourceID}` can execute the re-finding user interface `ringbindersshowcontexts`. If `{resourceID}` is allowed to execute this re-finding user interface, a `GET` request is sent to `/ringbindersshowcontexts/start/{resourceID}`. This will start the user interface.

To allow the re-finding user interface to work, contexts of the ring binders of the shelf are needed. The re-finding user interface will get this information by sending a `GET` request to the `/contexts/{resourceID}` endpoint on the re-finding module on the server. It is important to note that this time the `/contexts/{resourceID}` endpoint is used rather than `/context/{resourceID}`. The difference is that `/contexts/{resourceID}` will return an array of objects. Every object represents a ring binder and contains an array. In this array, the five most available contexts are stored. The response object is sent to `/gsms/` on the Raspberry Pi in the filing cabinet using a `PUT` request. This will show the contexts on the ring binders in a stacked column chart.

4.2.2 Implicit - Context Selection

Type: Context Cue

Alison wants to research in detail the animals she encountered during her last excursion to South-Africa. At least one of the ring-binders is showing a yellow colour in the stacked column chart matching the "South-Africa Summer 2014" context. When Alison touches this context on one of the ring binders, the other ring binders containing this context will show a yellow background colour if they contain documents related to this context.

Implementation

When a specific context is selected on the ring binder, a `GET` request is sent to the re-finding module on the server at endpoint `/invokeRUI/ringbindersshowselectedcontext/{hwID}/{resourceID}`. The variable `{resourceID}` represents the context of the ring binder. `{hwID}` represents the ID of the organisational structure of ring binders. The re-finding module will check in ReViTa if `{resourceID}` is allowed to execute the re-finding user interface `ringbindersshowselectedcontext`. When everything is fine, a `GET` request

to `/ringbindersshowselectedcontext/start/{hwID}/{resourceID}` at the re-finding module on the server will start the re-finding user interface.

The re-finding user interface will start with sending a request to `/context/{hwID}/{resourceID}` on the re-finding module to get all the contexts of the children of the organisational strategy `{hwID}`. The context of the ring binder is passed with the variable `{resourceID}`. This is done to make a difference between an actual resource ID and a context ID. This object returned from the request is sent to the endpoint `/gsms/` on the Raspberry Pi in the filing cabinet to augment the ring binders. The background colour of the smartphones in the ring binders will mirror the selected context colour.

4.2.3 Implicit - Interaction Log

Type: Time Cue

The past month, Alison already started her research about the animals she encountered during her past excursions. To continue where she left off she wants to know when every ring binder was last accessed. A button on the tablet will allow Alison to show a calendar on every ring binder. In this calendar, the days the ring binder was accessed in the last month are marked. Alison can conclude that a ring binder with a lot of marked days was already researched in detail.

Implementation

To start the re-finding user interface, Alison has to push the `ShowLastUsed` button on the tablet. This button will send a GET request to `/invokeRUI/ringbinderstimeselection/{resourceID}` on the re-finding module on the server. The variable `{resourceID}` corresponds with the ID of the ring binders organisational structure. When ReViTa has decided that `ringbinderstimeselection` can be executed `{resourceID}`, the re-finding module will send a `GET` request to `/ringbinderstimeselection/start/{resourceID}`. This will start the re-finding user interface.

The re-finding user interface will start by asking the re-finding module in the server when the ring binders were accessed in the last month. This is done by sending a `GET` request to `/time/{resourceID}/type`. The JSON object received from the request contains an array of objects that represents when this object is last accessed, the actual date is stored. This object is sent to `/gsms/` on the Raspberry Pi in the filing cabinet to augment the ring binders. Every smartphone will show a calendar. In this calendar, the marked days represent when the ring binder was used.

4.2.4 Explicit - Spatial Indication

As before, Alison can ask a third-party application to locate a ring binder based on the unique ID of that ring binder. When she finds a ring binder in an application that represents the ring binders in a virtual way, she can ask to locate the ring binder in the physical world.

Implementation

The implementation is identical to section 4.1.5.

4.3 Conclusion

We presented various re-finding user interfaces which help the user in re-finding activities in filing structures. Using two setups, one containing a filing cabinet and one containing ring binders, six implicit re-finding user interfaces were implemented. The re-finding user interfaces support the context cue or time cue. Also, three explicit re-finding user interfaces were implemented.

5

Documents Lost in a Pile

The next organisational strategy that is augmented using re-finding user interfaces is the piling organisational strategy. Piles are defined as follows, every element in a pile can have a label but this is not required. None of the elements are structured. A pile on its own is never labelled. Sometimes groups of piles can be placed is a specific order but this is not required. Piles can support context and spatial cues. One setup was built to implement three re-finding user interface. The setup consists of a touch table that replaces the desk of the user. As shown in Table 5.1, two implicit re-finding user interfaces support the context cue, the other implicit re-finding user interface supports a context cue as well as a spatial cue. The first implicit re-finding user interface will show the context label next to the pile. The second implicit re-finding user interface will allow the user to interact with the pile and show linked documents. One re-finding user interface that supports both a context and spatial cue will show the user an overview of the documents in the pile. Also, one explicit re-finding user interface was implemented. This re-finding user interface allows third-party applications to locate a document in a pile. An overview of this explicit re-finding user interface is shown in Table 5.2.

		Context Cue	Spatial Cue	Time Cue
Filing	File folders in a filing cabinet	2		1
	Ring binders on a shelf	2		1
Piling	Piles on a desk	2	1	
Mixing	Paper trays	2		2

Table 5.1: Overview of implemented implicit re-finding user interfaces

Filing	Filing cabinet	Label search and highlight file folder
	Ring binders	Highlight ring binder
Piling	Piles	Show pile and space in pile
Mixing	Paper Trays	Highlight tray

Table 5.2: Overview of implemented explicit re-finding user interfaces

5.1 Setup: Piles on a Touch Table

The main component used in this setup is a touch table. The touch table will replace the user's desk. Piles are placed on the touch table and are tracked by using a camera. The touch table consists of a touch panel that allows to detect 32-touch points. An LCD display is mounted underneath the touch panel to show content on the table. The touch table setup is shown in Figure 5.1.

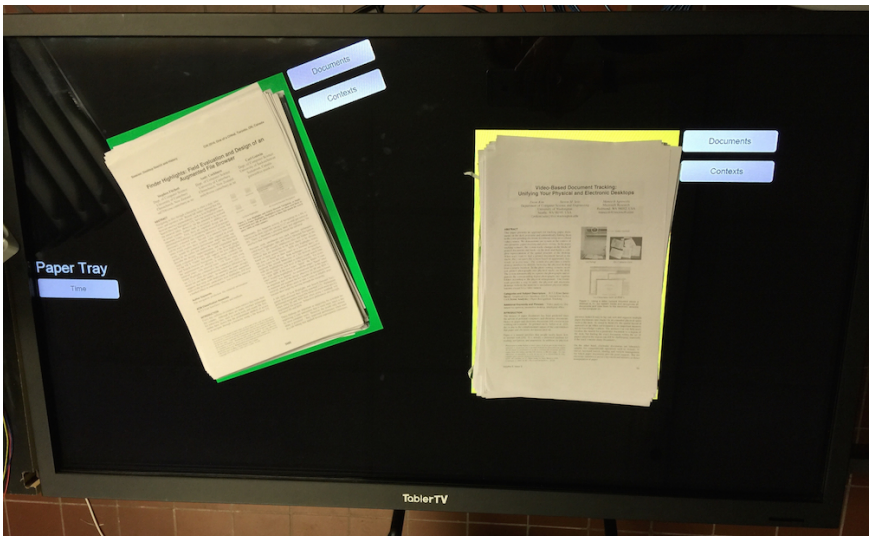

Figure 5.1: Touch table

When the user wants to interact with the piles placed on the touch table, a user interface is displayed on the table. The touch panel is responsible for detecting where the user touches the table.

The touch table on its own is used as a second screen. A driver is responsible to convert the touch points to mouse events. A web application shows the user interface and is connected to the re-finding module of the server as described in section 3.5.1. CSS, JQuery and JavaScript are used to build the interface. The touch table also has a unique ID. This ID is used to only process the object that the touch table needs to show. Depending on the objects received from the re-finding module on the server, different interfaces are shown. These interfaces mostly depend on the re-finding user interface.

5.1.1 Implicit - Show Context Labels

Type: Context Cue

Yesterday night, Alison arrived back from her last excursion to Germany. Before heading home, Alison puts the experiment results and measurement data gathered in Germany on her desk. Two piles are formed next to the three other piles that are already there. Today, she wants to start processing the results of the experiments. Because it was late last night, Alison cannot remember which pile contains the experiment results. A touch on the touch table helps Alison to remember what documents are placed on her desk. The piles show the context labels of the documents that are present in the pile. Figure 5.2 gives an overview of what Alison will see.

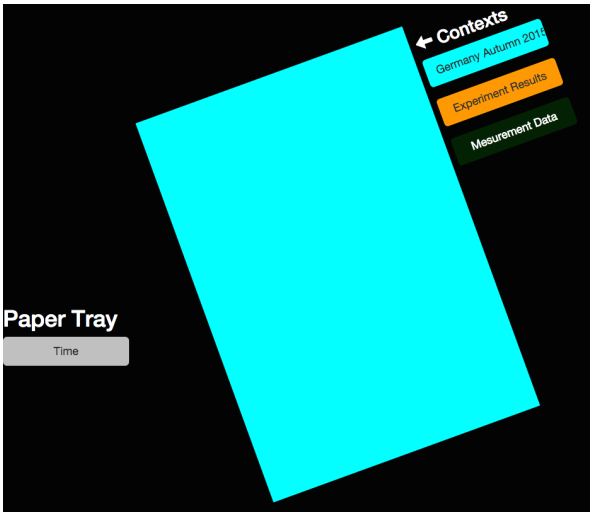

Figure 5.2: Contexts next to piles

Implementation

When Alison touches the touch table, a `GET` request is sent to `/invokeRUI/pilesshowcontextlabel/{resourceID}` offered by the re-finding module at the server. Variable `{resourceID}` represents the unique ID of the organisational structure of piles. The re-finding module will check with ReViTa if `{resourceID}` is allowed to execute this re-finding user interface. When everything is fine, the re-finding user interface is started by sending a `GET` request to `/pilesshowcontextlabel/start/{resourceID}`.

The re-finding user interface needs the context labels to augment the piles. This information is gathered by sending a `GET` request to `/contexts/{resourceID}` on the re-finding module. The object returned from the re-finding module is tagged with the ID of the touch table and sent to the `/pubsub/displays` endpoint on the re-finding module.

When the touch table receives an object from the re-finding module, first of all, the ID is checked. If the ID matches, the object is further processed. In this case, the location information inside the object is used to display the context labels next to the corresponding pile. Apart from the context labels, a command bar is also showed next to every pile. This command bar is used in the re-finding user interfaces described below.

5.1.2 Implicit - Related Documents in Contexts

Type: Context Cue

All the documents of the experiment results and measurement data gathered in Germany are either linked to "Experiment results" or "Measurement data". Every document of the two piles is linked with "Germany". Before Alison starts to process the results of the experiments, Alison wants to find out if she already has documents related to the context "Germany". She can touch the context "Germany" which is shown in the context list next to the pile containing experiment results. A list of documents linked to the context "Germany" is shown. Some of the documents are highlighted to show that they are present in the pile. Other documents are probably archived in the filing cabinet. Figure 5.3 pictures the user interface.

Implementation

When the list of contexts is shown next to a pile using the previous re-finding user interface, Alison is able to touch one of the contexts. This will result in a `GET` request to `/invokeRUI/pilesshowrelateddocs/{hwID}/{resourceID}` on the re-finding module. The variable `{hwID}` represents

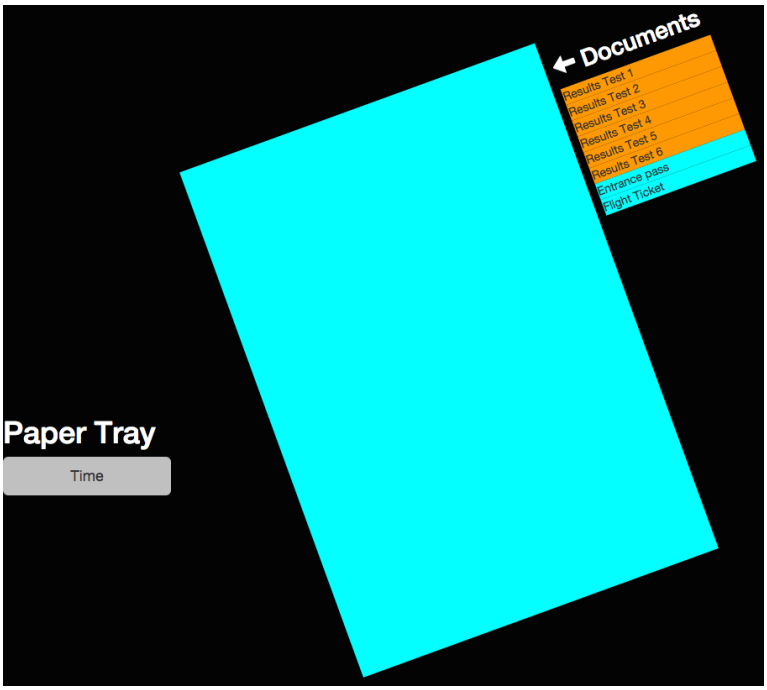

Figure 5.3: Related documents of a pile

the unique ID of the pile. `{resourceID}` represents the unique ID of the selected context. The re-finding module will check ReViTa to know if `{hwID}` is allowed to execute this re-finding user interface. When no problems are detected, the actual re-finding user interface is executed by sending a `GET` request to `/pilesshowrelateddocs/start/{hwID}/{resourceID}`.

The re-finding user interface will start with asking the re-finding module what the related documents of `{hwID}` are depending on the context provided by `{resourceID}`. The endpoint `/content/{hwID}/{resourceID}` that accepts `GET` requests on the re-finding module provides this data. This request will respond with an array of document objects related to the context selected before. When the re-finding interface received this data, the object is tagged with the ID of the touch table. A `POST` request will send the object to the `/pubsub/displays` endpoint on the re-finding module.

When the touch table receives an object, the object is processed if the ID matches. The touch table will show a list of related documents next to the pile that sent the request. Documents stored in the pile are highlighted.

5.1.3 Implicit - Digital Augmentation

Type: Context Cue + Spatial Cue

In the first implicit re-finding user interface, apart from the context labels, a command button is displayed next to the piles. This command button will become useful in the next re-finding user interface. Alison knows that she had a document containing results of a specific experiment she wants to process first. She knows that the document is located in the pile containing the experiment results but she does not know where exactly. To help Alison re-find the document, she can push a button on the command bar to show a digital representation of the pile. The representation will show the order of the documents in the pile. Alison can look in the list for the document she needs and find out more or less where the document is located in the pile. An example of the digital representation of a pile can be found in Figure 5.4.

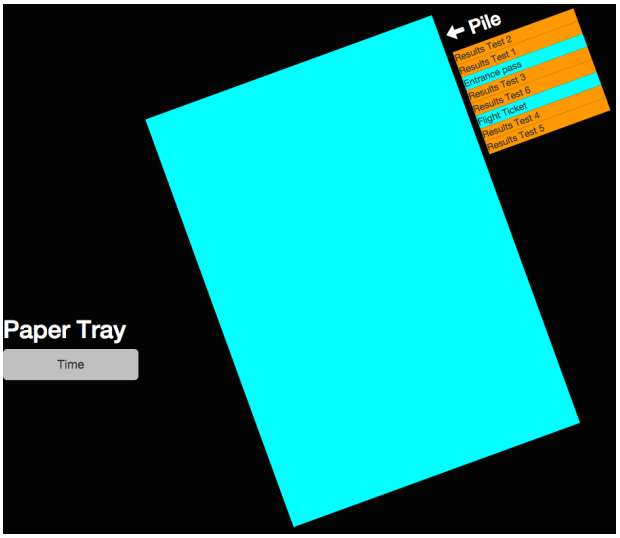

Figure 5.4: Digital representation of the pile

Implementation

When Alison pushes the button on the command bar, a `GET` request is sent to the re-finding module on the server at endpoint `/invokeRUI/pilesshowdigipile/{hwID}/{resourceID}`. ReViTa is checked by the re-finding module to find out if the pile represented by ID `{resourceID}` is allowed to be augmented with this re-finding user interface. Assuming that everything is fine, the re-finding user interface is started by sending a `GET` request to `/pilesshowdigipile/start/{hwID}/{resourceID}`.

As this re-finding user interface needs to know which documents are available in `{resourceID}`, the re-finding module is asked to provide this data. Endpoint `/content/{hwID}/{resourceID}` that accepts `GET` request will return an array of document objects tagged with an order integer. This integer is needed to know in what order the documents are placed on the pile. If this array is received by the re-finding user interface, it is tagged with the ID of the touch table and send to the `/pubsub/displays` endpoint on the re-finding module to finally arrive at the touch table.

The touch table will process this object if the ID matches. A virtual representation in the form of a stacked column chart is displayed next to the pile that triggered the request.

5.1.4 Explicit - Spatial Indication

As before, Alison can ask a third-party application to locate a pile based on the unique ID. The piles can be presented in a virtual way in the third party application. Asking to locate the pile, will highlight the pile.

Implementation

The implementation is identical to section 4.1.5.

5.2 Conclusion

In this chapter re-finding user interfaces were presented that help the user in re-finding activities in piling structures. One setup, containing a touch table, was build to built three implicit re-finding user interfaces. These implicit re-finding user interfaces provide support for the spatial cue and context cue. Also, one explicit re-finding user interface was implemented.

6

Chaotic Paper Trays

The last organisational strategy that is augmented using re-finding user interfaces is the mixing organisational strategy. Mixing represents every structure that is not filing nor piling. The elements can have a label but this is not required. At the same time, there can be an order between the elements but this is also not required. Groups of mixtures do not require any labelling or order but they can be labelled or ordered. Mixtures provide a context cue as well as a time cue. To implement some re-finding user interfaces one setup was built. The setup consists of a stack of paper trays. Four implicit re-finding user interfaces are implemented. As shown in Table 6.1, two re-finding user interfaces implement the context cue, the other two support the time cue. The first implicit re-finding user interface will allow the user to easily see the contexts of the paper trays by waving a hand in front of the paper trays. The second implicit re-finding user interface will allow the user to request the contents of a paper tray by pushing a button on the corresponding paper tray. Another implicit re-finding user interface will allow the user to show the paper trays that are most recently accessed and most frequently. The last implicit re-finding user interface allows the user to find out if a paper tray was assessed in a selected time span. Also, one explicit re-finding user interface was implemented as shown in Table 6.2. This explicit re-finding user interface provides support for third party applications to locate and highlight a paper tray containing the requested document.

		Context Cue	Spatial Cue	Time Cue
Filing	File folders in a filing cabinet	2		1
	Ring binders on a shelf	2		1
Piling	Piles on a desk	2	1	
Mixing	Paper trays	2		2

Table 6.1: Overview of implemented implicit re-finding user interfaces

Filing	Filing cabinet	Label search and highlight file folder
	Ring binders	Highlight ring binder
Piling	Piles	Show pile and space in pile
Mixing	Paper Trays	Highlight tray

Table 6.2: Overview of implemented explicit re-finding user interfaces

6.1 Setup: Paper trays

In this setup, a stack of paper trays is used. In total five trays are augmented. Every tray is extended with a LED strip and a button. To detect if the user waves a hand in front of the paper trays, an ultrasonic range finder is placed in front of the paper trays. This ultrasonic sensor is also used to track documents but requires a camera to identify the document. To give the user some more information about what is shown in most of the re-finding user interfaces, more information is shown on the touch table located near the paper trays. The touch table can show a list of contexts linked to a colour. These colours are emitted by the LEDs in the paper trays. Figure 6.1 shows the setup.

The hardware for this setup consists of a LED strip and button per tray, an ultrasonic range finder in front of the paper trays and a Raspberry Pi. Figure 6.2 gives a schematic representation of the hardware. The Raspberry Pi is responsible for translating the requests of the RESTful interface to the correct protocol to talk to the LEDs. The Raspberry Pi is also responsible to react on the events generated by the buttons mounted on the paper trays. Because the Raspberry Pi does not provide enough GPIO pins that support PWM, extra hardware is added to solve this problem. In total 15 GPIO pins supporting PWM are needed because every LED strip needs data of the red, green and blue colour. This amount of pins is provided using a 16-channel LED controller that is connected to the Raspberry Pi using the I^2C protocol. The LED controller is shown in Figure 6.2(1). LED strips tend to use a lot of energy, more energy than any microcontroller can provide. To solve this

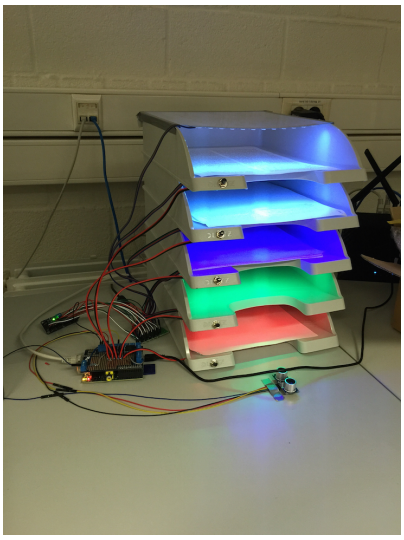

Figure 6.1: Paper tray setup

problem, MOSFETs are used to power the LED strip from a high current power supply. The MOSFETs are shown in Figure 6.2(2). The construction with three MOSFETs and the LED strip as shown in Figure 6.2(2) is repeated four more times. The five buttons mounted on the paper trays are connected to available I/O pins the Raspberry Pi. In Figure 6.2(3) the schematic representation of the buttons is shown. Two I/O pins are needed to connect the ultrasonic range finder to the Raspberry Pi as shown in Figure 6.2(4). The Raspberry Pi (1), LED controller (2), MOSFETs (3), ultrasonic sensor (4) and paper trays with buttons and LED strips of the prototype are shown in Figure 6.3.

The Raspberry Pi is running three python applications, the first one is responsible for providing a web server, the second one will monitor the buttons mounted on the trays. A third application is responsible for querying the ultrasonic sensor. Like the setup described in section 4.1, the web server is responsible for providing a RESTful interface to control the LEDs.

The state of the buttons is read in a while loop. Every time the state of a button is True, a request is sent to the re-finding module of the server.

To measure the distance between the ultrasonic sensor and a possible object, the application will time how long it takes to get a response on one of the two pins, called the ECHO pin, of the ultrasonic sensor when the other pin, called the TRIG pin, is set to high state. With some calculations the distance between the ultrasonic range finder and an object can be calculated based on the time it takes to receive response on the ECHO pin. Exactly knowing the distance between the ultrasonic sensor and the user's hand is

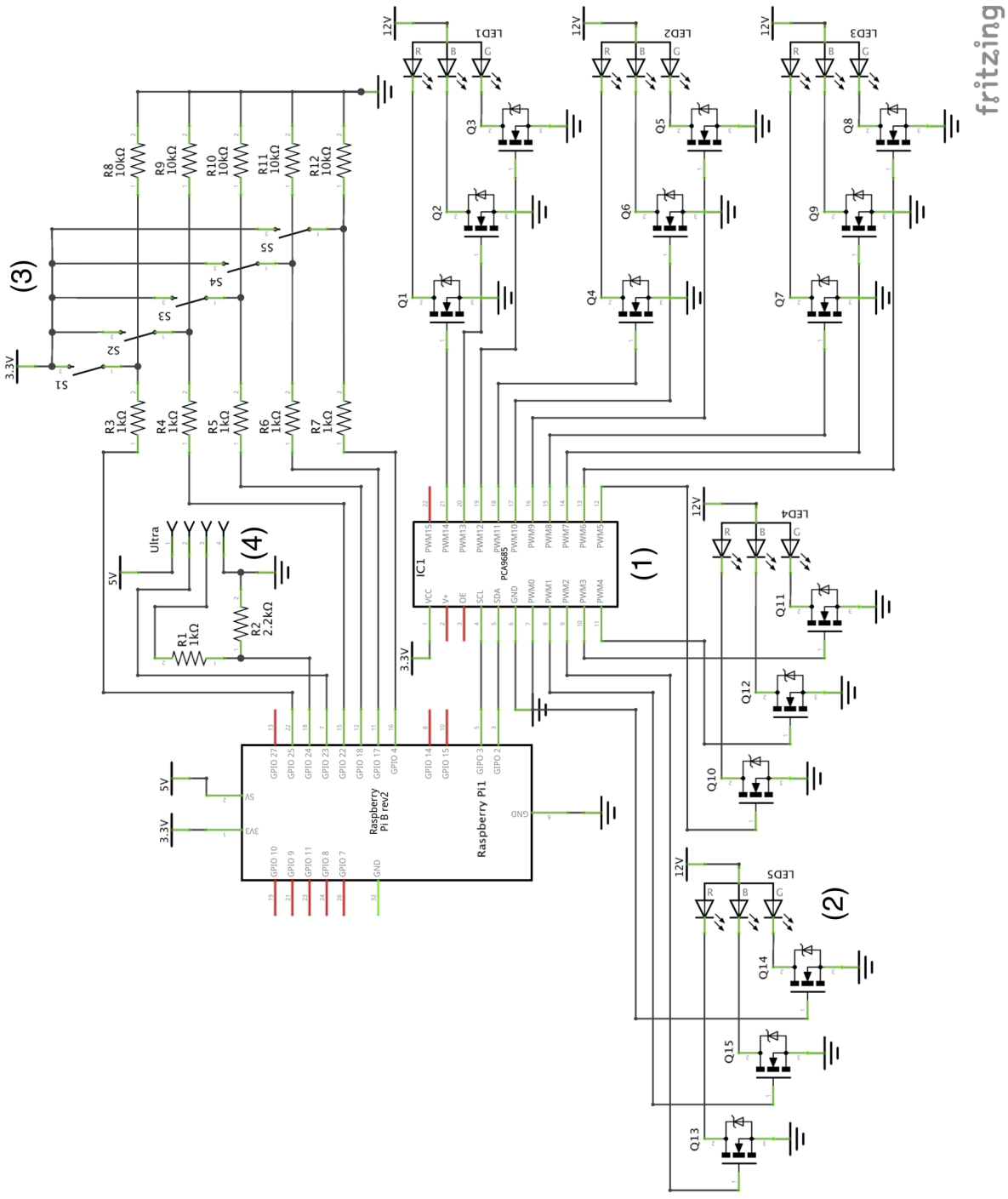

Figure 6.2: Paper trays hardware schema

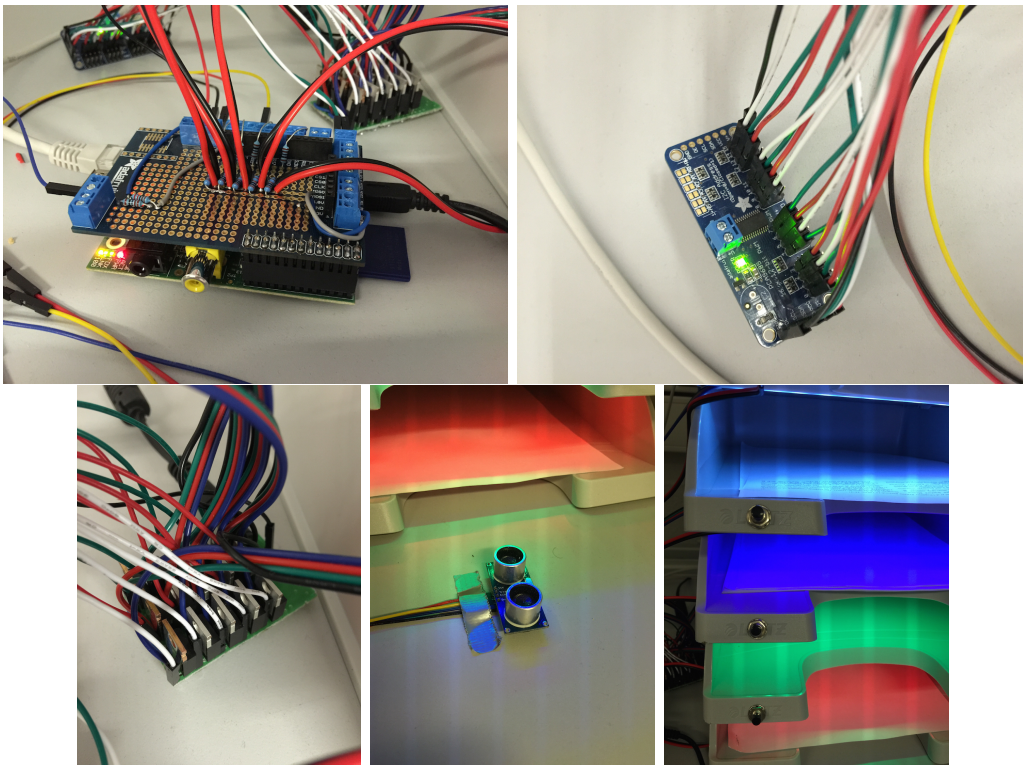

Figure 6.3: Prototype of paper tray hardware

not important to detect if the user is waving a hand in front of the paper trays. Knowing that the distance is smaller than the top of the stack of paper trays is enough to detect that the user is waving a hand in front of the paper trays. This behaviour will result in sending a request to the re-finding module to trigger a re-finding user interface. When tracking in which paper tray a document is placed, the exact distance between the ultrasonic sensor is needed to find out in which tray the document is placed. In combination with a camera, the document can be tracked and linked to a paper tray.

The Bottle framework[1] is used to provide an endpoint to communicate with the LEDs in the paper trays. The endpoint is called `/leds/` and only supports `PUT` requests. When an object is received on this endpoint, the contents of the object are processed to receive the RGB colour information. This information is then sent to the LED controller using a library provided by Adafruit[2]. For every colour, red, green and blue, a different channel of the LED controller is updated. By mixing those three channels, the correct

[1] `http://bottlepy.org/docs/dev/index.html`

[2] `https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code/tree/master/Adafruit_PWM_Servo_Driver`

colour is represented by the LED strip.

The touch table placed near the stack of paper trays is implemented as described in section 3.5.1.

6.1.1 Implicit - Show context

Type: Context Cue

Alison is planning a new excursion to Australia. The excursion will take place in the spring of 2016. As she travels a lot to different destinations, she has to organise a lot. At least two paper trays will contain information about places to stay and tickets that are already booked. The other three trays contain documents of experiments that she will execute during the next excursions. Some of the experiments need more preparation, other experiments are already finished and grouped together in one paper tray. Alison can easily find out which contexts are available in every paper tray by waving her hand in front of the paper trays. Every paper tray shows a context colour. The touch table provides a list of contexts that are linked to a specific context. Alison will see a list containing the contexts "Australia spring 2016", "Peru summer 2016" and "Greece autumn 2016". Figure 6.4 gives an example of the user interface.

Figure 6.4: Contexts information about paper trays

Implementation

When Alison waves her hand in front of the paper trays, a `GET` request is sent to `/invokeRUI/traysshowcontext/{resourceID}` on the re-finding module of the server. Variable `{resourceID}` represents the ID of the stack of paper trays. The re-finding module will check with ReViTa if `{resourceID}` is allowed to execute the `traysshowcontext` re-finding user interface. If everything is fine, a `GET` request to `/traysshowcontext/start/{resourceID}` is initiated to start execution of the re-finding user interface.

The re-finding user interface needs the context of every paper tray. A `GET` request to `/context/{resourceID}` on the re-finding module provides this information. When the re-finding user interface has received the object, the LEDs of the paper trays are lighted by sending a `PUT` request to `/leds/` on the Raspberry Pi. The body of this request contains the object received from the re-finding module. The same object is also sent to the touch table using endpoint `/pubsub/displays` after it was tagged with the ID of the touch table. The touch table will display a list of contexts linked to a colour. After some time, the LEDs in the paper trays will turn off.

6.1.2 Implicit - Show content

Type: Context Cue

Alison wants to be sure that she already booked her flight tickets to Australia. She can start to look for them in the paper tray containing all the tickets. Or she can push the button mounted on the paper tray. Doing this, the touch table will show a list of documents that Alison put in this paper tray. Two documents named "Reservation BRU - MEL" and "Reservation MEL - BRU" show up in the list confirming that the flight tickets to Melbourne Airport are booked. A demonstration of this user interface is shown in Figure 6.5.

Implementation

When Alison pushes the button on one of the paper trays, the Raspberry Pi will detect this and sends a `GET` request to `/invokeRUI/traysshowmoredetails/{hwID}/{resourceID}` provided by the re-finding module on the server. The re-finding module will check with ReViTa to find out if this `{resourceID}` can invoke the re-finding user interface. The re-finding module will send a `GET` request to `/traysshowmoredetails/start/{hwID}/{resourceID}` if everything is fine. This will start the re-finding user interface.

The re-finding user interface will start by getting the content of the paper tray by invoking a `GET` request to `/content/{hwID}/{resourceID}` on the

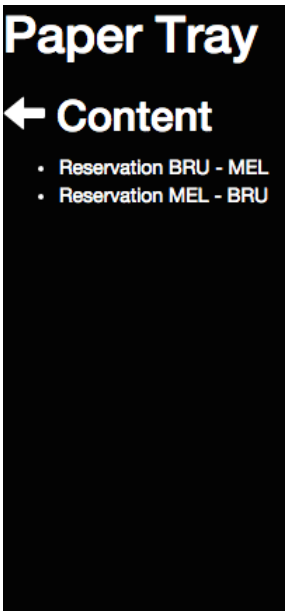

Figure 6.5: Content of a paper tray

re-finding module. The response of this request is an array of document objects physically available in the corresponding paper tray. The response object is tagged with the ID of the touch table and sent to the touch table using a `PUT` request to `/pubsub/displays` on the re-finding module. The touch table will show a list of documents available in the paper tray.

6.1.3 Implicit - Interaction Log

Type: Time Cue

Alison knows that the top two paper trays contain "Unfinished prepared experiments". To find out if these two paper trays are actively accessed, the frequency Alison accessed the paper trays can be requested by pushing a button on the touch table named "Frequently accessed". This will colour the paper trays. A lighter colour means the paper tray is not used a lot, a brighter colour means that the tray is used a lot. When Alison sees that the top two trays are lighted up in very bright colours, she is sure that she is doing whatever it takes to finish all the preparations for the experiments in time. Alison also wants to know which paper tray is last accessed to find out what paper tray is used to process the last "Unfinished prepared experiments". To provide this information Alison pushes on the "Recent accessed" button shown on the touch table. The paper trays will show a

colour. The brightness of every colour is linked to how long ago a paper tray was accessed. A lighter colour means accessed very recently, darker colour means accessed a longer time ago. Figure 6.6 shows the options presented on the touch table.

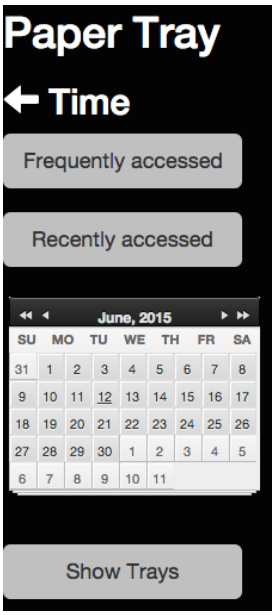

Figure 6.6: Time controls to interact with the paper trays

Implementation

When Alison pushes the "Frequently accessed" button, a `POST` request is sent to `/invokeRUI/mixturestimefrequentlyaccessed/{resourceID}`.
The "Recent accessed" button will do the same but will send `POST` request to `/invokeRUI/mixturestimerecentaccessed/{resourceID}`. Both endpoints are provided by the re-finding module on the server. As before `{resourceID}` represents the ID of the paper trays. The re-finding module will start to check with ReViTa if `{resourceID}` can execute this specific re-finding user interface. When everything is fine a `GET` request to `/mixturestimefrequentlyaccessed/start/{resourceID}` or `/mixturestimerecentaccessed/start/{resourceID}` is initiated to start the corresponding re-finding user interface. Both re-finding user interfaces act the same, they will send the same request to the re-finding module. The only difference is the type used to get the information. The object that contains time data is requested with a `GET` request to `/time/{resourceID}/{type}`. `{type}` is replaced with `frequent` or `recent` depending on the re-finding user

interface. When the re-finding user interface received the requested object containing object information with an extra intensity parameter controlling the brightness of the colour, this object is sent to the Raspberry Pi using the endpoint `/leds/`. This endpoint only understands `PUT` requests. The paper trays will show a colour that is brighter or darker depending on the initially pushed button.

6.1.4 Implicit - Timespan selection

Type: Time Cue

To allow Alison the ability to find out which paper tray was used most recently last week, she can select a time span on the touch table. The paper trays that are accessed most recently in that time span are showing a very bright colour compared to the paper trays that are used the longest time ago. In Figure 6.6 time span selection is shown.

Implementation

Alison is able to select the time span in a calendar shown on the touch table. Pressing the button near the calendar will trigger the re-finding user interface. The calendar is shown using the Kalendae library[3]. When the button is pressed, a `POST` request is sent to the endpoint `/invokeRUI/mixturestimeselection/{resourceID}` located at the re-finding module on the server. `{resourceID}` represents the ID of the paper trays. The body of the request will contain a JSON object representing the selected time span. Before executing the actual re-finding user interface, the re-finding module will check with ReViTa if that `{resourceID}` is allowed to execute the re-finding user interface. If everything is fine, a `POST` request to `/mixturestimeselection/start/{resourceID}` is executed to start the re-finding user interface. The body contains the selected time span. When the re-finding user interface is started, a `POST` request to `/time/{resourceID}` will request an array object containing the time information of the paper trays. Every object contains an intensity parameter that will influence the brightness of the colour shown by the LEDs. This object is sent to the `/leds/` endpoint on the Raspberry Pi with a `PUT` request. The LEDs will light up with a brightness depending on how long ago the paper trays were accessed in the selected time span.

[3] `https://github.com/ChiperSoft/Kalendae`

6.1.5 Explicit - Spatial Indication

As before, Alison can ask a third-party application to locate a document in a paper tray based on the unique ID the document. A third party application can ask to highlight the paper tray where the document is located.

Implementation

The implementation is identical to section 4.1.5.

6.2 Conclusion

This chapter elaborated different re-finding user interfaces that provide support for re-finding documents in mixing structures. One setup was built to implement four implicit re-finding user interfaces. The context cue, as well as time cue, are supported by these re-finding user interfaces. Also, one explicit re-finding user interface was implemented.

7 Conclusion and Future Work

7.1 Conclusion

The goal of this thesis was to develop a rapid prototyping framework that is based on the KOR principles and provides re-finding support on physical documents using a PIM system. Re-finding is supported using implicit re-finding user interfaces. The user has to be able to re-find documents in a natural way. Organisational strategies filing, piling and mixing are used as well as the re-finding cues context, spatial and time. Providing support for re-finding activities is accomplished by building re-finding user interfaces that support specific re-finding cues. In addition, the framework has to be extensible so that it is easy to add new re-finding user interfaces only by registering them in the system. Finally, third party applications should be able to easily communicate with the framework to request the framework to re-find a document in an explicit way.

The framework is divided into three modules, a tracking module, a module called ReViTa that communicates with the OC2 PIM system and a re-finding module. The tracking module will store data in the OC2 by using the ReViTa module. ReViTa is responsible for communicating with OC2. Apart from this, ReViTa also maps the organisational structures as well as extra information that has to be kept in OC2 to support the re-finding cues. The re-finding module reads the data via ReViTa from OC2 and communicates

with the re-finding user interfaces. Due to the extensiveness of the framework, this thesis mainly focuses on the re-finding module and extensions in ReViTa as well as the re-finding user interfaces.

Extensions in the ReViTa module The ReViTa module is extended to support organisational structures and re-finding cues. ReViTa also allows access to OC2 if the re-finding module needs more data.

Implemented re-finding module The re-finding module provides a RESTful interface that allows the re-finding user interfaces to get the required data. This data is used to send to the hardware responsible for augmenting the physical world. At the same time, the hardware offers a RESTful interface to allow easy access to functionality from inside a re-finding user interface. The re-finding module is also responsible to check if a re-finding user interface can be connected to an organisational structure. The reason for this is that not every organisational structure does support every re-finding cue. On the other hand, the re-finding module is also responsible for checking if a re-finding user interface can be executed on a resource.

Proof of concept re-finding user interfaces As a proof of concept we have developed 13 implicit and 4 explicit re-finding user interfaces. Based on the organisational strategies re-finding user interfaces are implemented that support specific re-finding cues. The implicit re-finding user interfaces provide a natural interaction.

Systems like DigitalDesk [42], DocuDesk [11] and PaperSpace [49] show that augmenting physical documents have their use. Because we do not know what will happen with physical paper [47], systems that combine physical as well as digital documents are important to allow them to coexist. The systems built by Jervis [46] [22] [23] [24] focus even more on the combination of physical and digital documents. Since none of the systems are integrated with a unified PIM system like Haystack or Semantic Desktop, the three problems, information overload, classification problem and information fragmentation, still exist. In addition, none of the systems implement re-finding based on re-finding cues. As proven by research, the spatial cue is not the main re-finding cue [51]. Context and time are also important cues to re-find documents. The existing PIM systems force the user to start a digital search to re-find a document, there is some navigation needed to find the document they want without the help of an augmented user interface in the physical world. Users also sometimes do not know what they are exactly looking for [50]. This is not natural [50] and the user has to change between digital and

physical space. The solutions available at the moment are standalone applications and only support spatial indications of where a document is located. To overcome information fragmentation, a PIM system is used.

The system we present uses a central repository PIM system to store all the metadata of the documents. Using some implicit re-finding user interfaces that support the re-finding cues, context cue, spatial cue and time cue, re-finding activities are supported. Explicit re-finding is supported in a way that the user does not has to start with digital search but rather using an external third-party application.

The framework and architecture presented in this thesis takes into account the problems that exist with current augmented user interfaces and PIM systems. The goal was to create a system that helps the user to re-find physical documents faster by augmenting organisational strategies that trigger not only the spatial re-finding cue but also the context cue and time cue. As discussed before, the rapid-prototyping framework presented is based on a unified PIM system and thus solves some of the problems that are existing in the PIM domain. Because of implicit re-finding user interfaces, the user is able to re-find documents in a more natural way rather than first start a digital search to look for a document. The framework, consisting of three modules, and the architecture of the re-finding user interfaces is built in such a way that there is always room for extending the framework as well as the system in general with new features or re-finding user interfaces.

7.2 Future Work

An evaluation of all the user interfaces is useful to find out which re-finding user interface is actually useful to use and which re-finding user interface is not. It looks also useful to study in which ways a file structure, pile structure and mixing structure can exist. In this thesis only a filing cabinet, ring binders, paper trays and piles are used but there are other ways possible. For example, vertical filing spaces or other types of paper trays. These structures can then be augmented with specific re-finding user interfaces and re-evaluated. Another point of improvement is the hardware. To augment the filing cabinet, every file folder is extended with a LED and a button. This is a good solution but the way it is built at the moment is that it generates a lot of wires. The same problem exists with the augmentation of the paper trays. Converting the hardware to SMD packaging can already reduce the size a lot. The best solution would be to integrate the hardware inside a file folder or paper tray so that it looks like the hardware was built like this. Apart from using a tablet near the filing cabinet or a touch table, other ways

of displays can be researched. Projection on the filing cabinet or ring binders can also be a solution to augment file folders in the filing cabinet. The implemented re-finding user interfaces can also be finalised more. Because of the smartphones placed in the ring binders, almost everything can be shown on the ring binders. This opens possibilities to implement user interfaces specific for mixing inside a ring binder. Another point of improvement is the way the system communicates with the hardware. A dedicated hardware layer can be added to do a mapping between the digital organisational structure and the physical hardware required to augment the physical organisational structure. Settings like IP-address and REST endpoints are hard coded at the moment.

A
Appendix

Bibliography

[1] Anand Agarawala and Ravin Balakrishnan. Keepin' It Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen. In *Proceedings of CHI 2006, Conference on Human Factors in Computer Systems*, pages 1283–1292, Montréal, Canada, April 2006.

[2] Joao Aires and Daniel Gonçalves. Personal Information Dashboard - Me, At a Glance. In *Proceedings of CSCW 2012, Conference on Computer Supported Cooperative Work*, pages 1–8, Seattle, USA, February 2012.

[3] Damián Arregui, Christer Fernstrom, François Pacull, Gilbert Rondeau, Jutta Willamowski, Elisabeth Crochon, and François Favre-Reguillon. Paper-based Communicating Objects in the Future Office. In *Proceedings of Smart Object conference*, Grenoble, France, 2003.

[4] Olha Bondarenko and Ruud Janssen. Documents at Hand: Learning from Paper to Improve Digital Technologies. In *Proceedings of CHI 2005, Conference on Human Factors in Computing Systems*, pages 121–130, Portland, USA, April 2005.

[5] Vannevar Bush and Jingtao Wang. As We May Think. *Atlantic Monthly*, 176:101–108, 1945.

[6] Irene Cole. Human Aspects of Office Filing: Implications for the Electronic Office. In *Proceedings of HFES 1982, Human Factors and Ergonomics Society Annual Meeting*, pages 59–63, Seattle, USA, October 1982.

[7] Christian Decker, Uwe Kubach, and Michael Beigl. Revealing the Retail Black Box by Interaction Sensing. In *Proceedings of ICDCS 2003, Conference on Distributed Computing Systems*, pages 328–333, Providence, USA, May 2003.

[8] Paul Dourish, Keith W. Edwards, Anthony LaMarca, John Lamping, Karin Petersen, Michael Salisbury, Douglas B. Terry, and James Thorn-

ton. Extending Document Management Systems with User-specific Active Properties. *ACM Transactions on Information Systems*, 18(2):140–170, April 2000.

[9] Susan Dumais, Edward Cutrell, JJ Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. Stuff I'Ve Seen: A System for Personal Information Retrieval and Re-use. In *Proceedings of SIGIR 2003, Conference on Research and Development in Informaion Retrieval*, pages 72–79, Toronto, Canada, July 2003.

[10] Susan T. Dumais and Thomas K. Landauer. Using Examples to Describe Categories. In *Proceedings of CHI 1983, Conference on Human Factors in Computing Systems*, pages 112–115, Boston, USA, 1983.

[11] Katherine M. Everitt, Meredith R. Morris, A. J. Bernheim Brush, and Andrew D. Wilson. DocuDesk: An Interactive Surface for Creating and Rehydrating Many-to-many Linkages Among Paper and Digital Documents. In *Proceedings of TABLETOP 2008, IEEE International Workshop on Horizontal Interactive Human Computer Systems*, pages 25–28, Amsterdam, The Netherlands, October 2008.

[12] Daniel Fällman. The BubbleFish: Digital Documents Available On Hand. In *Proceedings of INTERACT 2001, Conference on Human-Computer Interaction*, pages 9–13, Tokyo, Japan, July 2001.

[13] Stephen Fitchett and Andy Cockburn. AccessRank: Predicting What Users Will Do Next. In *Proceedings of CHI 2012, Conference on Human Factors in Computing Systems*, pages 2239–2242, Austin, USA, May 2012.

[14] Stephen Fitchett, Andy Cockburn, and Carl Gutwin. Improving Navigation-based File Retrieval. In *Proceedings of CHI 2013, Conference on Human Factors in Computing Systems*, pages 2329–2338, Paris, France, April 2013.

[15] Stephen Fitchett, Andy Cockburn, and Carl Gutwin. Finder Highlights: Field Evaluation and Design of an Augmented File Browser. In *Proceedings of CHI 2014, Conference on Human Factors in Computing Systems*, pages 3685–3694, Toronto, Canada, April 2014.

[16] Eric Freeman and David Gelernter. Lifestreams: A Storage Model for Personal Data. *ACM SIGMOD Record*, 25(1):80–86, March 1996.

[17] François Guimbretière. Paper Augmented Digital Documents. In *Proceedings of UIST 2003, Conference on User Interface Software and Technology*, pages 51–60, Vancouver, Canada, November 2003.

[18] Wilfred J. Hansen and Christina Haas. Reading and Writing with Computers: A Framework for Explaining Differences in Performance. *Communications of the ACM*, 31(9):1080–1089, September 1988.

[19] Bae S. Hark, Jong H. Choi, and Choon S. Leem. A Database Design of RFID Document Management System with e-Ink Technology. In *Proceedings of NCM 2008, Conference on Networked Computing and Advanced Information Management*, pages 14–17, Gyeongju, Korea, September 2008.

[20] Steve Hinske. Determining the Position and Orientation of Multi-Tagged Objects Using RFID Technology. In *Proceedings of PerCom 2007, Conference on Pervasive Computing and Communications*, pages 377–381, New York, USA, March 2007.

[21] Steve Hinske and Marc Langheinrich. Using a Movable RFID Antenna to Automatically Determine the Position and Orientation of Objects on a Tabletop. In *Proceedings of EuroSSC 2008, Conference of Smart Sensing and Context*, pages 14–26, Zurich, Switzerland, October 2008.

[22] Matthew G. Jervis and Masood Masoodian. Digital Management and Retrieval of Physical Documents. In *Proceedings of TEI 2009, International Conference on Tangible, Embedded, and Embodied Interaction*, pages 47–54, Cambridge, UK, February 2009.

[23] Matthew G. Jervis and Masood Masoodian. SOPHYA: A System for Digital Management of Ordered Physical Document Collections. In *Proceedings of TEI 2010, International Conference on Tangible, Embedded, and Embodied Interaction*, pages 33–40, Cambridge, USA, January 2010.

[24] Matthew G. Jervis and Masood Masoodian. Evaluation of an Integrated Paper and Digital Document Management System. In *Proceedings of INTERACT 2011, International Conference on Human-Computer Interaction*, pages 100–116, Lisbon, Portugal, September 2011.

[25] William P. Jones. Personal Information Management. *Annual Review of Information Science and Technology*, 41(1):453–504, 2007.

[26] William P. Jones. *Keeping Found Things Found: The Study and Practice of Personal Information Management.* Elsevier Science, 2010.

[27] William P. Jones and Susan T. Dumais. The Spatial Metaphor for User Interfaces: Experimental Tests of Reference by Location Versus Name. *ACM Transactions on Information Systems*, 4(1):42–63, January 1986.

[28] Hyunmo Kang and Ben Shneiderman. MediaFinder: An Interface for Dynamic Personal Media Management with Semantic Regions. In *Proceedings of CHI 2003, Conference on Human Factors in Computing Systems*, pages 764–765, Florida, USA, April 2003.

[29] David R. Karger, Karun Bakshi, David Huynh, Dennis Quan, and Vineet Sinha. Haystack: A Customizable General-Purpose Information Management Tool for End Users of Semistructured Data. In *Proceedings of CIDR 2005, Conference on Innovative Data Systems Research*, Asilomar, USA, January 2005.

[30] David R. Karger and Dennis Quan. Haystack: A User Interface for Creating, Browsing, and Organizing Arbitrary Semistructured Information. In *Proceedings of CHI 2004, Conference on Human Factors in Computing Systems*, pages 777–778, Vienna, Austria, April 2004.

[31] Akrivi Katifori, Costas Vassilakis, and Alan Dix. Ontologies and the Brain: Using Spreading Activation Through Ontologies to Support Personal Interaction. *Cognitive Systems Research*, 11(1):25–41, March 2010.

[32] Alison Kidd. The Marks Are on the Knowledge Worker. In *Proceedings of CHI 1994, Conference on Human Factors in Computing Systems*, pages 186–191, Boston, USA, April 1994.

[33] Jiwon Kim, Steven M. Seitz, and Maneesh Agrawala. Video-based Document Tracking: Unifying Your Physical and Electronic Desktops. In *Proceedings of UIST 2004, Conference on User Interface Software and Technology*, pages 99–107, Santa Fe, USA, October 2004.

[34] David Kirsh. A Few Thoughts on Cognitive Overload. *Intellectica*, 1(30):19–51, 2000.

[35] Scott R. Klemmer, Mark W. Newman, Ryan Farrell, Mark Bilezikjian, and James A. Landay. The Designers' Outpost: A Tangible Interface for Collaborative Web Site. In *Proceedings of UIST 2001, Conference on User Interface Software and Technology*, pages 1–10, Orlando, USA, November 2001.

[36] Hideki Koike, Yoichi Sato, and Yoshinori Kobayashi. Integrating Paper and Digital Information on EnhancedDesk: a Method for Realtime

Finger Tracking on an Augmented Desk System. *ACM Transactions on Computer-Human Interaction*, 8(4):307–322, December 2001.

[37] Michael G. Lamming and William M. Newman. *Activity-based Information Retrieval Technology in Support of Personal Memory.* Rank Xerox, EuroPARC, 1991.

[38] Thomas K. Landauer and D. W. Nachbar. Selection from Alphabetic and Numeric Menu Trees Using a Touch Screen: Breadth, Depth, and Width. *ACM SIGCHI Bulletin*, 16(4):73–78, April 1985.

[39] Thomas W. Malone. How Do People Organize Their Desks?: Implications for the Design of Office Information Systems. *ACM Transactions on Information Systems*, 1(1):99–112, January 1983.

[40] Dwight P. Miller. The Depth/Breadth Tradeoff in Hierarchical Computer Menus. *Proceedings of HFES 1981, Conference Human Factors Society and Ergonomics Society*, 25(1):296–300, October 1981.

[41] Bonnie Nardi, Ken Anderson, and Thomas Erickson. Filing and Finding Computer Files. Technical Report 118, Apple Computer, Inc, 1994.

[42] William Newman and Pierre Wellner. A Desk Supporting Computer-based Interaction with Paper Documents. In *Proceedings of CHI 1992, Conference on Human Factors in Computing Systems*, pages 587–592, Monterey, USA, May 1992.

[43] Moira C. Norrie, Beat Signer, and Nadir Weibel. Print-n-link: Weaving the Paper Web. In *Proceedings of DocEng 2006, Conference on Document Engineering*, pages 34–43, Amsterdam, Netherlands, October 2006.

[44] G. P. O. *Report on the Progress and Condition of the U.S. Nation Museum for the Year Ending June 30.* United States National Museum, 1939.

[45] Leo Sauermann. The Semantic Desktop - a Basis for Personal Knowledge Management. In *Proceedings of I-KNOW 2005, Conference on Knowledge Management*, pages 294–301, Graz, Austria, June 2005.

[46] Thomas Seifried, Matthew G. Jervis, Michael Haller, Masood Masoodian, and Nicolas Villar. Integration of Virtual and Real Document Organization. In *Proceedings of TEI 2008, Conference on Tangible and Embedded Interaction*, pages 81–88, Bonn, Germany, February 2008.

[47] Abigail J. Sellen and Richard H.R. Harper. *The Myth of the Paperless Office*. MIT Press, 2003.

[48] Beat Signer and Moira C. Norrie. A Framework for Cross-Media Information Management. In *Proceedings of EuroIMSA 2005, Conference on Internet and Multimedia Systems and Applications*, pages 318–323, Grindelwald, Switzerland, February 2005.

[49] Jeff Smith, Jeremy Long, Tanya Lung, Mohd M. Anwar, and Sriram Subramanian. PaperSpace: A System for Managing Digital and Paper Documents. In *Proceedings of CHI 2006, Conference on Human Factors in Computing Systems*, pages 1343–1348, Montréal, Canada, April 2006.

[50] 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 *Proceedings of CHI 2004, Conference on Human Factors in Computing Systems*, pages 415–422, Vienna, Austria, April 2004.

[51] Sandra Trullemans and Beat Signer. From User Needs to Opportunities in Personal Information Management: A Case Study on Organisational Strategies in Cross-media Information Spaces. In *Proceedings of DL 2014, Conference on Digital Libraries*, pages 87–96, London, UK, September 2014.

[52] Sandra Trullemans and Beat Signer. Towards a Conceptual Framework and Metamodel for Context-Aware Personal Cross-Media Information Management Systems. In *Proceedings of ER 2014, Conference on Conceptual Modeling*, pages 313–320, Atlanta, USA, October 2014.

[53] Dennis C. Tsichritzis. Form Management. *Communications of the ACM*, 25(7):453–478, July 1982.

[54] Ayrton Vercruysse. Big Brother is Watching Your Documents. Master's thesis, Vrije Universiteit Brussel, 2015.

[55] Fernanda B. Viégas, Scott Golder, and Judith Donath. Visualizing Email Content: Portraying Relationships from Conversational Histories. In *Proceedings of CHI 2006, Conference on Human Factors in Computing Systems*, pages 979–988, Montréal, Canada, April 2006.

[56] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.

[57] Pierre Wellner. The DigitalDesk Calculator: Tangible Manipulation on a Desk Top Display. In *Proceedings of UIST 1991, Conference on User Interface Software and Technology*, pages 27–33, Hilton Head, USA, November 1991.

[58] Pierre Wellner. Interacting with Paper on the DigitalDesk. *Communications of the ACM*, 36(7):87–96, July 1993.

[59] Ron Yeh, Chunyuan Liao, Scott Klemmer, François Guimbretière, Brian Lee, Boyko Kakaradov, Jeannie Stamberger, and Andreas Paepcke. ButterflyNet: A Mobile Capture and Access System for Field Biology Research. In *Proceedings of CHI 2006, Conference on Human Factors in Computing Systems*, pages 571–580, Montréal, Canada, April 2006.