Document Tracking

Diploma Thesis

Marcel Rieser
<mrieser@student.ethz.ch>

Moira C. Norrie
Corsin Decurtins

Global Information Systems Group
Institute of Information Systems
Department of Computer Science
ETH Zurich

1.0
25th July 2005
Abstract

Paper is still an important information carrier, even in our more and more electronically penetrated environment. Every day we use paper documents in our office, even when electronic versions of the documents are available. Instead of replacing the paper completely as proposed by the myth of the paperless office, new ways for better integrating paper into our daily work are needed.

This diploma thesis researches some possibilities how paper documents can be tracked on a desktop, enabling a range of new methods of interactions between paper documents and electronic devices, mainly personal computers. The thesis focuses mostly on algorithms and data models used to track documents, but not on sensor technology for tracking objects. Proposed algorithms to track documents on a desktop are implemented in a prototype system. A demonstrator application allows users to perform actions on a computer with documents currently located on a desk.

The goal was to restrict the user working with paper as little as possible in handling the paper documents. The user should be allowed to pile documents and move such stacks as he may be used to do. The goal could be nearly reached, as most of the movements of documents are tracked correctly. But because of some latency when locating documents, fast movements can yield to temporarily wrong results.
Acknowledgments

I would like to thank Corsin Decurtins for his engagement in this work. From the first minute I contacted him about this project I received valuable feedback and guidance from him. I also appreciate very much Prof. Dr. Moira C. Norrie for leading the Global Information Systems Group at ETH Zurich and thus making such interesting projects available to students.

The implementation makes heavy use of OMSjp, developed and maintained by Michael Grossniklaus. For his support and work I am very grateful. Michael Rohs offered his code to locate two-dimensional barcodes in images coming from a video camera for what I want to thank him.

Special thanks go to my parents who always supported me and my decisions, and who made this education possible to me.
**Contents**

Abstract iii

Acknowledgments v

1 Introduction 1

2 Tracking Variants 3
  2.1 Technologies ........................................... 3
  2.2 Tracking of Paper Documents ........................ 4

3 Concept for Tracking Documents 7

4 Implementation 11
  4.1 Components and Technologies .......................... 11
  4.2 Database Model ...................................... 12
  4.3 Sensor: Recognition of Markers ......................... 13
  4.4 Handling of Localised Markers ........................ 15
  4.5 Automated Data Aggregation ........................... 16
  4.6 Movement of Documents ................................ 18
  4.7 Event-Listeners ..................................... 20
  4.8 Viewer ............................................ 20

5 Applications 23

6 Conclusions 27

7 Future Work 29

A Object Model 31
Distributed and Ubiquitous Computing resounds throughout the land. But in most cases, only electronic devices are part of the installation/network. This resembles the euphoria about the paper-less office some decades ago, meanwhile proven to be a myth. For this reason, new ways are required for the handling of paper documents, integrating them to one’s best in our workflow instead of substituting them. First approaches were made by Pierre Wellner using his DigitalDesk [Wel91, Wel93], which included a calculator using numbers on paper as input. Soon, additional applications were made, e.g. a digital drawing board by Mackay et al. [MVC+93].

Another contemporary issue are applications and devices, which act differently depending on the context the user is in. One of the best-known examples are mobile phones which recognise when its owner is in a theatre or a meeting and thus select a soundless profile on its own. Adding a bit of complexity, the phone could try to find out what the discussion its owner participates in is about and then decide, if the call may be important for the callee in that situation or not. Such context-awareness could also be used by mail applications on our computers, which pop up windows every now and then informing us about newly received emails and interrupting our work. What if the mail application would only show us emails important for the current topic we are working on, maybe showing us other emails when having finished a topic or when coming from a break, when the flow of work is already interrupted? For such questions, one of the most important context information in a desktop environment is what the user is currently working on (user context). This information could be derived from currently opened (or most recently accessed) documents on the computer, the last sent mail messages and so on. But as it was already mentioned, our office is not a completely electronic one. We may have talked about the current work to a co-worker or had a phone call with someone else, or we could be reading a paper document, make annotations in there and not use the computer for a while. So it is clear that user context information must also be gathered in the “real” world. Paper is still a very important part of our office environment, its usage even rising
while electronic documents and mail messages are printed out, filed, and taken into reconsideration later again. Thus paper documents would be a good source to collect context information, if they could be located and identified on a desk. Today’s personal computers offer ever increasing computer power, mostly used by games or high-end applications like digital audio- and video-editing. But in office environments, this computing power lies idle and could be used to gather context information.

This thesis researches some possibilities to track paper documents on a desk, using a personal computer and some additional tracking equipment. Chapter 2 has a look at various possible tracking variants, some better suited to track papers, some less. Chapter 3 introduces the proposed concept to track documents, while in chapter 4 a concrete implementation is described. The results of the work are shown in chapter 5 by the means of a working prototype application, making use of the document tracking implementation. Chapter 6 summarises the experience made in this project, while an outlook at possible extensions and improvements is given in chapter 7.
2 Tracking Variants

2.1 Technologies

Over the last decades, a multitude of tracking technologies emerged. Some of them are designed for very special tasks, while others are more generally applicable. Early experiments for object tracking were mostly based on infrared or super sonic signals. An example is the Active Badge System [WHFG92] developed in the late 1980s at the Olivetti Research Laboratories (ORL) to locate its employees within its building. The employees wore badges, broadcasting a unique code using infrared signals. The codes were received by passive sensors distributed in the building, allowing for a simple localization of the wearer of the badge.

A few years later, AT&T improved the Active Badge System, allowing for more detailed localization and also identifying the direction of the person wearing the badge. The new system used ultrasonic waves and was called Active Bat System [Act05].

Over the years, the active beacons got more and more replaced by passive ones. The last example are Radio Frequency Identification (RFID) tags 1, used to mark clothes, food, transport documents and many other objects.

To track documents, often digital video cameras are used as devices to capture images which are then analysed. Pierre Wellner used a camera in combination with Optical Character Recognition (OCR) to input numbers into a calculator on his Digital Desk [Wel93], while Kawashima et al. [KNMA99] use image algorithms to track the movements of documents as well as the actions of the user’s hands and fingers. Kim et al. [KSA04] describe a project where a camera acquires a small image of a document placed on the desk, which is then compared to similar sized thumbnails of documents generated from their electronic counterparts (e.g. from PDF files) to identify it. Several other examples could be mentioned like LivePaper [RR99], the digital drawing board [MVC+93] or augmented flight strips [MF99].

2.2 Tracking of Paper Documents

Several of the above mentioned technologies cannot be reasonably applied to track documents, mostly because of their size (e.g. beacons). RFID tags are still too expensive to be attached to every printed document, besides that they must be manually attached and cannot be integrated in the paper yet. Additionally, Floerkemeier and Lampe [FL04] show that RFID tags cannot be distinguished and tracked well if too many of them are in a small, restricted area—a placement not uncommon to papers when piled on top of each other.

Many video camera based solutions suffer from the limited resolution of today’s video cameras. Improvement can be reached by using either multiple cameras as in [HVA+05], or a steerable camera to take close-ups when needed as described in [BSS04, KNMA99]. Both solutions enlarge the technical complexity to acquire document positions significantly.

By using simple identifiable, visual markers, the position of documents can be acquired even with the limited resolution of today’s video cameras. Many different forms of markers exist today. One of the first bar codes was the bull’s eye barcode, developed by Woodland and Silver in the late 1940s [WS49]. Later, several two-dimensional barcodes were developed, e.g. CyberCode by Rekimoto and Ayatsuka [RA00] or VisualCodes by Rohs and Gfeller [RG04]. While these 2D barcodes can contain a limited amount of information, usually some unique identifier, markers for the ARToolkit [KB99] contain only visual information (fig. 2.1).

Figure 2.1: Examples of some 2D markers: Bull’s eye, CyberCode, VisualCode, ARToolkit pattern (from left to right)

But visual markers pose the problem that they can be hidden by other documents and thus no longer be visible to the camera, a disadvantage that solutions based on IR or supersonic waves do not have. The projects using cameras to locate documents treat hidden documents or stacks of documents differently. Some add restrictions like the one that only the top-most document can be moved at a time (e.g. [KSA04]), ensuring that any hidden document remains at its place. Others just ig-
nore the correct layering within a pile, only identifying the top-most document on a stack [KNMA99] while ignoring the other documents. Fujii et al. [FSAA03] implement a way to keep the correct order in a stack of objects, even when objects are extracted from the middle of a pile. This requires not only a top-view of the desk, but also a side-view on the stack. However, it only works with relatively thick objects like compact disc cases but not with single sheets of papers. In office environments where regularly piles of documents are moved around in the search for some specific paper, many of the above mentioned restrictions are severely limiting the user’s traditional work style. A way to specify a document’s position, even after being moved around on a desk while its marker was hidden, is required if the tracking should be applicable to real world scenarios.
3

Concept for Tracking Documents

As discussed in the previous chapter, many different tracking technologies are available. It is feasible that not only one technology is used to track documents, but several complementing technologies are adapted. In such a case, multiple sensors will collect data.

A central database stores all sensor data along information about markers and documents. Additionally, meta information about documents can also be stored in the database such as dimension, title and keywords. Based on the marker locations, data can be aggregated and analysed, storing the derived information also in the database.

The stored data can be grouped into three different categories:

**Facts** For correct operation, the database must be given some basic information before tracking starts. This includes information about existing markers, documents identified by them as well as the markers’ position within the document area. This information must be either entered manually or using an administration tool.

**Hard Context** Every time a sensor locates a marker, a hard context information is created. Hard context can be understood as a fact about a document’s place, unless the sensor fails or works inaccurate and delivers wrong input. The information stored as hard context must be true and verifiable at the time it is entered in the database.

**Soft Context** Information derived from hard context is said to be soft context [GBSV03, Bel04]. Its correctness has to be assumed based on the knowledge stored in the hard context information. For example, given the fact where a region is and the hard context information where a document is located, documents can be assigned to regions. The information about the ordering of documents in a pile is also a soft context information, as it is derived from the
locations and times the documents where first seen in the area the pile covers.

The sensor applications should be as lightweight as possible, only implementing sensor-specific transformation of data. A data access layer, common to all sensors, will handle the correct storage of the data and trigger events when needed, thus taking part in implementing the business logic.

The business logic is responsible for correctly interpreting the stored marker information, drawing the right conclusions about the relation of multiple documents and handling the aggregation of information.

Because more than one sensor could track documents or the place of a sensor could change, all locations submitted to the database must use the same coordinate system. Otherwise, the data from different sensors could not be combined, or the correct interpretation of the stored data would get lost, when sensors are moved around.

The most natural coordinate system is thus the system of the observed object—in our case the desk. Consequently each sensor must ensure it can transform the recorded location into desk coordinates.

Two types of clients can be distinguished:

**Viewers** access the stored locations of documents for visualisation or spacial queries.

**Event-Listener** subscribe themselves to one or more types of events, getting informed everytime a relevant change in the database happens.

Viewers and Event-Listeners can be combined into one process to build sophisticated user-applications, e.g. allowing the user to see a real-time view of the captured data by updating its view everytime a document is located at a new place.

Figure 3.1 shows a schematic overview of the technical concept.
Figure 3.1: Sensors store information in a database, while viewers and event-listeners read and act on the data.
4 Implementation

4.1 Components and Technologies

A workplace was set up at the GlobIS lab at ETH Zurich to develop and test the document tracking functionality. As computer, a machine running Windows XP on a Pentium 4 processor at 2.8GHz with 1 GB DDR RAM was used.

As database, OMS Pro [OMS05a] was chosen. OMS Pro is part of the object-oriented OMS Database Development Suite developed at ETH Zurich and well suited for rapid prototyping and database design. Using OMSjp [OMS05b], the database can be accessed from Java-Applications, in which several Graphical User Interfaces (GUIs) are implemented acting as sensors, viewers or event-listeners.

The database uses the Object Model (OM) as a generic, object-oriented data model. The OM can be seen as an extended version of the Entity-Relationship-Model (ERM). Collections store one or more objects of the same type, comparable to relations holding tuples. Associations interconnect collections similar to the relationships found in ER-Models. Because all data is stored in objects, other object-oriented concepts can be integrated into the OM: types not only hold data, but also methods to operate on the data. Additionally, types can have sub-types, and thus collections can have sub-collections.

OMS Pro supports macros as a kind of stored procedures as well as methods on stored objects. This allows it to include most of the business logic right into the database itself. But OMS Pro also imposes restrictions, the most central one being its inability to handle concurrent connections to more than one process. This is overcome in the realised prototype by using a proxy-like class OMSConnector, which also implements the data access layer.

Additionally, a cache containing a copy of the database’s content at runtime complements the data abstraction layer. The cache prevents redundant database access while at the same time ensuring that multiple viewers have the same dataset they are working on. Event-Listeners subscribe to the cache to be informed about events,
4.2. DATABASE MODEL

as the cache acts as a proxy or gateway to the database. The cache loads the stored data from the database at start-up. It can then either be updated manually on request, reloading the database content, or kept in sync while new data is processed from the sensors. Because the continual synchronisation leads to higher loads and thus may impact the performance of the marker recognition, it will only be activated when at least one event-listener is subscribed to the cache. Figure 4.1 shows a modified concept to match the given characteristics of the used components.

![Figure 4.1: The modified concept, including a database connector and a data-cache, based on the characteristics of the used components.](Image)

The GUIs for the different applications (sensors, viewers, event-listeners) are implemented in Java.

4.2 Database Model

In this realisation, a single database stores all information needed to track the documents. Figure 4.2 shows a simplified object-model of the database. The full model is included in appendix A. Facts are stored in the collections Markers, Documents, Positions and the associations connecting them as well as in the collection Regions. Hard context is stored in the collection DocLocations and the associations locatedAt and basedOn. All other entities are used to store soft context information: the collection of Piles, the association inPile describing which documents are part of which pile at a certain location as well as the ordering of documents within a pile, stored in the association above. Additionally, soft context information can also be stored in DocLocations alongside hard context information. This is the case when assumptions about the location of a document has to be made, e.g. when a pile of documents is moved, but not every marker of all involved documents can be located (see chapter 4.6 for more information about movements of documents). Figure 4.3 shows the defined object classes together with their members and functions in an UML diagram.
4.3 Sensor: Recognition of Markers

The implemented sensor recognises the location and rotation of two-dimensional barcodes. Here Visualcodes [RG04] are used as markers because they allow for relatively precise localisation. Furthermore an implementation for finding the position of markers in an image was already available in Java. The code for the visual recognition analyses single frames from a video stream to find the markers. Each time a marker is found, its location is measured in pixels from the top-left corner of the image and passed to a function for further processing.

A digital video camera (Sony DCR-TRV18E) is used as capture device. Mounted on a tripod about 1 metre above the desk (fig. 4.4), its image covers an area of roughly 60 by 45 centimetres (about the size of a DIN A2 paper). Connected via Firewire, a video-stream in PAL quality (720 by 579 pixels) can be acquired. Because of the cam-

---

**Figure 4.2:** Simplified Object Model

**Figure 4.3:** UML diagram of the defined types in the Object Model of the database.
era’s limited resolution and its distance from the desk, the barcode markers must be at least 25 millimetres long on each side to be recognised by the camera. Future video standards (e.g. High Definition Television – HDTV) or a combination of multiple sensors could further boost the covered area or allow smaller markers.

After starting the sensor, the camera needs to be calibrated before any markers can be tracked, so the sensor can transform the marker locations in desk coordinates. Alternatively a previous calibration can be loaded in case the camera is known not to have moved relatively to the desk since. Given three points whose coordinates must be known in image coordinates as well as desk coordinates, a 2D affine transform can be defined that performs a linear mapping from one 2D coordinate system into another. Such a transform can be constructed using a sequence of translations, scales, flips, rotations, and shears. This implies that parallel lines remain parallel to each other after the transform, requiring that the camera plane is parallel to the plane of the observed desk. If this cannot be guaranteed, four points must be given to define an affine transform in 3D.

For testing purposes, the sensor can store the captured marker-locations in a file and reload it later. This makes it possible to replay previously captured scenes and to test different algorithms such as for recognising the movement of documents (see chapter 4.6).

Figure 4.5 shows the GUI of the implemented sensor.
4.4 Handling of Localised Markers

The sensor calls a macro in the database to store the location of a marker in the database, passing the ID and the calibrated location of the marker as well as the time the marker was seen as parameters. The macro first loads the matching marker and the document identified by it. Next, the macro compares the current location with the document’s last known location, checking whether the document has moved or not. If the difference between the new and the latest stored position is within a set threshold, it is assumed that the document did not move. This prevents the system from too many updates caused by tracking fluctuations.

If the document did not move and the last location was a hard context information, only its last-seen time is updated. In all other cases, a new location is created because the document either moved or its (previously soft context) location got confirmed. Listing 4.1 shows in pseudocode the basic differentiations which are acted upon when a new localisation is made.

When a new location is created for a document, it must be ensured that a possible piling order is maintained. This happens either in `confirmLocation` or in
4.5 Automated Data Aggregation

Many applications do not need to know the exact position of a document. Often it is enough to know that a document has entered or left a certain area on the desktop, or that a document has been placed on top of another. The assignments of regions to a document location as well as the recognition of putting one document on top of another is done within the database, each time a new document location is generated. This prevents different clients from assigning documents to different regions or from using different algorithms to recognise document stacks, thus having a different view despite using the same base data.

The assignment of regions to a document location is fairly simple: For each newly generated document location the database checks whether the document lies within one or more of the stored regions. This is done by looping over all stored regions,
Listing 4.2: most important steps of the method confirmLocation (pseudocode)

```java
method DocLocation.confirmLocation(DocLocation NewLoc) {
    /* self: the old soft context location being confirmed.
       * NewLoc: the hard context location replacing self.
       */
    int time = NewLoc.getTimeFirstSeen(); // 'current' time

    /* add the confirmed location to the same piles
       * as the soft context location is currently in.
       */
    Pile[] myPiles = self.getPiles();
    NewLoc.addPile(myPiles);

    /* Update the last-seen times of all soft-context
       * locations beneath the confirmed location.
       */
    foreach (DocLocation loc = self.getDocsBelow()) {
        /* Hard context location must not be changed,
           * otherwise they would no longer be hard context!
           */
        if (loc.isSoft()) {
            loc.setTimeLastSeen(time);
        }
    }

    /* check all soft context locations above whether they
       * would hide the marker of newLoc.
       */
    foreach (DocLocation loc = self.getDocsAbove()) {
        if (loc.isSoft() &&
            loc.coversPoint(NewLoc.getMarkerPosition())) {
            /* 'loc' is soft and would hide our marker. But the
               * marker was seen! Thus we have to assume that
               * the document is not located at 'loc' (it is just
               * a soft context location after all).
               * Reduce its last-seen time by on tick so it is no
               * longer counted as part of the 'latest locations',
               * assuming the same document has other soft
               * context locations with the same time where the
               * document could be.
               */
            loc.setTimeLastSeen(loc.getTimeLastSeen()-1);
        }
    }
}
```
adding a corresponding entry in the association collection inRegion if the center of
the document is covered by the region.
To recognise whether a document was placed on top of another, a similar approach
was chosen: when placing a document, the database loops over all other documents
fetching their last location and comparing it to the location of the newly placed doc-
ument. If the area of a former document covers the centre of the new document, it is
assumed that the document was placed above the other.
The information that one document is above another document is also stored in the
database. It facilitates correct visualisation of the piling order of documents in a
viewer. It also helps to recognise when a document may have been moved, although
its marker could not be seen (see chapter 4.6 for details).
The order of documents within a pile is primarily given by the time a document was
seen at a certain location: documents with an earlier first-seen time at a location are
assumed to be lower in a pile than documents with a later first-seen time. As soon
as two documents are assigned a spatial relationship within a pile, the order of the
documents must not change when both documents are moved around and relocated
together. The following chapter will cover the challenges posed by this requirement
in more detail.

4.6 Movement of Documents

When a marker is located at a new place, the corresponding document is informed
about the location change by a call to document.moveTo, passing the new location
as a parameter. The document is then responsible for setting the new position and
moving potential documents beneath to the new location.
In a first step, the document loops through all other documents’ last locations, check-
ing whether it covers their area or not. If an overlap of the two documents is found,
the new location is added to the same pile the other document is already in. When
adding the new location to the pile, the method tries to restore a possible previous
stacking order by checking whether the covered document is also in the pile the
moved document was in at its last position and restoring the same order if found
so. If this is the case, all other documents between the two documents are also as-
sumed to have moved along them. New (soft context) locations are generated for
them relative to their position in the old pile at the new location.
After testing all the other documents on a possible overlap, the new location should
be at least in one pile if a cover was found. If not, the document seems to have been
moved to a previously empty area and a new pile is generated for this document.
In a last step, all documents beneath the moved document in the old pile are moved
to the new location, as long there is no evidence that the documents stay in place.
Because we cannot be sure that the documents beneath really did move as long as
their markers are not located somewhere, the documents can be either at the old
place or in the new pile. To indicate this, the documents are not only moved to the
new (assumed) place, but a copy is made and moved. This way, a document can
have two or more “last locations” where it can be with some probability.
Listing 4.3 gives an pseudocode-implementation of the method.
When a pile is moved several times, the documents below the top-most one leave a
Listing 4.3: main structure of the method moveTo (simplified, pseudocode)

```java
method Document.moveTo(DocLocation NewLoc, int Time)
{
    /* STEP 1: add us to any existing pile we cover by testing
    * all last locations of all other documents.
    * If we were at our last location in the same pile as the
    * last location of the other document then move all docu-
    * ments between us in the old pile over to the new pile
    * (done in 'pileMoving').
    */
    foreach(Document doc = getAllDocuments()) {
        if (doc != self && NewLoc.covers(doc.getLastLocation())) {
            DocLocation otherLoc = doc.getLocation();
            Pile[] piles = otherLocation.getPiles();
            NewLoc.addPile(piles);

            DocLocation myOldLoc = self.getLastLocation();
            Pile[] myOldPiles = myOldLoc.getPiles();
            DocLocation otherOldLoc = doc.getLastLocation();
            Pile[] otherOldPiles = otherOldLoc.getPiles();
            if (intersect(myOldPiles, otherOldPiles).count() > 0) {
                // we were on the same pile at our last locations
                // restore correct order in pile
                if (myOldLoc.isAbove(otherOldLoc)) {
                    NewLoc.setAbove(otherLoc);
                    pileMoving(otherOldLoc, myOldLoc, otherLoc, NewLoc);
                } else {
                    otherLoc.setAbove(NewLoc);
                    pileMoving(myOldLoc, otherOldLoc, NewLoc, otherLoc);
                }
            }
        }
    }

    /* STEP 2: if we have not yet any pile associated,
    * it means we didn’t cover any of the other documents.
    * Create a new pile and add us to it in this case.
    */
    if (NewLoc.getPiles() == null) {
        NewLoc.addPile(new Pile());
    }

    /* STEP 3: move any documents that were beneath us at our
    * last location and which may have moved along with us.
    */
    myOldLoc.moveDocsBelow(myLoc);
}
```
trail on the desk with possible locations, because it cannot be assumed that always
the complete pile moves. This can quickly lead to a cluttered virtual desktop with
the same documents shown at multiple places. Some of these document locations
could be ruled out after some amount of time: When the marker should be visible
because no other document is above it, but the marker was not located for several
seconds, it can be assumed that the document is not at that place and the document
location can be removed.
This “cleanup” is done by the method cleanupDocLocations, which is regularly called
by the OMSConnector (see chapter 4.1). The method loops over all documents’ last
locations, checking each whether it is a soft context information and whether the
marker should be visible to the camera. If both tests succeed and the soft location
was created at least 5 seconds before the current time, the document’s location is
modified such that it no longer is one of the last locations of the document.

4.7 Event-Listeners

Two different groups of events are distinguished in the implementation: The first
group contains events on a document itself, currently the two cases that a document
either moves or stays in place. The seconds group consists of events caused by the
movement of a document. As examples, the two cases that a document enters or
leaves a specified region are implemented. This makes a total of four events which
are currently supported in the implementation, all listed in table 4.1. Clients can
subscribe to be informed whenever one of these events happens.
A very simple Event-Listener was implemented logging all events into a text area, as
shown in fig. 4.6.

![Figure 4.6: GUI of the implemented event-listener.](image)

4.8 Viewer

A simple viewer application was implemented to visualise the stored locations, al-
lowing to quickly examine the correct behaviour of the stored data and data aggrega-
tion algorithms within the database. Figure 4.7 shows the GUI of the application.
The user can select from a list those documents that should be visualised together...
Table 4.1: Implemented Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Passed Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document moved</td>
<td>Document, Time, DocLocation</td>
</tr>
<tr>
<td>Document stayed at place</td>
<td>Document, Time, DocLocation, Duration</td>
</tr>
<tr>
<td>Document entered region</td>
<td>Document, Time, Region</td>
</tr>
<tr>
<td>Document left region</td>
<td>Document, Time, Region</td>
</tr>
</tbody>
</table>

with the respective time the locations should be showed for. The timeline offers an easy way to follow the positions of moving documents. The viewer shows the position of the documents and the markers on which the positions are based. When several documents are piled on top of each other, the viewer correctly shows the stacking order and marks those documents being part of the same pile. Additionally, regions defined in the database can be highlighted in the image.
Figure 4.7: GUI of the implemented viewer.
Document tracking features cannot only be used to gather context information, as it was mentioned in the introduction. The tracked locations could also be used to build a search engine for objects on the desktop or by providing additional possibilities for interaction between physical documents and their electronic counterparts.

The implemented viewer (see chapter 4.8) can already be used as a simple search engine. The user can select a document from the list and set the time to the latest time available to see the location where the document was last seen or where the document is believed to be in the case it was moved while being hidden by another document.

To test and demonstrate some interaction possibilities, another application was built. The prototype uses the event-listener interface to keep up-to-date on the document’s locations. As soon as a document enters a user-defined region, the document’s name is shown in a list together with the file path of the electronic version of the document. The user is then given several possibilities to interact with the listed files:

**Opening electronic version of document** By double-clicking on the list-entry, the file whose path is stored in the database is opened.

**Retrieving additional information from file** The application can extract author-information from the meta-data stored in PDF files or extract keywords from the full text of the document.

**Providing simplified lookup of additional information** The user can search the internet for either the document title, the author’s name or keywords with one simple click.

**Broaden or narrow the search** By adding documents to the selected region, the information from the documents can be combined in several ways. This allows the user to modify the terms the internet is searched for.
Figure 5.1 shows the GUI of the implemented prototype. To extract meta-information from PDF documents, the PJ class library from Etymon [ety05] is used. Keywords are extracted using tools from the multivalent project [mva05]. This project provides java classes to extract text from a variety of document types. It also offers algorithms to reduce longer texts to a set of keywords which identify the document (in most cases) uniquely. Such a set of keywords is known as a **lexical signature** and could be used to create so-called robust hyperlinks [PW00].

When multiple documents are located in the selected region, the documents’ keywords can be combined either by a union or an intersection. Using the differently combined keywords for a search on the internet will most likely yield to different documents found.

The application uses Google\(^1\) as search engine. The user can either manually search for the listed terms by clicking on a button or have the application automatically search for the set of terms whenever the set changes. Additionally the user can choose whether he wants to see the list of search results, or if the best match should be chosen and opened automatically by using Google’s “I’m feeling lucky”-function.

\(^1\)http://www.google.com/
Several of the demonstrated technologies could be useful to other projects. For example, the keyword and meta-data retrieval could be integrated into community awareness systems like [SWS01], displaying information on what users are working in an office to visitors or other users.

By not only referencing documents, but other electronic information as well, additional interactions will be possible. If the printout of a mail message is recognised, why not offer the functions to open that mail, to reply to it or to call the mail’s sender on the phone?
Conclusions

The implemented prototype demonstrates that document tracking allows for new ways of interacting with the computer. It also shows that documents can be tracked on a desk, even when some documents are hidden in a pile. But the prototype also showed the limitations of the system:

- The range of the implemented sensor is too small for a real office desktop.
- The used markers must be rather big and are thus not very pleasing to the eye.
- The quality of the marker recognition is currently limiting how fast documents can be moved or covered by other documents, because the markers are not recognised in every frame of the video.
- The marker recognition algorithm is able to process 4 to 5 frames per seconds on a otherwise idling system on our test machine. The frame rate drops further down when other processes also run on the same machine, e.g. the database to store the marker locations.
- Data access operations to the database are relatively slow, as are the execution of macros and methods in the database.
- Because of the slow marker recognition, documents have to be placed consciously and should not be moved again or covered too soon after.

Despite all the aforementioned limitations, the system works and can track the location of documents, even of hidden ones, in a desktop environment. By studying the detailed implementation of the algorithms, it can be assumed that the algorithms can be fooled and irritated with special document movements and under certain conditions. But the likelihood of such cases are very rare in a typical office and as time passes, the stored information will return to the real image as additional information is gathered in the further time lapse.
While the current prototype works relatively good in the small desktop environment, it can still be further improved in a variety of ways. It even must be improved if it should be used in real office environments. One of the most severe shortcomings is the currently limited tracking area. As already mentioned, this could be overcome by adding additional sensors. Support to store data from multiple, even different types of sensors, is already integrated in the database. But the algorithms used to interpret and aggregate the information will most likely fail to work correctly when multiple sensors are involved.

The main problem using multiple sensors is the concurrency of the sensors. As long as only one sensor is used, the database receives all localisations in a strictly chronological manner. This is no longer ensured when two or more sensors are used. Additionally, the implemented macros and methods must be tested thoroughly in regards of real object-orientation—currently the code contains some assumptions about class’ members for localisations based on visual markers.

An interesting enhancement would be to add a semantic meaning to markers. This would lead to a more generalised object tracking instead of just tracking documents. An example could be that folders can be differentiated from paper documents. Having this possibility would allow to recognise defined actions, e.g. a document put into a folder on the real desk would move the electronic version of the document into a directory corresponding to the folder, or the document could be marked with the folder’s label stored in the file’s metadata.

In a similar step to diversify the tracked objects, the view should not only be focused on electronic documents, but on any kind of electronic information. Applications may act different when they know that the tracked object is the print-out of a mail message compared to a book.

Several other improvements could be thought of:

- Support to locate one marker at several places at the same time. Currently, each marker must be unique (at least within the area covered by the sensors). This
prevents copies of a document to be correctly tracked when they are visible at the same time.

- Support for sensors locating documents not at an exact point, or sensors adding a quality value to their localisation. Some types of sensors, e.g. ones using RFID, cannot locate markers as exact as the video camera used in the prototype can, maybe only giving a region where a document is located in. Such fuzzy information lead to much more complex algorithms, which must be first researched.

- Regions are not identified by their coordinates on the desk, but by movable objects or markers. This would allow to have the in-tray defined as a region, giving the pile within its area a special meaning. This may be combined with the semantic meaning of markers, where one or more markers could define a region.

Furthermore, the current implementation is limited in speed, as was already mentioned. One limiting factor is the image-analysis to find the barcodes in the video-image, another one is the used database. While the first one could be solved by using a different sensor or a more optimised code for the image-analysis (besides using a faster machine), the second one requires the exchange of the database backend. Finally, ways must be found to integrate the new possibilities for interaction into the users environment, most likely on the computer. One idea is to have a little panel listing the topmost tracked documents together with some buttons to start actions, like opening the document, replying to the tracked mail message or call the author of a document on the phone. Such a panel could be integrated into the sidebar seen in some early alpha versions of Microsoft’s upcoming Operating System “Windows Vista” (codenamed “Longhorn”; the sidebar was removed in later developer preview versions—time will tell if this feature is available in the final version), or into Dashboard on computers running Mac OS X\(^1\). Another idea is to use the keywords to search on one’s own computer instead on the internet. This could be done by using Google’s Desktop Search\(^2\) or Apple’s Spotlight search technology\(^3\).

If the object or document tracking could be delegated to a background process, and with it the analysis of the location information and context retrieval, an interface should be defined to allow other applications make use of this data. Many applications today can be extended by plug-ins which could be an easy way to add context-sentive functionality to existing business applications.

\(^{1}\)http://www.apple.com/macosx/features/dashboard/, accesses July 2005

\(^{2}\)http://desktop.google.com/, accesses July 2005

The Object Model (OM) shows how data is stored in the database (fig. A.1). The collection Markers, which stores all objects of type marker, can be extended by sub-collections of sub-types of marker. This makes it possible to have additional attributes for different types of markers.

Similarly, the collection DocLocation can be diversified so that locations of different types of markers can be stored. Currently, the locations of VisualCode-Markers are stored in OrientationLocation, a sub-type of doclocation which allows to store one point (the position the marker was seen) and a direction (the orientation of the marker). Other types of marker may have other storage requirements, e.g. to store the four vertices of a located document in VertexLocations.

Regions can either be defined by rectangles or ellipses. Other forms like polygons can be added simply by sub-typing region. Regions can be compound to define larger, complex regions or to build a hierarchy within the regions: A campus-region could consist of several building-regions, which can be compound from several room-regions, and so on.
Figure A.1: Complete Object Model of the database
Bibliography


[KB99] Hirokazu Kato and Mark Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In


