An Extensible Digital Ink Segmentation and Classification Framework for Natural Notetaking

Adriana Ispas
Institute for Information Systems, ETH Zurich
8092 Zurich, Switzerland
ispas@inf.ethz.ch

Beat Signer
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
bsigner@vub.ac.be

Moira C. Norrie
Institute for Information Systems, ETH Zurich
8092 Zurich, Switzerland
norrie@inf.ethz.ch

ABSTRACT

With the emergence of digital pen and paper technologies, we have witnessed an increasing number of enhanced paper-digital notetaking solutions. However, the natural notetaking process includes a variety of individual work practices that complicate the automatic processing of paper notes and require user intervention for the classification of digital ink data. We present an extensible digital ink processing framework that simplifies the classification of digital ink data in natural notetaking applications. Our solution deals with the manual as well as automatic ink data segmentation and classification based on Delaunay triangulation and a strongest link algorithm. We further highlight how our solution can be extended with new digital ink classifiers and describe a paper-digital reminder application that has been realised based on the presented digital ink processing framework.

Author Keywords

digital pen and paper, digital ink, natural notetaking, digital ink segmentation and classification framework

ACM Classification Keywords

H.5.m Information Interfaces and Presentation: Miscellaneous

INTRODUCTION

Despite the availability of advanced digital information management tools, information workers often still rely on paper-based notetaking for recording information. Unfortunately, these paper-based work practices do not integrate well with digital applications dealing with recorded information in a post-capture phase [12, 19]. Recent technological innovations, such as Anoto's digital pen and paper technology[1], enable the digitalisation of handwritten paper notes into digital ink data without an intermediary transcription step. This creates opportunities for the integration of regular paper-based notetaking practices with information management tools and services. However, a number of studies have revealed that paper notebooks are an amalgam of notes destined for multiple tasks, activities and purposes [7, 25], parts of which might not be required in digital applications [6]. Even with remarkable advances in the digital ink parsing and processing domain [16, 3, 26], digital systems are still not able to classify different notes with the same accuracy that an information worker would achieve [15].

Most existing frameworks for developing digital pen and paper applications focus on the particular case where the identification of different paper notes as well as their integration with digital applications and services is achieved by introducing specific notetaking conventions and guidelines for this new type of interactive paper interface. Through specific pen and paper interactions, such as gesture-based marks or writing within special purpose capture areas, users have to indicate which digital ink data corresponds to particular paper notes. While this transformed pen and paper use may be suitable for specific information tasks, it does not integrate well with natural notetaking [4, 6]. There is a trade-off between providing comprehensive and accurate support for the digital integration of paper-captured information and the preservation of natural notetaking practices.

We have developed a digital ink segmentation and classification framework to address various requirements of digital pen and paper-based notetaking applications. In addition to user-driven approaches for segmenting and classifying digital ink data, our solution provides support for automatic and semi-automatic digital ink data processing. Digital ink data is separated into basic blocks of ink traces analogous to blocks visually perceivable to users such as paragraphs or bullet list structures. Moreover, heuristics for the high-level processing and classification of digital ink data can be implemented and incrementally added to the framework for further processing of these basic ink data blocks. The user-driven classification of digital ink data can be combined with an automatic separation and classification process, thus providing developers further options for dealing with specific limitations of both approaches, depending on specific needs of information workers and domain-specific applications.

We start by discussing limitations of existing tools for developing notetaking applications with respect to their support for natural notetaking. Further, we describe our approach

[1] http://www.anoto.com/digital-pen-paper.aspx

EICS'11, June 13–16, 2011, Pisa, Italy.

for separating digital ink data into blocks of ink data that reflect the visual representation of paper notes. After presenting our extensible framework for digital ink processing and classification, we highlight how the digital ink processing functionality has been integrated with an existing interactive paper application development framework, before providing some concluding remarks.

BACKGROUND

Various solutions have been proposed to offer information workers the advantages of both paper and digital media for notetaking. Early attempts range from paper notebooks with a predefined page configuration where the content can be extracted through an offline scanning process, such as introduced in the Paper PDA [5], to solutions that proposed the replacement of paper with pen-based computers, including PDAs [8] or Tablet PCs [24, 25]. Anoto's digital pen and paper technology enabled new types of notetaking solutions and these are commercially available for end users in the form of Livescribe's Desktop[2] application, Oxford's Easybook[3] or solutions integrated with Logitech's io Pen[4]. Information written on regular paper that is covered with a special Anoto dot pattern is captured by the digital pen's integrated infrared camera and transmitted to a computer via a Bluetooth connection or when the pen is connected to a computer via a docking station. Once transmitted to a computer, the pen stroke data that is represented as timestamped x and y coordinates can be further processed and integrated with digital applications.

Custom digital pen and paper solutions can be developed with the aid of a series of toolkits and SDKs, including the Anoto SDK for PC applications [1], Livescribe's Platform and Desktop SDKs [2], PaperToolkit[5] [28] and iPaper [14]. The focus of these solutions is on designing interactive paper applications based on active page areas that can be associated with digital callback functions to be executed while processing the pen data. Content written inside a predefined page area is interpreted by the application logic assigned to that particular part of the page. This approach is suitable for applications such as Anoto's form-based processing of information or the annotation of presentation slides based on printed handouts as realised in the PaperPoint [18] solution. However, the strict interaction conventions required by such approaches may result in discarding the digital pen and paper technology for more natural notetaking [6].

The use of dedicated page areas for semantic content identification can be replaced by the use of gesture-based content classification. PapierCraft [10] proposed a set of gestures for both the marking of excerpts of paper-captured information and the specification of how the corresponding digital ink data should be digitally processed. General gesture recognition solutions, such as the iGesture [17] framework, provide support for defining custom gesture sets and integrating gesture recognition functionality with digital pen and paper applications. The drawback of a gesture-based classification approach is that the digital ink processing step has to be able to distinguish between the ink data representing the content and the ink data to be interpreted as gestures. Since the most reliable solution is still to give control to the notetakers themselves by providing them with some content marking mechanism that they have to use, the process is currently limited by the degree of change in natural notetaking behaviour. For example, NiCEBook [4] uses dedicated page areas that have to be touched with a digital pen before performing the pen-based gestures to mark and classify specific handwritten notes. Furthermore, to support natural or quasi-natural notetaking, the set of gestures has to be designed in such a way that its use does not constrain the notetaking process [6]. In ButterflyNet [27], even a single simple gesture command used to mark specific paper content received negative feedback due to the increase in notetaking time.

Given the apparent correlation between the lack of interaction rules and the preference for paper-based notetaking, it seems obvious that an enhanced notetaking solution should employ automatic approaches for digital ink data processing as much as possible. A body of work has pointed out that even natural notes contain an implicit organisation based on spatial relations between pen strokes and systems have been proposed that exploit these implicit note structures for interactive whiteboard systems [13], "rough" document image editors [15] and interactive notetaking systems [9, 21]. While these solutions do not always provide a correct interpretation, the interactivity of these systems enables the user to immediately observe the resulting interpretation of their actions and to intervene in the case of misinterpretations.

Extensive work on automatic digital ink data processing exists. In particular, work on detecting lines of text and distinguishing between textual and graphical handwriting seems to be relevant for a potential notetaking solution that exploits implicit structures in paper notes [16, 3, 26]. However, to the best of our knowledge, there are no solutions based on digital pen and paper technology that integrate such automatic approaches for digital ink data processing. One of the reasons might be the fact that no direct feedback can be provided based on a paper interface. Possibilities for providing feedback about a user's actions include digital pen feedback [11] or feedback via various other external devices such as smart cameras as used in ButterflyNet [27]. Even if these approaches reach a certain maturity, the continuous feedback while taking notes might not be desirable. Solutions such as PapierCraft, Paper PDA or PaperProof [23] propose a graceful degradation approach where the interpretation of a user's actions is digitally reviewed in a post-processing phase.

It becomes apparent that a digital ink data processing solution for paper-based notetaking can neither rely solely on automatic processing nor on user-driven interpretation. We think that developers should be provided with framework support for both automatic and user-driven means of processing paper-based notes and have the possibility to switch between or combine the two approaches based on particular application requirements. We will present our proposal for

[2] http://www.livescribe.com
[3] http://www.oxfordeasybook.com
[4] http://www.logitech.com
[5] http://hci.stanford.edu/research/paper/

such an extensible digital ink processing framework, show how it has been integrated with an existing interactive paper solution and present a paper-based notetaking application that was implemented based on the presented framework.

DIGITAL INK DATA PROCESSING FRAMEWORK

Previous studies on natural notetaking have revealed that it is unlikely to be able to automatically identify different note categories and how they are meant to be used digitally unless some form of user intervention is involved [6]. At the same time, it was observed that notetakers are reluctant to adapt their notetaking behaviour to include user-generated metadata about how paper notes should be digitally processed. However, current framework support for developing digital pen and paper applications relies heavily on approaches for clustering and further processing digital ink data that introduce changes in the natural notetaking behaviour. Users are required to either write specific notes within designated page areas or mark them with pen-based gestures chosen by the developers based on certain observed notetaking patterns. All notes written in a predefined area or delimited in some way by one or more ink gestures will result in a digital ink cluster that is processed in a unitary manner defined by the developer in a digital callback function.

Works such as Ispas et al. [6], Wattenberg and Fisher [22] or Li et al. [9] have pointed out that handwritten notes represent a number of structured elements such as sketches or aggregations of text lines in the form of paragraphs or bullet lists. Such structured elements could potentially be automatically identified and extracted by clustering digital ink data based on spatial and temporal proximity. We propose that current software support for developing digital pen and paper-based applications should be extended to provide access to and means of manipulating such automatically extracted structures. For particular application domains, this could relax imposed notetaking conventions or even introduce an alternative to relying on multiple page areas or gestures and the assumption that users will use defined rules while taking notes. For example, a natural notetaking application that only requires access to note structures at the granularity level of paragraphs could rely on a paper-based interface with just one page area defined for each page. However, this approach requires more effort from developers since more sophisticated functionality able to identify and handle the structures within notes captured from a single page area needs to be provided inside associated callback functions. Certain automatic or semi-automatic approaches for the digital processing of handwritten information at the level of basic structures may be useful even for application domains that might require more refined access to the content of notes. Our goal is to provide means to combine user-driven approaches with automatically extracted structures to enable more flexible paper-based interfaces.

Figure 1 shows the paper-based user interface of a sample notetaking application. Except for a timeline positioned at the bottom, the page has the appearance of a regular notebook page with a writing area covering the remaining part. In the writing area, notetakers can take freehand notes. In addition, they can use the timeline to indicate that they want to be reminded by the digital notetaking application about particular notes within a certain period of time. The approach to mark handwritten notes implemented by the developer could for example consist of requiring notetakers to first mark the timeframe by touching the corresponding part of the timeline with the pen (1) and then select the notes by drawing a vertical line (2). As a result, all note structures located adjacent to the vertical pen stroke are combined into a higher level structure which is associated with the corresponding temporal metadata.

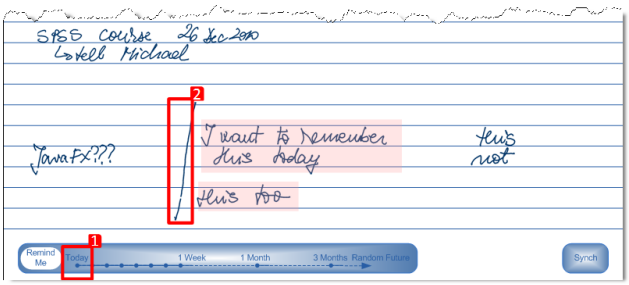


Figure 1. Notetaking application interface

Given that the support for extracting note structures is in place, developers may deal with handwritten notes at the level of the concepts shown in Figure 2. For a writing area represented by a `NotebookPage`, a developer can access its structural elements represented as `PageElement` instances. The `BasicElement` specialisation of the `PageElement` class represents the most basic entity that can be generated by a given digital ink data segmentation algorithm. Therefore, `BasicElement` instances aggregate low level ink data that has been assigned to single structures by the segmentation algorithm.

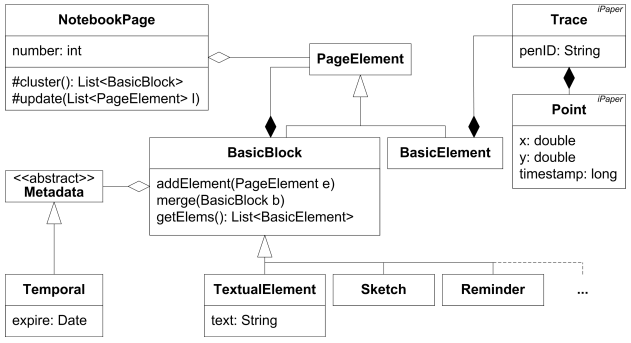


Figure 2. Notebook page class diagram

Typical representations for the digital ink data consist of providing a model for the handwritten strokes. We use here a representation based on `Traces` of timestamped `Points` which is the representation introduced by the iPaper framework [14]. The digital ink representations used by other digital pen and paper frameworks can easily be transformed to the iPaper format, for example, by using the InkML[6] representation as an intermediary format.

[6]http://www.w3.org/2002/mmi/ink

Multiple `BasicElement`s positioned close to each other can be grouped forming `BasicBlock`s that represent collections of note elements. By default, page elements are grouped based on containment and intersection relationships between their bounding boxes. Custom heuristics for the grouping of note structures can furthermore be specified by overriding the protected `cluster()` method. For example, two `BasicElement`s corresponding to two lines of handwritten notes could be grouped into a single block if the temporal gap between their last and respectively first digital ink data strokes fits into a certain interval identified as the time for a notetaker to switch to the next line. The method is invoked by default after the note segmentation step. A `BasicBlock` can be further specialised into various semantic structures such as the `TextualElement`, `Sketch` or `Reminder` classes. For this purpose, various heuristics for processing digital ink data at block level have to be implemented. For example, blocks that can be parsed into digital text can be classified as `TextualElement`. On the other hand, a `BasicBlock` consisting of one or several `PageElement`s for which the handwriting recognition does not produce any accurate results might be classified as `Sketch`.

In addition to automatic approaches for detecting note structures, notetakers are also given the possibility to mark blocks of notes to be treated as a single semantic structure during the processing step as explained earlier when describing the paper interface in Figure 1. As opposed to automatically generated blocks of notes, user-specified blocks are associated with different `Metadata` classes such as the `Temporal` class in Figure 2 that could be used for our example to represent information about notes that users need to be reminded about at a specific time in the future.

Automatic Digital Ink Data Segmentation

In a previous study of notetaking practices [6], we identified three major types of note structures: paragraphs, bullet lists and sketches. For the first two types of structures, API support for manipulating individual lines of text may be useful and therefore we decided to take a bottom-up approach consisting of first extracting lines from the handwritten information and then grouping individual lines into blocks of notes according to their spatial relationships. As mentioned before, several approaches have been proposed for the clustering problem. Our implementation is based on the work of Ao et al. [3]. The authors present a technique for identifying textual lines based on the notion of a *link model*. Furthermore, they propose a solution for distinguishing between textual and graphical information. Inspired by their suggestions, we have chosen to identify sketch classes of note blocks based on the fact that the handwriting recognition engine does not return valid results.

According to the link model, a set of blocks composed of tightly connected pen strokes belong to the same textual line if their corresponding bounding boxes are located close to each other in a linear way and have comparable sizes. The three criteria are translated into measures applied to all segments formed between the centre points of all adjacent bounding boxes, called *links*. The links between the bounding boxes are identified by applying Delaunay triangulation.

In our digital ink data segmentation, we start with a set of note traces represented by the set of coordinates between successive pen down and pen up actions. As described in Ao et al., traces are first merged into blocks based on the timestamp information associated with each trace. Further, the minimum bounding boxes of the constructed blocks are computed. The minimum bounding box provides additional information about its associated trace such as the rotation with respect to the x-axis. Our computation of the minimum bounding boxes is based on rotating calipers [20] that are applied to the convex hull of a trace's points. For the Delaunay triangulation, we use the implementation of Paul Chew[7]. The three text line criteria μ_1, μ_2 and μ_3 for the closeness, linearity and similarity in size are implemented as follows:

Closeness

We consider that the measure μ_1 of the closeness of two bounding boxes is inversely proportional to the length l of the link between them:

$$\mu_1 = \frac{1}{l+1} \tag{1}$$

This results in $0 \leq \mu_1 \leq 1$. Note that we add 1 to the denominator to avoid division by zero in the situation where the two bounding boxes have the same centre point. In this case, we get the maximum value of 1 whereas a value of 0 results for $l \rightarrow \infty$.

Linearity

Given the two angles α_1 and α_2 representing the rotation of the two bounding boxes relative to their link, we define the linearity μ_2 of two bounding boxes as follows:

$$\mu_2 = \frac{1}{\mid \alpha_1 - \alpha_2 \mid +1} \tag{2}$$

For $\alpha_1 = \alpha_2$ the corresponding traces are positioned linearly.

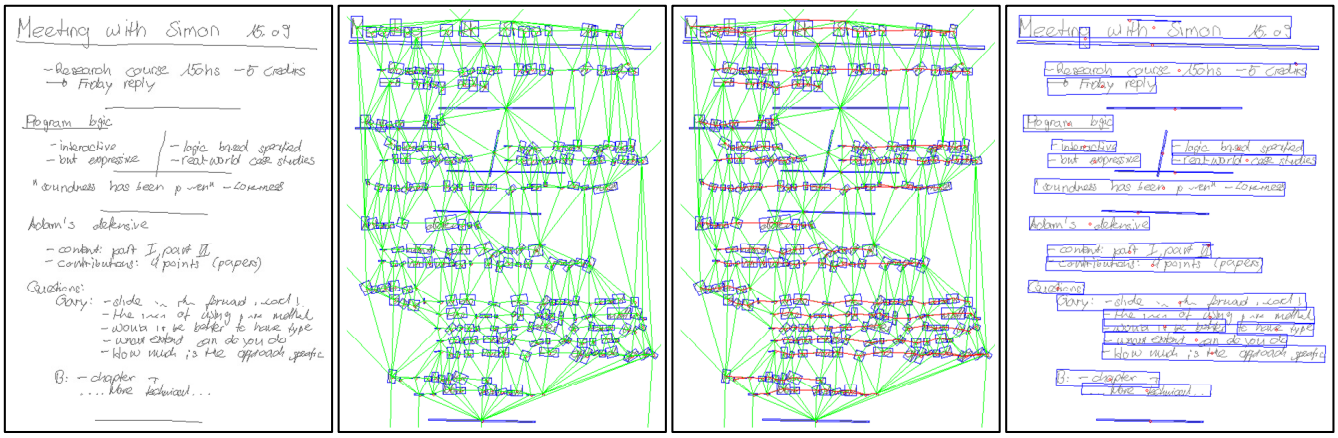
Comparable sizes

To verify that two bounding boxes have similar sizes, we compute the value μ_3 based on their corresponding areas:

$$\mu_3 = \frac{1}{\mid A_1 - A_2 \mid +1} \tag{3}$$

The areas of the two bounding boxes are represented by A_1 and A_2, respectively.

[7] http://www.cs.cornell.edu/info/people/chew/Delaunay.html



(a) Original notes (b) Delauney triangulation (c) Strongest links (d) Line segmentation

Figure 3. Successive line segmentation steps

Link strength

The link strength μ is computed based on the three criteria μ_1, μ_2 and μ_3 and is used in the detection of textual lines:

$$\mu = \mu_1 + \mu_2 + \mu_3 \tag{4}$$

Given the original note page shown in Figure 3(a), the final result of the line segmentation is highlighted in Figure 3(d). In an intermediary step, we first apply Delaunay triangulation for the minimum bounding box computation shown in Figure 3(b) and then identify the strongest links shown in Figure 3(c).

The separation into basic page elements, which is equivalent to extracting lines in the case of the algorithm proposed by Ao et al., is provided by the `separate()` method of the `Separator` class shown in Figure 4. Every time some digital ink data is provided, for example by the iPaper framework in the form of a `Note` containing a number of traces in combination with information about the page number and the document and pen identifiers, this data is added to the `Separator` class. The invocation of the `separate()` method results in a list of `BasicElements` introduced earlier in Figure 1. The `cluster()` method that is invoked after the segmentation step generates a list of `BasicBlock` instances for each `NotebookPage`.

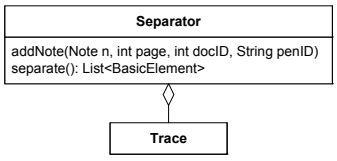


Figure 4. Class diagram for separation into basic elements

Listing 1 shows how the segmentation process was integrated with the iPaper framework for the sample notetaking application. iPaper provides support for defining active page areas and associating active components to be invoked when ink written within an area needs to be processed. The `DrawAreaStub` active component's `handleNote()` method contains the code to be invoked when processing notes captured from the main writing area of the application. When the user triggers the explicit synchronisation of their notes with the computer by touching the printed `Synch` button shown at the bottom right of Figure 1, the `SynchButtonStub` active component is instantiated. As illustrated in the listing, the `addNote()` method of the `Separator` class is invoked every time a new note has been captured. At synchronisation time, the `separate()` method of a `Separator` class is invoked. Existing `NotebookPage` instances are then refreshed via the `update()` method to reflect the new document structure after the last separation step.

Listing 1. iPaper active components for the sample application

```
1  public class DrawAreaStub extends CaptureNoteStub {
2    public void handleNote(String docID, int page, Note n) {
3      String penID = getDeviceAddress();
4      Separator s = Separator.getInstance();
5      s.addNote(n, page, docID, penID);
6    }
7  }
8
9  public class SynchButtonStub extends SingleEventStub {
10   public void finish() {
11     Separator s = Separator.getInstance();
12     // redo the separation and clustering for all pages
13     List<BasicElement> basicElements = s.separate();
14     List<BasicBlock> blocks = notebookPage.cluster();
15     // add all blocks and basic elements into an update list
16     List<PageElement> elements = ...
17     notebookPage.update(elements);
18   }
19 }
```

User-Driven Segmentation

In the case of the notetaking application interface described in Figure 1, we proposed an interaction model consisting of two successive pen-based interactions through which note-

takers could provide guidelines for processing specific notes subsets. The first interaction step consists of pointing with the pen to a specific part of the printed timeline. This action is interpreted as an upper bound value for the lifetime of the reminder notes which are about to be selected in a second step. The system will interpret the selected notes as forming part of a single block structure containing one or several basic elements according to the length of the vertical line used to mark notes. The functionality is an example where traditional note processing support provided by a digital pen and paper application is used for manual user-driven segmentation and classification of a page's structural elements. For the classification, various metadata specified at the design time of the paper interface and associated with specific active page areas will generate the corresponding block metadata.

To support developers in designing paper-based interfaces with a user-driven classification of free-form notes, we propose a framework for the description of paper-based interaction models based on finite state machines (FSM). Figure 5 shows the class diagram of our solution for the specification of interaction models based on an FSM. The framework provides an extensible set of `States`, each of which has a unique name. Based on a given `Action` of type `GESTURE`, `TEXT`, `MESSAGE` or `TRACE`, the FSM gets into a new state via the `transition()` method. Each state can be configured with a method to be invoked and executed whenever the state is reached by using the `setInvokeMethod()` method. For each `Action`, an optional data value represented by the `data` field of a parametric type `T` can be configured. This allows the configuration of each action type with custom values such as the gesture class that has been recognised in combination with an action of type `GESTURE` or the text associated with an action of type `TEXT`.

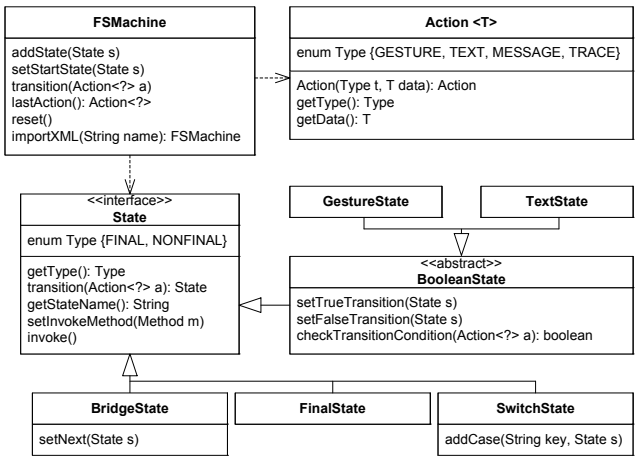


Figure 5. Class diagram for the finite state machine framework

Particular FSMs can be defined by a developer via an XML specification. Listing 2 shows the XML definition of a series of state types that are already provided by our framework.

Listing 2. XML representation of predefined state types

```
1  <machine>
2    <state type="bridge" name="...">
3      <next state="..." />
4    </state>
5    <state type="final" name="..." />
6    <state type="gesture" name="..." />
7      <condition className="..." />
8      <true state="..." />
9      <false state="..." />
10   </state>
11   <state type="text" name="..." />
12     <condition text="..." />
13     <true state="..." />
14     <false state="..." />
15   </state>
16   <state type="switch" name="..." />
17     <case key="default" state="..." />
18     <case key="..." state="..." />
19     ...
20   </state>
21 </machine>
```

- *Bridge state*: From this type of node, the FSM will always transition to the state specified as `next`, regardless of the input action.

- *Final state*: When the FSM reaches a final node, it will continue its execution at the node that was marked as a start node as soon as the FSM receives a new input action.

- *Gesture state*: The FSM will transition from a gesture node to the state denoted by `true` when the `condition` is met. In all other cases, it will transition to the state denoted by `false`. The condition is met when the type of the input action is `GESTURE` and if the action's `data` field contains the value denoted by the `className` attribute.

- *Text state*: The text node is similar to a gesture node, with the difference that the input action type must be `TEXT`. In this case, the `data` field must contain the string value defined in the `text` attribute.

- *Switch state*: A switch state may have several possible outgoing transitions. It requires an input action of type `MESSAGE` and the `data` field must be assigned a value equal to one of the `key` attributes of the `case` elements. When none of the keys match the passed value, the transition marked by the default key is followed.

The FSM for our sample notetaking application is shown in Figure 6 and the corresponding XML code in Listing 3. The code for the iPaper active components presented previously has to be adapted to account for controlling the FSM as shown in Listing 4. Note that a third active component is associated with the different parts of the timeline. Every time the user touches the timeline with the pen, an instance of the active component will infer the temporal metadata value associated with the page area and create an `Action` of type `MESSAGE` configured with a "temporal" string value for its `data` field. Similarly, the `SynchButtonStub` active component creates an `Action` of type `MESSAGE`, but configured with a specific "synch" string value. No additional data is required in the `data` field of an `Action` of type `Trace`, an instance of which is created by the `DrawAreaStub` active component.

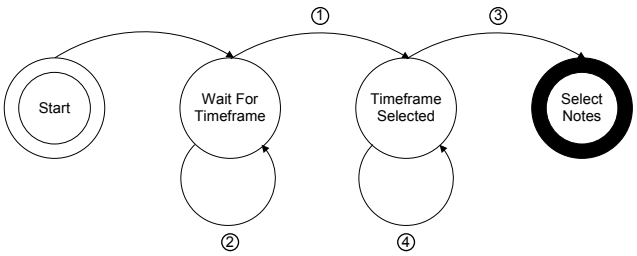


Figure 6. Finite state machine for sample notetaking application

In this case, the separation is managed through method calls specified in the XML description of the FSM. As long as a user takes notes in the main writing area, the FSM remains in the *Wait For Timeframe* state as shown by transition (2) and notes have to be added to the buffer maintained by the `Separator` class as shown by the `invoke` XML element. The `addNote()` method defined in a custom class, represented by the `Logic` class in Listing 3, is responsible for forwarding the call to the previously mentioned `addNote()` method of the `Separator` class. When the user selects a timeframe with the pen, transition (1) into the *Timeframe Selected* state is triggered. From this state, transition (3) is taken only if a user selects a group of notes by marking them with the digital pen. Otherwise, the FSM does not leave the state as indicated by transition (4). The `selectNotes()` method invoked after transition (3) is responsible for creating a new `BasicBlock` associated with the corresponding temporal metadata. Since new content has been added after the last segmentation, a new page separation into basic page elements is done before updating the block structure of the page. Further, `BasicElements` located next to the selecting vertical pen stroke (a single trace) will be grouped into a new block.

Listing 3. XML representation of a sample FSM

```
<machine>
  <state type="switch" name="WaitForTimeframe"
    start="true">
    <case key="default" state="WaitForTimeframe" />
    <case key="temporal" state="TimeframeSelected" />
    <invoke class="org.paperNotesManager.Logic"
      method="addNote" />
  </state>
  <state type="switch" name="TimeframeSelected">
    <case key="default" state="SelectNotes" />
    <case key="temporal" state="TimeframeSelected" />
    <invoke class="org.paperNotesManager.Logic"
      method="addNote" />
  </state>
  <state type="final" name="SelectNotes">
    <invoke class="org.paperNotesManager.Logic"
      method="selectNotes" />
  </state>
</machine>
```

In this particular case, the last pen stroke written in the main writing area will be used for the selection of the page elements that have to be associated with specific metadata, independently of their content or shape. Another possibility would be to perform the selection only if users draw a specific gesture or write a specific keyword immediately after defining the timeframe. For this purpose, a gesture or a text state could be introduced in the specified FSM after the *Timeframe Selected* state.

Listing 4. Adapted iPaper active components specification

```
public class DrawAreaStub extends CaptureNoteStub {
  public void handleNote(String docID, int page, Note n) {
    Action<String> action = new Action<String>(
      Action.Type.TRACE, null);
    Logic.getFSM().transistion(action);
  }
}

public class TemporalMarkerStub extends SingleEventStub {
  public void finish() {
    Date date = getTemporalMarker();
    Logic.addMetadata(new TemporalMetadata(date));
    Action<String> action = new Action<String>(
      Action.Type.MESSAGE, "temporal");
    Logic.getFSM().transistion(action);
  }
}

public class SynchButtonStub extends SingleEventStub {
  public void finish() {
    Action<String> action = new Action<String>(
      Action.Type.MESSAGE, "synch");
    Logic.getFSM().transistion(action);
  }
}
```

Custom Classification

The segmentation into basic page elements and the subsequent clustering into basic blocks reveals some details about the high-level structure of handwritten notes. However, this grouping into basic blocks only reflects spatial and temporal properties of the different notes and content is not taken into consideration. In this section, we therefore highlight how developers can further process the existing basic structures based on the classification components shown in Figure 7.

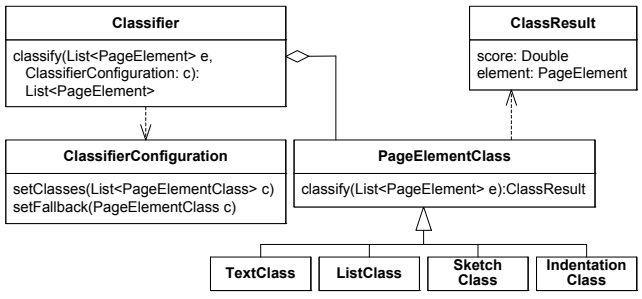


Figure 7. Classifier

Given a set of `PageElementClasses`, the `classify()` method defined in the `Classifier` class will analyse a collection of `PageElement` instances provided as input and create more specific `PageElement` subclasses. Each classification class returns a `ClassResult` instance consisting of a confidence score ($0 \leq score \leq 1$) for the membership of a given `PageElement` in a specific classification. The classifier generates this score for each of the classes

configured in the `ClassifierConfiguration` and recommends the classification result with the highest score. The score has to be higher than a certain threshold, which in our implementation was set to 0.5. If there is no possible classification with a score higher than the threshold, the `Classifier` will return a fallback classification class as specified in a classifier's configuration. The basic configuration of a classifier is presented in Listing 5.

Listing 5. Basic configuration for the classification

```
1  List<PageElement> originalElements = ...
2
3  // set up the classes to be checked against
4  List<PageElementClass> classes =
5     new LinkedList<PageElementClass>();
6  classes.add(new ExampleClass());
7
8  // set the fallback class to SketchClass
9  PageElementClass fallback = new SketchClass();
10 ClassifierConfiguration config =
11    new ClassifierConfiguration(classes, fallback);
12
13 // classify the input elements
14 Classifier c = new Classifier();
15 List<PageElement> classifiedElements =
16    c.classify(originalElements, config);
```

Our digital ink processing framework offers an implementation for some default classifiers:

Text Class

The `TextClass` is based on the output of a handwriting recognition algorithm and the score is directly related to the confidence value of the used handwriting recognition engine. In our implementation, we used the MyScript Intelligent Character Recognition from VisionObjects[8] for the handwriting recognition.

Sketch Class

The `SketchClass` is meant to be used as a fallback class with a classification score of 1. Typically, the classifier will generate a `Sketch` element if every other configured classification fails.

Indentation Class

The `IdentationClass` verifies whether a `BasicBlock` that contains several `PageElement` instances represents a multi-level bullet list structure. In a first step, the indentation level of each basic element contained within a block is determined. Page elements are sorted in ascending order based on the upper left corner of their bounding box. The ordering is done first for the x coordinate and then for the y coordinate. After the ordering, the corner points are iterated over and it is checked whether the difference to the previous anchor point is greater than some threshold. The threshold value is used to define what is to be interpreted as simple white space and what has to be considered list item indentation. Depending on the result of the comparison, an element's indentation level is set to the same value as the indentation level of the previous element or to a level increased by 1. Figure 8 shows the result of such a procedure. Finally, the score of the classification is set to a value equal to the average computed for the confidence levels returned by the handwriting recognition engine for each of the structural elements.

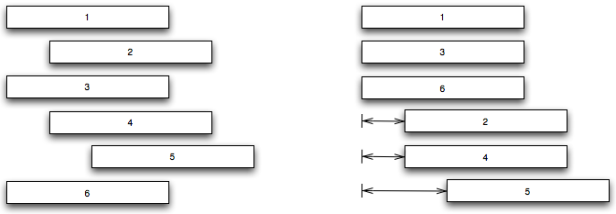


Figure 8. Determining the indentation of a set of lines

List Class

The `ListClass` can be seen as a specialisation of the indentation class where the subelements represent an item list on a single indentation level, each of them starting with a given token.

To give an idea about the effort required for implementing classification classes, we refer to Listing 6. Given a set of `PageElements`, the class creates a block classified as `Title` if all page elements can be parsed into text with a certain confidence and the element placed below all other elements is a straight line having a length comparable to or bigger than the length of all other elements placed above. An example of such a title element is shown at the top of the page in Figure 3(a). The score of the class can be computed, for example, by averaging the score of the handwriting recognition with the score from the straight line detector, the score of the straightness being given a higher importance.

Listing 6. Custom title classification

```
1  public class TitleClass implements PageElementClass {
2     ...
3     @override
4     public ClassResult classify(List<PageElements> e) {
5        double score = 0.0;
6        // get the lowest element
7        PageElement last = getLowestElement(e);
8
9        // is the last element long?
10       if (isStraightLineLong(e, last)) {
11          // is the last element a straight line?
12          score = scoreStraightLine(last);
13       }
14
15       // get text representation without the line
16       e.remove(last);
17       Result hwrResult = parseText(e);
18
19       // construct the classification result
20       ClassResult r = new ClassResult();
21       r.score = (score*2 + hwrResult.getConfidence()) / 3.0;
22
23       // create a new title page element
24       e.add(last);
25       r.element = new Title(e, hwrResult.getText());
26       return r;
27    }
28 }
```

[8] http://www.visionobjects.com/

It can be seen that developers are provided with the possibility of working at block level and simple computations such as determining relationships between the bounding boxes of the various elements already provide relatively powerful results. Furthermore, digital ink processing operations at page element level, as in the case of the list class presented earlier, can reveal further classification possibilities. However, the ink processing effort is restricted to parts of the notes.

Framework-based Application Development

Existing frameworks for digital pen and paper application development focus on the design of the paper-based interface of a particular solution in terms of active page areas and marking gestures. In addition to imposing changes on natural notetaking practices, this also leads to less flexibility in developing the digital counterpart of the solution since the latter becomes bound to the first. The structure of the paper interface determines the segmentation of the digital ink data and, subsequently, the level of granularity at which developers can access and handle digitally handwritten information. In the case that access at lower granularity levels is needed, developers are required to implement the segmentation of digital ink data. Furthermore, changing the paper interface leads to the necessity to also implement changes in the digital counterpart of the application.

Our framework provides access to basic note structures captured from a single page area and supports the further grouping of elementary structures based on custom heuristics. The basic or composed structures can be further classified based on an extensible set of heuristics. The benefits of our solution are twofold. First, the framework facilitates the handling of digital ink data, which reduces the amount of required low level digital ink processing by developers and enables them to focus on the GUI of an information management application and the rendering of digital ink data represented as `PageElements`. Second, rather than relying on complicated paper interfaces to enforce the digital ink data processing, parts of the processing can be shifted to the post-capture phase. In addition to improving the flexibility of the development process, applications may rely on less complicated paper interfaces, further relaxing changes imposed on natural notetaking.

Figure 9 highlights two applications that have been realised based on our digital ink processing framework and integrated with the paper interface described in Figure 1. Paragraphs and bullet lists of notes written in the main writing area of a page are extracted through an automatic segmentation process and integrated with a to-do list application or presented as post-it notes on the digital desktop. Notes can be presented in handwritten form or users can also switch to a version processed by a handwriting recognition engine. If some of the notes are associated with temporal metadata through user-driven segmentation, the corresponding post-its or to-do list items will have their due date automatically set based on the value selected by the user via the paper interface. Instead of having to deal with the low level processing of ink data, the main task of the developer of these applications was to implement the appropriate set of functionalities required by the end user, such as support for updating the captured information or configuring user interface elements.

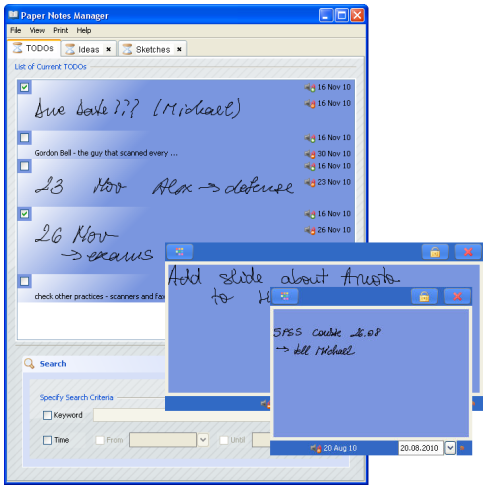


Figure 9. Paper notes as to-do list and post-it reminders

CONCLUSIONS

We have presented a digital ink segmentation and classification framework that significantly simplifies the development of digital pen and paper-based notetaking applications. The framework provides access to the main structural elements of handwritten notes made available after a segmentation process, support to further group such elementary structures and an extensible set of classification heuristics for digital ink data. This allows an application developer to focus on the aspects of presenting different classes of notes as part of digital applications rather than having to deal with the low level processing of digital ink data. In addition, we have presented a solution for the definition of paper-based interaction models based on finite state machines, simplifying the complex note processing and interactive paper interface definition. Last but not least, we have introduced two digital pen and paper-based notetaking solutions that have been realised based on the presented digital ink processing framework.

REFERENCES

1. Anoto SDK for PC Applications, June 2010. V 3.3.2.0.
2. Livescribe Desktop SDK, June 2010. V 0.7.0.
3. X. Ao, J. Li, X. Wang, and G. Dai. Structuralizing Digital Ink for Efficient Selection. In *Proc. of Conf. on Intelligent User Interfaces (IUI '06)*, Sydney, Australia, January 2006.
4. P. Brandl, C. Richter, and M. Haller. NiCEBook - Supporting Natural Note Taking. In *Proc. of Conf. on Human Factors in Computing Systems (CHI '10)*, Atlanta, USA, April 2010.
5. J. M. Heiner, S. E. Hudson, and K. Tanaka. Linking and Messaging from Real Paper in the Paper PDA. In *Proc. of Symposium on User Interface Software and Technology (UIST '99)*, Asheville, USA, November 1999.

6. A. Ispas, B. Signer, and M. C. Norrie. A Study and Design Implications for Incidental Notetaking with Digital Pen and Paper Technologies. In *Proc. of BCS Conf. on Human Computer Interaction (HCI '10)*, Dundee, Scotland, September 2010.

7. F. Khan. A Survey of Note-Taking Practices. Technical Report HPL-93-107, HP Laboratories Bristol, December 1993.

8. J. A. Landay and R. C. Davis. Making Sharing Pervasive: Ubiquitous Computing for Shared Note Taking. *IBM Systems Journal*, 38(4):531–550, 1999.

9. Y. Li, Z. Guan, H. Wang, G. Dai, and X. Ren. Structuralizing Freeform Notes by Implicit Sketch Understanding. In *AAAI Spring Symposium on Sketch Understanding SS-02-08*, Palo Alto, USA, March 2002.

10. C. Liao, F. Guimbretière, and K. Hinckley. PapierCraft: A Command System for Interactive Paper. In *Proc. of Symposium on User Interface Software and Technology (UIST '05)*, Seattle, USA, October 2005.

11. C. Liao, F. Guimbretière, and C. E. Loeckenhoff. Pen-top Feedback for Paper-based Interfaces. In *Proc. of Symposium on User Interface Software and Technology (UIST '06)*, Montreux, Switzerland, October 2006.

12. M. Lin, W. G. Lutters, and T. S. Kim. Understanding the Micronote Lifecycle: Improving Mobile Support for Informal Note Taking. In *Proc. of Conf. on Human Factors in Computing Systems (CHI '04)*, Vienna, Austria, April 2004.

13. T. P. Moran, P. Chiu, W. van Melle, and G. Kurtenbach. Implicit Structure for Pen-based Systems within a Freeform Interaction Paradigm. In *Proc. of Conf. on Human Factors in Computing Systems (CHI '95)*, Denver, USA, May 1995.

14. M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proc. of Workshop on Collaborating over Paper and Digital Documents (CoPADD '06)*, Banff, Canada, November 2006.

15. E. Saund, D. Fleet, D. Larner, and J. Mahoney. Perceptually-supported Image Editing of Text and Graphics. In *Proc. of Symposium on User Interface Software and Technology (UIST '03)*, Vancouver, Canada, November 2003.

16. M. Shilman, Z. Wei, S. Raghupathy, P. Simard, and D. Jones. Discerning Structure from Freeform Handwritten Notes. In *Proc. of Conf. on Document Analysis and Recognition (ICDAR '03)*, Edinburgh, Scotland, August 2003.

17. B. Signer, U. Kurmann, and M. C. Norrie. iGesture: A General Gesture Recognition Framework. In *Proc. of Conf. on Document Analysis and Recognition (ICDAR '07)*, Curitiba, Brazil, September 2007.

18. B. Signer and M. C. Norrie. PaperPoint: A Paper-based Presentation and Interactive Paper Prototyping Tool. In *Proc. of Conf. on Tangible, Embedded and Embodied Interaction (TEI '07)*, Baton Rouge, USA, February 2007.

19. A. Tabard, W. E. Mackay, and E. Eastmond. From Individual to Collaborative: The Evolution of Prism, a Hybrid Laboratory Notebook. In *Proc. of Conf. on Computer Supported Cooperative Work (CSCW '08)*, San Diego, USA, November 2008.

20. G. Toussaint. Solving Geometric Problems with the Rotating Calipers. In *Proc. of Mediterranean Electrotechnical Conf. (MELECON '83)*, Athens, Greece, May 1983.

21. T. Wang and B. Plimmer. SmartList: Exploring Intelligent Hand-written List Support. In *Proc. of Conf. of NZ ACM Special Interest Group on Human-Computer Interaction (CHINZ '09)*, Auckland, New Zealand, July 2009.

22. M. Wattenberg and D. Fisher. A Model of Multi-scale Perceptual Organization in Information Graphics. In *Proc. of Symposium on Information Visualization (INFOVIS '03)*, Seattle, USA, October 2003.

23. N. Weibel, A. Ispas, B. Signer, and M. C. Norrie. PaperProof: A Paper-digital Proof-editing System. In *Proc. of Conf on Human Factors in Computing Systems (Extended Abstracts) (CHI '08)*, Florence, Italy, April 2008.

24. S. Whittaker, P. Hyland, and M. Wiley. FILOCHAT: Handwritten Notes Provide Access to Recorded Conversations. In *Proc. of Conf. on Human Factors in Computing Systems (CHI '94)*, Boston, USA, April 1994.

25. L. D. Wilcox, B. N. Schilit, and N. Sawhney. Dynomite: A Dynamically Organized Ink and Audio Notebook. In *Proc. of Conf. on Human Factors in Computing Systems (CHI '97)*, Atlanta, USA, March 1997.

26. M. Ye, , P. Viola, S. Raghupathy, H. Sutanto, and C. Li. Learning to Group Text Lines and Regions in Freeform Handwritten Notes. In *Proc. of Conf. on Document Analysis and Recognition (ICDAR '07)*, Curitiba, Brazil, September 2007.

27. R. Yeh, C. Liao, S. Klemmer, F. Guimbretière, B. Lee, B. Kakaradov, J. Stamberger, and A. Paepcke. ButterflyNet: a Mobile Capture and Access System for Field Biology Research. In *Proc. of Conf. on Human Factors in Computing Systems (CHI '06)*, Montréal, Canada, April 2006.

28. R. B. Yeh, A. Paepcke, and S. R. Klemmer. Iterative Design and Evaluation of an Event Architecture for Pen-and-paper Interfaces. In *Proc. of Symposium on User Interface Software and Technology (UIST '08)*, Monterey, USA, October 2008.