

# PaperProof: A Paper-Digital Proof-Editing System

Nadir Weibel, Beat Signer, Patrick Ponti, Moira C. Norrie  
Institute for Information Systems, ETH Zurich  
CH-8092 Zurich, Switzerland  
{weibel,signer,norrie}@inf.ethz.ch

## ABSTRACT

Recent approaches for linking paper and digital information or services tend to be based on a one-time publishing of digital information where changes to the printed document become isolated from its digital source. Structural information which is available when authoring a digital document is lost during the printing process making it difficult to map interactions within the physical document to the corresponding elements of the digital document. We identify the necessary requirements for an integrated digital and paper-based document lifecycle and present our solution which supports a seamless transition between digital and physical document instances. PaperProof is presented as a paper-based proof-editing application that exploits our new approach for mapping pen-based interactions with paper documents to the corresponding operations in the digital document instance.

## 1. INTRODUCTION

Recent developments in the area of interactive paper in terms of enabling technologies, supporting infrastructures and applications have demonstrated the potential benefits arising from an integration of paper with digital services and information. Multiple projects have focused on paper-based physical interfaces for digital services where actions on paper documents are mapped to their digital counterparts.

The realisation of paper-digital document integration normally involves two main steps. First, a digital device has to bridge the gap between a physical paper document and the digital world. Nowadays, there exist several technologies that enable the linking from printed documents to digital information including digital cameras, ultrasonic receivers, magnetic field sensors, barcode readers, RFID antennas and Anoto's Digital Pen and Paper technology [2]. Regardless of the hardware solution used, the general approach is to couple a physical object with an identifier—a Globally Unique Identifier (GUID) in the case of barcodes and RFID tags or areas for ultrasonic positioning technologies, magnetic receivers and Anoto-based devices—and transmit that information to a computing device. Second, one needs to define a link between a user action on paper and the corresponding digital service. The information available on the physical paper document should therefore be captured and translated into a digital language.

A variety of research projects and commercial products have investigated how to enable this kind of paper-driven digital services. Wellner's Digital Desk [30] is a camera-based approach, while the Wacom Graphics Tablets [28] make use of a weak magnetic field applied to the writing surface and the mimio Xi [13] is a product based on ultrasonic positioning. Barcodes have been used in a range of projects in the past, while RFIDs tags are quite common in more recent ubiquitous computing applications such as the mediaBlocks project [25]. The Anoto technology seems to be more flexible and may be easily integrated into the paper document space since it is essentially based on regular paper and a digital pen. In the last few years, multiple projects and applications have been realised based on Anoto technology including PADD [6], ButterflyNet [31], Hitachi iJITinLab [7], EdFest [15], PaperPoint [24] and Print-n-Link [17].

However, so far, most of the approaches for linking paper and digital documents tend to be “one-way solutions” which means that they generate interactive paper documents from a digital instance rather than enabling seamless transitions back and forth between the digital and paper representations of a document. The main focus is on the interaction from a paper document to some sort of digital service, neglecting the information which was available during the authoring of the digital document instance. If we take into account the authoring process, it is easy to realise that a large amount of structural and possibly also semantic information about the content of the document is available within the authoring tool. At the time of printing, information about the structure of the document as well as the different elements that the document consists of (paragraphs, images, tables, etc.) is usually lost when the document is transformed into a graphical representation without any structural metadata. To give an example, this research paper was authored using a  $\text{\LaTeX}$  editor and transformed into a PDF document which could then be printed on paper. The important thing to note is that from a printed version of this document we no longer have access to the explicit structural information present in the  $\text{\LaTeX}$  authoring tool.

Therefore, the linking between paper and digital documents is often bound to the concept of areas only and does not provide easy access to elements within the digital document that the user actually interacted with on paper through actions such as underlining or circling

(e.g. a word or a paragraph). This capability would require links between the physical and digital document instances based not only on IDs or the physical (x,y) position of elements within the printed document, but also based on structural information available in the source document. By having such a mapping back to the original structural elements, we could select single logical elements in a printed document and perform actions on the corresponding digital counterpart.

In previous work, we defined a model for the seamless mapping between a digital document instance and its printout [29]. Based on this model, we developed the iDoc framework which is capable of combining the metadata from both digital and printed documents, thereby enabling a true two-way interaction. In this paper, we present PaperProof, an interactive paper application based on the open source OpenOffice authoring tool [18]. PaperProof exploits the features of the defined model and provides a real-world application of the iDoc infrastructure. It proposes a solution for the common problem of automatically integrating corrections and annotations done during the paper-based proof-reading of a document. A generic approach to dynamically map a physical position on paper to elements within the original digital document instance is proposed and the infrastructure capable of managing the physical representation of digital documents on paper is outlined.

In Section 2, we motivate the need for an integrated paper-digital proof-editing system and present different existing approaches for mapping between paper and digital document instances. Section 3 introduces the requirements for paper-digital document representations and the corresponding interactive proof-editing system. In this section, we also classify existing approaches based on these requirements. In Section 4, we present our vision for an integrated paper-digital document lifecycle. Sections 5 and 6 outline our approach for binding a digital document to its paper versions and the mapping from paper back to the digital document. The PaperProof proof-editing application that validates our paper-digital mapping solution is presented in Section 7. Some details about the implementation of PaperProof are given in Section 8, while Section 9 compares it to existing approaches. Finally, concluding remarks as well as some comments about future extensions are provided in Section 10.

## 2. BACKGROUND

By observing how people work while creating documents, we can identify three main phases: i) the authoring phase, which normally is performed on a computer and results in a first draft of the digital document, ii) the proof-reading phase which is often done on a printed paper version and iii) the editing phase where annotations from a printout are transferred back to the digital source document. During the proof-reading phase, two types of handwritten markup are common—editing instructions and comments. The phases are often iterative and concurrent. This means that, in the extreme case, multiple “reviewers” may be annotating different paper copies of the same document concurrently, while one or more authors continue to work in parallel on dig-

ital versions of the document. In this case, the process of inserting annotations and corrections into the digital document becomes even more difficult since the author has to deal with corrections from multiple users which may also overlap. There are cases, especially in a concurrent scenario, where digital aids such as a change tracking mechanism offered by an authoring tool might be used. Even if these tools are only bound to the digital document and have no connection with the paper instances, they share the concept of annotations as a building block of the proof-reading system.

Document annotations have been extensively analysed in the past and their role in increasing the value of the source document has been recognised. As outlined by Marshall [11], different systems have been implemented to support the annotation process of digital information. Sellen and Harper [21] point out how the process of annotating and marking up is one of the key affordances which often makes paper the preferred medium for interacting with a document. System architectures and applications for different types of paper-based annotations have been described in the past including, for example, a framework for cross-media annotations [4].

In this section, we provide a brief overview of existing digital and paper-based approaches for annotating documents, focussing on projects using digital pens or Tablet PCs.

### Adobe Acrobat

The prevalence of Portable Document Format (PDF) documents as a document exchange format in combination with the possibility to easily read them on any device and any operating system, places Adobe Acrobat[1] at a privileged position for the digital proof-reading of documents. Adobe Acrobat Professional allows a reviewer to annotate a document either by means of free-form annotations as written on a Tablet PC or with regular textual annotations entered via the keyboard. The notes are stored as separate metadata and may be printed together with the document. A highlighting feature is provided in order to emphasise specific parts of a document. Other functionality includes text and graphical editing.

The latest version of the Acrobat tool also supports shared reviews by means of network folders, WebDAV servers, or Microsoft SharePoint Services and the corresponding synchronisation mechanism. Authors of different comments may be identified by their notes’ style attributes (e.g. colours). If a document was originally authored in Microsoft Word and exported as a *tagged* PDF or if it has been drawn in AutoCAD and then transformed into a PDF with the AutoCAD PDFMaker, Acrobat provides the possibility to export the annotations and to import notes, comments and even text elements added to the source document. However, while it is convenient to use the stylus on a Tablet PC for simple free-form annotations, more complex operations, such as deleting a word or changing some text, require the use of a mouse and keyboard.

## Microsoft Ink Annotations

With the Windows XP Tablet PC and Windows Vista operating systems, Microsoft introduced the concept of ink annotations. Ink annotations were first introduced in the Windows Journal tool which allows a Tablet PC user to draw on the screen with a stylus and insert free-form annotations. This feature is now also part of Microsoft Office tools such as OneNote and Word, where ink annotations can be inserted in three different ways: (i) an *ink drawing* rendered into a dedicated canvas, (ii) an *ink note* rendered on top of existing content as a free-form annotation, or (iii) an *ink comment* consisting of a side note with the free-form annotation.

Similar to Adobe Acrobat, the widespread use of Microsoft Office also places this tool at a privileged position for the proof-reading of digital documents. However, even though Microsoft Office provides a change tracking mechanism, the free-form notes supported by ink annotations are not integrated within this mechanism and changing the layout of the document can cause problems. For example, adding or removing content before or after the annotations may destroy the alignment between annotations and content. Thus, reviewers who want to collaboratively proof-read a document using the change tracking functionalities still have to use mouse and keyboard interactions.

## XLibris

XLibris [20] is a project that supports the *active reading* of documents. The system was built for an early Tablet PC prototype. It proposes an annotation system based on free-form annotations which tries to mimic interaction with regular paper documents. The user is presented with a scanned version of the paper document which may be annotated with a pen. The annotations do not have to conform to any form or structural constraints and are inserted into the documents based on the pen's (x,y) position without considering any components of the anchored elements. The algorithm on which XLibris is based preserves the position of the annotations based on a reflow mechanism, even if the document is viewed with different fonts, different aspect ratios or on different devices. The annotation reflow functionality anchors the notes at a particular position within the digital document. Therefore content may be moved without losing the connection with the annotated element. In XLibris, each annotation is attached to a pagination-independent location. However, the document's content is static and may not be changed.

## PenMarked

An annotation tool designed for teaching and correcting student assignments is PenMarked [19]. The system enables a teacher to annotate student assignments in computer programming with comments and scores using a Tablet PC. Assignments are regarded as fixed documents since there is no collaborative authoring. Therefore annotation reflowing is not supported. Marks are processed by means of Optical Character Recognition (OCR) technology and collected to summarise the re-

sults of the assignment. Even if the correction of programming assignments might be effective on a fixed paper-like representation, this task often requires interaction with the digital source which is not supported by the tool.

## ProofRite

ProofRite [3] is a word processor developed by Conroy et al. based on Anoto technology. It supports the merging of annotations made by different users on a printed copy of a document with the digital source document. After making corrections on the printed document version, users may continue editing the digital document. ProofRite also supports the reflowing of the markings according to changes made by the users. The main purpose of ProofRite is to automatically transfer any paper-based annotations to the corresponding digital document rather than to interpret editing markup such as the deletion of a word and execute the corresponding operations on the source document. Also, ProofRite does not provide any change tracking functionality within the digital document.

## 3. REQUIREMENTS ANALYSIS

The process of proof-reading a document is a complex procedure involving multiple tasks and may span the digital and paper worlds or be a purely digital operation. The affordances of paper documents with respect to their digital counterpart have already been introduced and paper has been identified as an important medium for the proof-reading of a document. However, we should not forget the tools for digital editing which can be extremely helpful, especially in collaborative authoring. For this reason, we introduce the concept of a mixed paper-digital proof-reading system and analyse its requirements.

To be effective, such a system must be fully integrated within the publishing process. In other words, users wishing to proof-read a document on paper should be able to use their standard tool for the authoring of the digital document and print out an interactive paper version of it without much effort. As soon as the process of proof-reading is finished, the *integration* of the corrections done on paper into the digital source document should be automatic. Therefore, our goal is to have a *proof-editing* system rather than simply a proof-reading system which only records changes to be made as annotations without actually performing them.

The interaction with the system should be as intuitive as possible and cater for different categories of users and tasks. There exist multiple forms of proof-reading or copy-editing a document. Standards such as the BS 5261 proof-reading mark revision [8] proposed by the British Publisher Association or the ISO standard 5776 [9] are used by professional publishers and should be supported, but also more open approaches used by non-professionals should be taken into consideration. By generalising the idea behind these standards as well as informal markup commonly used by authors, all of the symbols used may be grouped under the same concept of gestures. Therefore we believe

|                                  | Adobe Acrobat Professional | Microsoft Office + Ink Annotations | Xlibris | PenMarked | ProofRite |
|----------------------------------|----------------------------|------------------------------------|---------|-----------|-----------|
| <i>Paper-digital integration</i> |                            |                                    |         |           | ✓         |
| <i>Automatic publishing</i>      | ✓                          | ✓                                  |         | ✓         |           |
| <i>Free-form annotations</i>     | ✓                          | ✓                                  | ✓       | ✓         | ✓         |
| <i>Proof-reading integration</i> | ✓                          | ✓                                  | ✓       |           | ✓         |
| <i>Annotation reflow</i>         |                            | ✓                                  | ✓       |           | ✓         |
| <i>Handwriting recognition</i>   |                            | ✓                                  |         |           |           |
| <i>Multiple reviewers</i>        | ✓                          | ✓                                  |         |           | ✓         |
| <i>Gesture support</i>           |                            |                                    |         |           |           |
| <i>Change tracking</i>           |                            | (✓)                                |         |           |           |
| <i>Semantic mapping</i>          | (✓)                        |                                    |         |           |           |

Table 1: Comparative requirements analysis

that a gesture-based approach should be supported by every paper-based proof-editing system. Moreover, the system should support the definition of new gestures and the assignment of operations to the corresponding gestures to support different forms of markup. For example, an inverted caret (∨) might be interpreted as an insert operation to be executed within the digital authoring tool.

As soon as the number of operations increases and the corresponding corrections are executed automatically within the authoring tool, it becomes difficult for the users to retrieve and understand the changes that have taken place within the digital source document. For this situation, we can use the change tracking functionality that is available in most modern authoring tools. The tracking of changes within a digital document becomes more difficult if the document has been proof-read by multiple reviewers, both on paper or in digital form. Multiple reviewers should be supported by identifying their corrections and highlighting them in a distinctive manner within the digital document. Again, in most modern authoring tools, this functionality is already provided and a paper-digital proof-editing system should also support it when working with paper documents.

In Section 2, we recognised a second form of markup that is commonly used to simply provide comments to the authors rather than editing instructions. We refer to these as free-form annotations and it is important that a paper-digital system should also be able to integrate paper-based as well as digital free-form annotations into the digital source document. Further, in the case of paper documents, it should be possible for handwritten free-form annotations to be integrated automatically into the digital source document as digital text based on handwriting recognition technologies.

Once annotations have been inserted into the source document, users should be free to edit and update the document or to display it in a different format without losing the association of an annotation to the part of the document that it annotates. This is known as *annotation reflow* functionality and should be offered by the proof-editing system.

To fully support many of the requirements outlined above, it is necessary to have a *structural mapping* from paper back to document elements within the digital au-

thoring tool. By this, we mean that it should be possible to associate logical elements within a paper document such as paragraphs, words, table entries and images with their digital counterparts and to do so at different granularity levels. Hence, it should be possible to select an entire section, a paragraph within it, a word or even a single character and apply an operation to the corresponding element within the digital document. Furthermore, it is important that the representation of the element on paper should be fully independent from its digital representation so that the mapping can be maintained over different versions of both the digital and printed instances of a document. Thus the mapping from a sentence within a paper document back to the corresponding structural element within the digital source document should still apply even if that document has been edited since it was printed.

## Comparative Analysis

The solutions presented in Section 2 already support some of the requirements outlined above as summarised in Table 1. The only real paper-digital system is ProofRite while the other tools are based on a paper-like representation of a digital document on a display. The majority of these tools support the publishing process since it is reduced to just displaying the document. Xlibris requires a special transformation of the source document, while there is currently no support for the automatic publishing of ProofRite documents. Some of the requirements such as the insertion of free-form annotations or the reflow of these notes are supported by many systems. Note that PenMarked and the Adobe Acrobat tool do not support annotation reflow since they are based on static documents.

Even if annotations are supported by most of the mentioned systems, there is still very limited support for handwriting recognition which is currently fully supported only by Microsoft’s ink annotations. As soon as the interaction becomes complex and multiple authors or reviewers are involved, Adobe Acrobat, Microsoft Office and ProofRite are the only applications that can still handle the annotations. The concept of gestures is not supported by any of the presented systems and only Microsoft Word makes use of the integrated change tracking, even if there is no way to interact with the tool through the ink annotations.

Most approaches do not take into account any of the structural information present at authoring time. Once a digital document is printed on paper, it is no longer possible to directly access the digital counterpart of a structural element printed on paper, nor is it possible to automatically establish a link between the printed elements and the original objects of the source document. The only system which partially supports this interaction is the Adobe Acrobat tool which may export some of the corrections made on the PDF into the source authoring tool, but only if the PDF has been generated with special tools for tag insertion. ProofRite has similar problems since it tracks the position of annotations on paper and anchors them to the elements (i.e. words) rendered at the same position within the digital document. The created link is based only on the physical positions of the annotations and anchored elements and not on the link's semantic meaning. This implies that if the format of digital elements is changed after printing (e.g. different font size), the physical (x,y) positions of the elements may differ from the ones recorded at printing time and ProofRite is no longer able to link the correct elements. This lack of structural information is a major weakness of most existing systems. The solution for this problem is to retain structural information when generating a physical rendering of a document.

#### 4. THE DOCUMENT LIFE-CYCLE

The requirements analysis already highlighted the major steps in the production of interactive paper documents and stressed the need to support the bidirectional and continuous transition between paper and digital media. By analysing the life-cycle of a document in digital and paper media, we can identify four essential stages that form a logical and cyclic workflow starting and ending within the authoring tool as shown in Figure 1.

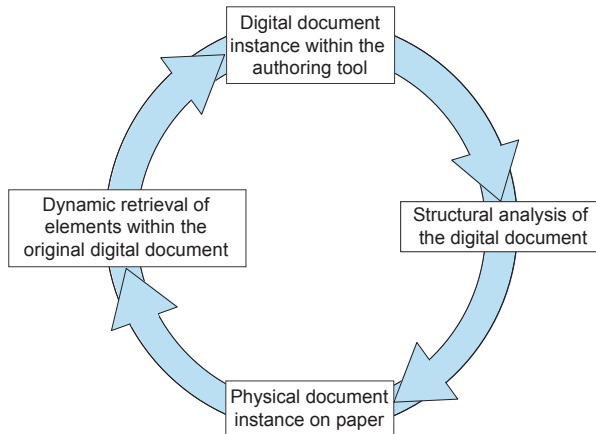


Figure 1: Document life-cycle

Nowadays, the first stage usually involves a digital authoring tool (e.g. Microsoft Word, OpenOffice Writer). An author normally uses such a tool to create an initial version of the document. Since we want to support paper-based proof-editing, the next step is to print out the document. In order to bridge the semantic gap

between paper and digital media, another important stage must be executed before the printing process—the structural analysis of the digital document. This step consists of extracting the document's structural information (e.g. sections, paragraphs and words) and making it persistent in order to be retrieved at a later stage while interacting with the physical document instance. For this reason, we enrich the publishing process in order to store additional information for single structural elements within the authoring tool. The third stage is the generation of the paper document which will be used for the proof-editing phase. At this point, the fourth stage is executed and, depending on the user's actions on paper, the digital elements are retrieved and the mapping between the physical and digital instances is performed.

As shown in Figure 1, the semantic analysis of the digital document and the dynamic retrieval of the digital elements are the core elements enabling a bidirectional mapping between printed and digital document elements. The definition of such a mapping mainly involves three tasks: (i) the implementation of a tool which intercepts the publishing flow within the authoring tool and analyses the structural information of the document prior to it being printed on paper, (ii) the design and implementation of an infrastructure that manages mixed physical and digital information coming from the two document instances and (iii) the design of a tool to effectively retrieve a digital element from a physical (x,y) position, resulting in a bidirectional mapping between printed and digital document versions.

Our vision of an integrated document life-cycle gives users the freedom to perform the same type of operations on either a printout or a digital version on screen. In the remainder of this paper, we will explain how this goal was achieved and provide more details about the mapping between digital and physical document instances. The approach described in the following is based on a specific Page Description Language (PDL), the Microsoft XML Paper Specification (XPS) [12], but is general enough to be used with other PDLs.

#### 5. FROM DIGITAL TO PAPER

The transition between digital and paper document instances must preserve the structural information available at authoring time. We enable the persistent storage of this information by applying two mechanisms that we call *structure highlighting* and *PDL annotation*.

##### Structure Highlighting

In order to extract the structural information from a digital document, we introduce the concept of a *highlighter* that uses graphical shapes as structural containers. During the publishing process, the document tree is analysed, information about the document structure is extracted and shapes are defined for every high-level object in the tree. These shapes are drawn around the different objects highlighting them and the document is transformed into a physical representation through an XPS virtual printer. Since the highlighting shapes are part of the document, they are not lost during the printing process and may be retrieved from the XPS file.

We have defined several types of highlighters depending on the structural elements that they are representing. The abstract class `Highlighter` is responsible for the activation of the connection with the authoring tool and for the printing of the document to an XPS file. For highlighting specific elements, several implementations are provided: section, table, graphics and text. They all provide a different implementation of the abstract class, adding specific methods for the execution of the highlighting process at the different structural levels. In order to better understand the concept, Figure 2 shows the result of highlighting a document at paragraph level: each paragraph of the first section has been enclosed by a polygon shape responsible for collecting all the words that are part of the paragraph.

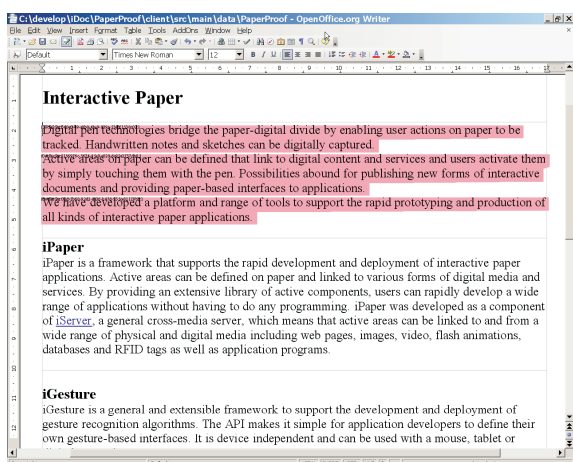


Figure 2: Highlighted paragraphs

## Dynamic Granularity

The document tree may contain structural information at different granularity levels. The solution of tracking single elements within a document, from section to character level, by highlighting them and extracting their positions is not a feasible solution, since it requires a large amount of space and computation time. A more appropriate approach is to make the operation dynamic by grouping smaller elements into higher level granularity sets (e.g. characters into words or words into paragraphs) and tracking the subelements only when they are really needed. Such an approach was first presented in [29].

The idea behind this approach is to have the possibility of referring to the logical object representation at different levels of granularity. For this reason, the concept of an element has been defined according to a composite design pattern [5] as shown in OM notation [14] in Figure 3.

An `Element` belongs to the `AtomicElements` collection if it does not contain any lower-level objects. Otherwise, it is inserted in the `CompositeElements` collection and a `ComposedBy` relation is created for each of its child elements.

Consequently, the element storage is first defined only at a high-level granularity, so that a small number of

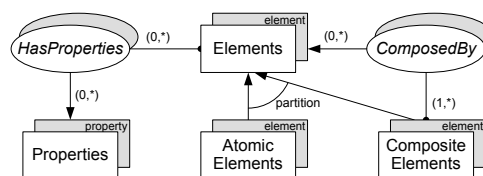


Figure 3: Element model

objects have to be analysed in a first step. For the analysis of elements at lower-level granularity, a dynamic approach is taken. Since the composite pattern creates a hierarchy of elements that also represents important structural information, we start from the highest granularity level and, if necessary, compute the subelements down to the desired granularity in a recursive manner. Therefore, the system offers a dynamic storage model for elements that refines the granularity of the stored structural information on demand.

## PDL Annotation

After the highlighting phase, elements that are structurally linked together are enclosed and presented as a single shape but the physical representation of the document does not yet provide a structural model. Moreover, the shapes interfere with the rest of the document content. In order to remove the shapes and insert structural information about the enclosed elements, we introduce the concept of *PDL annotations*. By using an *XPS snippet extractor*, we can track all highlighted shapes, extract physical information about their position and retrieve the elements that they contain.

To understand how the annotation process works, we first introduce the structure of an XPS document, focussing in particular on its `<Glyphs/>` and `<Path/>` elements. Note that a similar approach may be used for other PDLs.

An XPS file is basically a ZIP archive containing a set of XML files (one file for each physical page) describing the structure of the page at a graphical level. All fonts and images are also part of the ZIP archive. `Glyphs` elements are used within XPS to represent a sequence of uniformly formatted text. The most important attributes are the `OriginX` and `OriginY` defining a glyph's position on a page, the `FontUri` representing the font to be used, the `Indices` representing the distance between single characters composing a glyph and the `UnicodeString` containing the actual text content of the glyph. An example of an XPS file containing a `Glyphs` element is shown in Figure 4.

```
<Glyphs
  Fill="#ff000000" FontRenderingEmSize="14.7206"
  FontUri="/Documents/1/Resources/Fonts/
    083FE3E4-5F00-4F2E-85D9-B082CEBD4F5E.odttf"
  StyleSimulations="None" OriginX="96" OriginY="109.92"
  Indices="87;258;393;286;396,34;87;396;381;381;296,30;3"
  UnicodeString="PaperProof"/>
```

Figure 4: XML representation of a `Glyphs` element

`Path` elements are used to add vector graphics and raw images to an XPS page. They contain a `Data` attribute describing the geometry and position. Figure 5 shows an example of an XPS file containing two `Path` elements rendering a blue rectangle with a black border.

```
<Path
  Fill="#ff0000ff"
  Data="F1 M 95.04,131.84 L 224.96,131.84 224.96,
    203.84 95.04,203.84 z"/>
<Path
  Stroke="#ff000000" StrokeThickness="4"
  StrokeLineJoin="Miter" StrokeMiterLimit="8"
  StrokeStartLineCap="Round" StrokeEndLineCap="Round"
  Data="F1 M 95.04,131.84 L 95.04,203.84 224.96,
    203.84 224.96,131.84 z"/>
```

Figure 5: XML representation of `Path` elements

Since an XPS document is simply a ZIP archive, it is easy to add new content to it without interfering with the correct display of the paginated format. We store the information about the structure of the document retrieved from the highlighted shapes in a separate XML document, called `references.xml`, and add it to the XPS archive. The annotation process consists of the following steps:

1. Create an XML node (`Section`, `Paragraph`, `Table` or `Graphic`) for each highlighted shape and add a child `Shape` containing an XML description of the space covered by the highlighting shape.
2. Extract the physical information for each `Glyphs` and `Path` node of the XPS file and check if it resides within a highlighted shape.
3. If the `Glyphs` and `Path` nodes are part of the highlighted shape, create an `XInclude` node as a child of the element representing the highlighted shape within the `references.xml` file.

The `XInclude` element is a link to an external XML element referenced by an XPath expression in the `href` attribute [27]. By using such an approach, we reduce the amount of redundant information within the XPS document, since the original XML nodes are not replicated in the `references.xml` file. An example of an annotation for a section containing multiple paragraphs is shown in Figure 6. In this example, the `XInclude` link refers to the first XPS page `1.fpage`.

By means of these XPS annotations, XPS elements at any granularity level may be mapped back to the original digital document in a dynamic way as described in the next section.

## 6. FROM PAPER TO DIGITAL

In order to enable mapping back from any physical document instance to the corresponding digital elements, we define a *document mapper* exploiting the advantages of the dynamic granularity process during the publishing process and an *element finder* to retrieve the corresponding digital element within the source authoring tool.

```
<section
  id="iDoc:Sec:9603a909-66b7-419c-ae3c-4038762161d2"
  PageNr="1" secName="Introduction" secRelIndex="1">
  <paragraph
    id="iDoc:Par:ef39a647-bb4-45c0-bd81-7c42adcdd2fd"
    parent="iDoc:Sec:9603a909-66b7-419c-ae3c-4038762161d2"
    relativePosition="1" secRelIndex="1">
    <xi:include href="1.fpage" parse="xml"
      xpointer="element(/1/2)"/>
    <xi:include href="1.fpage" parse="xml"
      xpointer="element(/1/3)"/>
    <shape>
      <polygon>
        <point><x>74.4</x><y>160.0</y></point>
        <point><x>74.4</x><y>144.0</y></point>
        ...
      </polygon>
    </shape>
  </paragraph>
  <shape>
    <rectangle>
      <upperLeft>
        <point><x>72.96</x><y>140.0</y></point>
      </upperLeft>
      <size>
        <width>648.0</width><height>146.8</height>
      </size>
    </rectangle>
  </shape>
</section>
```

Figure 6: Annotation of paragraph elements

## Document Mapper

Once the PDL annotation has been successfully performed, it is possible to build a hierarchy of elements which recursively contains other elements as outlined earlier in Figure 3. We defined all the structural elements (`section`, `paragraph`, `word`, `character`) as objects that extend the base abstract class `Element`. This class contains the basic fields and methods common to all elements. These element objects build a hierarchical structure defined through the `getElements()` method computing an element's child elements as illustrated in Figure 7.

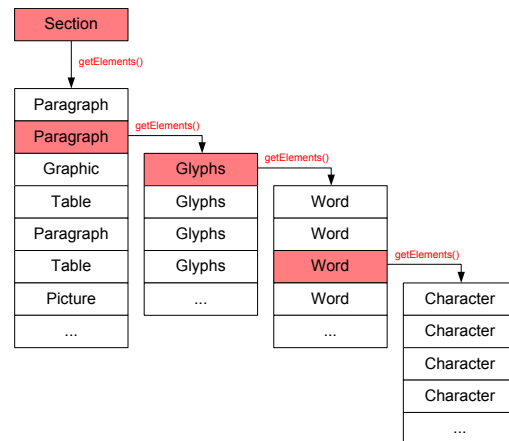


Figure 7: Element hierarchy



To retrieve a section's elements, `getElements()` first highlights the elements contained within the section in the authoring tool and produces the highlighted XPS file. The XPS file is parsed, the XML representations of the subelements are retrieved and the corresponding objects are created. Note that for graphical objects there is no implementation of the `getElements()` method since a raw image does not contain other elements.

For **Paragraph** elements, `getElements()` returns a list of **Word** objects. The method first creates **Glyphs** objects according to the glyphs node contained in the XML representation of the paragraph and calls the method `getElements()` for each of the **Glyphs** objects. The subelements of a **Paragraph** are therefore the words composing it.

To get the position and shapes of a paragraph's words, several operations have to be computed locally on the XPS file. As outlined before, the XPS **Glyphs** element stores the physical coordinates of the beginning of the glyph, the URI of the font used to render the text, the textual content of the glyph and a list of indices. By combining all this information, it is possible to access the dimensions of every single word composing a glyph and also of all the characters composing a word. This approach is used in the `getElements()` implementation for **Paragraph** and **Word** elements.

## Element Finder

When printing the document, the highest possible granularity level is retrieved from the digital document and an XPS file is created and annotated. In order to retrieve a digital element from paper, an *element finder* extracts the information about selected elements from the XPS file and forwards it to a *physical-digital adapter* that retrieves the corresponding digital elements in the source authoring tool.

Depending on the elements selected on paper, the `findElements()` method provided by the element finder returns one or more elements. The system relies on the gestures that a user performs on paper to understand their intentions. For instance, the system does not know whether a user wants to select an entire word or only some of its characters. Therefore, we implemented a simple algorithm for the automatic resolution of the granularity level.

Starting from the highest element granularity (i.e. section), the system first checks how many elements have been selected by the gesture. If more than one element is found, we are already at the correct granularity level and the elements are returned. If only one element is found, it might be because the desired granularity level is lower than the current one and the paragraphs, tables and graphics contained within the section have to be analysed. This process continues recursively until the lowest possible granularity level is found. If, during this process, all subelements of a parent node are selected by the gesture, it means that the gesture has to be applied at a higher granularity and the system goes back to the parent level. The resulting algorithm is outlined in Figure 8.

Since the retrieval of elements implemented within the `findElements()` method has to be general enough

```

Input: shape S on paper
Output: selected digital element(s)
Set Granularity level G at the highest possible value
while not at lowest granularity do
  search elements within shape S with granularity=G
  if all elements at granularity G found then
    | G = Parent.granularity
    | return Parent
  end
  else if only one element E found then
    | G = E.granularity
  end
  else if more elements found then
    | return all elements found
  end
  else
    | no elements found
    | return nothing
  end
end

```

Figure 8: Dynamic granularity algorithm

to be adapted to every authoring tool, we decided to use a very simple method where the relative position of elements within their parents is taken into consideration. When the elements are published within the XPS repository, a reference to their parent and their relative position is inserted along with all the information about their content and physical position. Figure 9 shows a snippet of the XML file which semantically annotates the XPS. It is possible to see the relationship between a section and its first paragraph.

```

<section
  id="iDoc:Sec:9603a909-66b7-419c-ae3c-4038762161"
  pageNr="1" secName="Introduction" secRelIndex="1">
  <paragraph
    id="iDoc:Par:ef39a647-bb4-45c0-bd81-7c42adddd2fd"
    parent="iDoc:Sec:9603a909-66b7-419c-ae3c-4038762161"
    relativePosition="1" secRelIndex="1">
    ...
  </paragraph>
  ...
</section>

```

Figure 9: Relative element position

This additional information is used in the retrieval phase, according to a two phase operation. To better explain the retrieval process, let us have a look at an example outlined by the sequence diagram in Figure 10. In this example, a digital pen is used to select a word in a printed document and we want to get access to the corresponding digital element within the authoring tool.

Starting from the coordinates received from the pen, the document mapper calls the `findElements()` method of the **ElementFinder** class. This method retrieves from the XPS annotations the physical elements bound to these coordinates. As explained above, it might happen that no elements are bound to this position causing the need to analyse the digital document at a finer granularity level. In this case, the highlighting process is started and the **Publisher** generates a new version of the annotated physical document containing more detailed structural information. The second phase runs in the opposite direction. The counting feature is implemented in



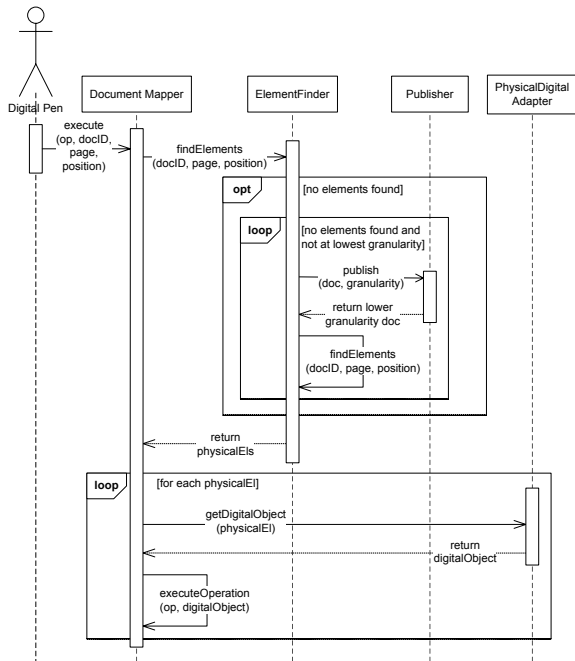


Figure 10: Paper-digital sequence diagram

the `PhysicalDigitalAdapter` class which, given one or more elements retrieved from the XPS file, gets the corresponding digital elements from the authoring tool by calling the `getDigitalObject()` method. By using the example above, the section found in the first phase is retrieved within the authoring tool and, based on the relative position of its children, the correct paragraph and word within the paragraph is found. Once this process is completed, the digital document is retrieved and the action defined through the digital pen on paper may be executed within the source authoring tool.

The supported structural mapping opens up a wide range of possibilities of interaction between digital and paper documents. In the next section we introduce the PaperProof proof-editing application that we have developed based on the presented mapping approach.

## 7. PAPERPROOF

Many authoring tools provide a way to track changes within a document. Whenever this functionality is used changes are made visible to the user rather than simply being executed on the text. They are therefore considered as tentative operations that can later be individually accepted or rejected, either by the same user or by other document editors in the case of collaborative authoring.

PaperProof makes use of the OpenOffice change tracking functionality in order to translate operations that a user performs on a printed OpenOffice document with a digital Anoto-based pen back into the digital document instance. Figure 11 shows the physical and digital representations of a text document processed by our system. On the left-hand side, the marks are done on paper

with a digital pen and, on the right-hand side, they are transferred into OpenOffice.

Once a document has been printed on paper, the proof-editing process can start. In order to execute commands in the PaperProof system, the five operations **insert**, **delete**, **replace**, **move** and **annotate** have been defined.

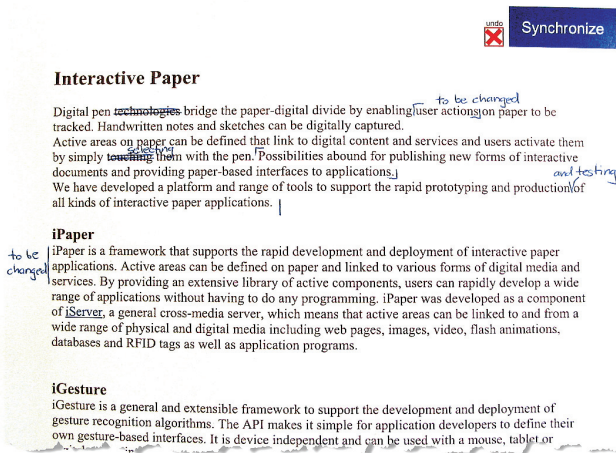
It is possible to apply operations at different granularity levels such as section, paragraph, table, graphics, word or character. One author might want to delete a whole paragraph, while another author just wants to correct a typo at the character level. An annotation might be made for a whole section or just for a particular word. The different granularity levels are automatically recognised by PaperProof using the approach outlined in Section 6.

In order to initiate commands for the desired operations, PaperProof makes use of gestures. Each operation is defined by simple or composed gestures recognised by the iGesture framework [23]. In the case of insert operations or annotations, an Intelligent Character Recognition software [26] further processes the pen input and translates the handwritten information into a string to be added to the digital document. Every operation is composed of one or more gestures, optionally followed by some textual input. The different gesture-based operations are illustrated in Figure 12. Please note that different gestures can be used for the same operation.

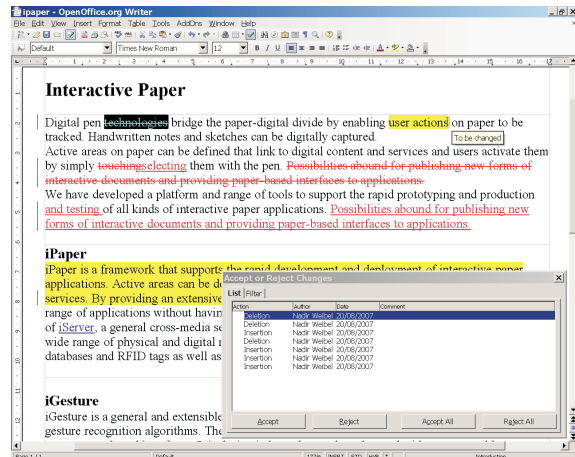
Depending on the gesture class, different granularity levels may be recognised and the system reacts in different ways.

- **Section:** **annotate** and **delete** are the two supported operations. The whole content of the section will be annotated or removed.
- **Graphics:** the **delete** operation which removes the graphics from a document is the only operation currently supported.
- **Table:** **annotate**, **delete** and **move** operations are supported. The annotation is inserted in the first cell of the table and, in the case of a deletion, only the content of the table is deleted, but not the table itself.
- **Text:** this category involves paragraphs, words and characters. All operations are supported.

Since users can make multiple corrections, it is important to ensure that the execution of an operation does not influence the position of succeeding operations within the digital document. Therefore, we first collect all operations with the digital pen and synchronise it with the OpenOffice source document. Collecting the operations might also be done in a mobile environment with the user on the move. In such a case, all information would be stored within the pen and the synchronisation would occur when users return to their computer. However, some sort of feedback would be useful to inform the user about the performed operations. Unfortunately, there is currently no way to provide feedback on the digital pens. This is a known issue and there have been solutions proposed to solve this problem [10]. In



(a) Corrections on paper



(b) Change tracking in OpenOffice

Figure 11: PaperProof

PaperProof, we currently only use the computer screen for feedback.

It may happen that a gesture is not recognised or that the written text is not detected correctly. For that reason, we provide an undo button which can be used to iteratively undo operations. As shown in Figure 11a, the synchronise and undo buttons are currently printed in the document header. As soon as the corrections are transferred to the system, PaperProof makes use of the approach outlined in Section 6 to retrieve the elements to be edited and execute the edit operations. Since we are using the change tracking functionality offered by OpenOffice, a user may still go through all edits and specify which to accept and reject.






















|                      |   |   |
|----------------------|---|---|
| Delete               |  or   |   |
| Replace              |  or   | + ICR   |
| Insert               |  or   | + ICR   |
| Move                 |  or  or  | +  or  or  +  |
| Free Annotation      |   | ICR   |
| Side Annotation      |  or   | + ICR   |
| Two Point Annotation |  or  or  | +  or  or  + ICR   |

Figure 12: PaperProof gestures

Note that based on the OpenOffice change tracking functionality that supports different authors, our application could easily be extended into a collaborative proof-editing system. In this case, a document could be printed several times and each copy distributed to a different author. Existing digital pens provide unique IDs enabling the system to handle corrections from different reviewers.

## 8. IMPLEMENTATION

The implementation of the PaperProof application is based on the four main components, iPublish, iDoc, iPaper and iServer, that have been developed for general paper-digital integration and interaction. We will first briefly introduce those four components and then provide specific implementation details about PaperProof.

The *iPublish* component consists of a series of plug-ins defined for different kinds of documents. Depending on the type of document we are interested in (PDF, Microsoft Word, OpenOffice, etc.), iPublish allows specific plug-ins to be defined that either track structured elements from within an authoring tool or analyse existing paginated documents. The basic purpose of the iPublish layer is to identify both the physical and the structural information of the document. Plug-ins have already been defined for the automatic authoring and publishing of documents coming from a Content Management System [15] and for the semantic analysis of PDF documents [17]. During the development of PaperProof, a new iPublish plug-in for OpenOffice was defined based on the OpenOffice SDK [18] that allows programmatic access to all of the features offered by the OpenOffice authoring suite. By using the SDK API, we could get access to the structural information of the document stored within the document in a tree-based model and define the OpenOffice highlighters outlined in Section 5. Using the same approach, other plug-ins for a given authoring tool, such as a PaperProof plug-in for Microsoft Word, could be defined. The implementation of this specific iPublish plug-in might be based completely on the existing infrastructure. However, since the creation of an XPS file from a Microsoft Office document through the plug-in for XPS already provides all the structural information that our infrastructure needs, the highlighting phase would be simplified and only a specific snippet extractor for Microsoft Office needs to be implemented.

The *iDoc* framework is based on a mixed digital and physical model, which is able to store metadata about

both the digital and paper instances of a document [29]. This mixed model is based on two distinct parts representing metadata coming from logical structures, also referred to as digital documents, and paginated formats sometimes called physical documents. The idea outlined in Section 5 to have the possibility to refer to the logical representation of the objects at every level of granularity was developed according to the composite design pattern shown earlier in Figure 3 and is one of the building blocks of the model implemented within iDoc.

The *iServer* and *iPaper* components are responsible for linking paper to digital services [22]. During the printing process, all information required to identify the interactive paper documents as well as link definitions are published to the iServer database. The iPaper plug-in is then responsible for enabling access to those links by selecting specific active areas on a paper document. Each time a user points to an (x,y) position within an active area, the iPaper plug-in resolves the selected shape and its associated links will be activated. In order to resolve the correct shape, the iPaper plug-in needs three inputs: a document identifier, a page number and the (x,y) position within the page.

The definition of the mapping from paper back to OpenOffice and the implementation of PaperProof involved two major tasks: the implementation of an iPublish plug-in for OpenOffice and the design and implementation of an iDoc extension that uses XPS documents to retrieve elements within the original digital document. In the rest of this section, we provide more details about the PaperProof implementation.

After a new digital document has been created with OpenOffice, the iPublish plug-in enables the publishing of an interactive paper version of the document based on Anoto technology. The structure of the file is analysed and information about the highest granularity level is extracted. An XPS file is then generated and annotated and the document is ready to be edited on paper. In order to support this paper-based interaction, the information about the Anoto pattern used and the location of the OpenOffice document has to be published to the iPaper plug-in.

As soon as an area on the page is touched with a digital pen, a special operation buffer is initialised and the operations, recognised by the combination of iGesture and ICR as explained in Section 7, will be stored. The entire application logic is implemented as an *active component* within the iPaper framework. An active component is basically a piece of program code that is loaded at link activation time and becomes a handler for the information from the input device [22].

When all annotations and corrections have been collected, the user touches the *synchronize* button (see Figure 11a) which activates another active component to retrieve the corrections from the buffer and send them to the document mapper. In order to do that, PaperProof provides the **Corrections** class with the signature shown in Figure 13. The constructor of this class requires four parameters: a list of edit operations to be performed, the path and the version of the OpenOffice document to be edited and the path of the XPS file that contains the mixed digital-physical information.

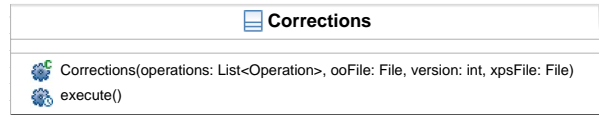


Figure 13: **Corrections** class

The **Corrections** class is the connection between the existing iServer/iPaper infrastructure, the iDoc framework and the PaperProof application. A schematic representation of the information flow is outlined in Figure 14. The `execute()` method iterates over the received edit operations and extracts the (x,y) position of the corrections from the digital pen. At this stage, the **XPSElementFinder** class extracts information about the selected printed elements from the XPS file which is then forwarded to the **AdapterXPSOO** class retrieving the corresponding digital OpenOffice elements.

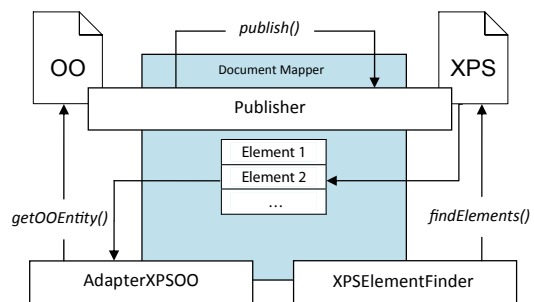


Figure 14: Information flow in PaperProof

After the corresponding digital elements have been retrieved, a **ChangeTracker** will transform the user operations into OpenOffice annotations and edits. As already discussed in Section 7, the changes to be applied in OpenOffice depend on the detected granularity level. For that reason, there exist four types of ChangeTrackers for sections, tables, graphics and text, respectively. Moreover, before executing any corrections, we first instantiate the ChangeTrackers needed for the execution of all the corrections. In the case of several authors, all their ChangeTrackers get instantiated at the same time and all the elements of the document that have to be corrected are properly referenced. Even if they were moved by a previous operation, the corresponding ChangeTrackers are still able to retrieve them. In this way, we ensure the safe execution of all corrections.

## 9. DISCUSSION

In Section 3, we presented the requirements for paper-digital proof-editing tools and highlighted some of the features that are missing in existing systems. After presenting and validating our solution, we now position it with respect to the original requirements.

An analysis of existing approaches shows that they differ significantly from the solution presented in this paper. In terms of paper-digital interaction, our proof-editing application seamlessly integrates paper and digital document versions in a similar way to ProofRite and

paper-based annotations are automatically transferred to the original digital document. However, instead of simply integrating handwritten annotations into the digital document, PaperProof interprets them and applies the edits to the digital document instance.

In digital-only solutions as well as existing paper-digital approaches, captured annotations will reflow if the digital copy is edited. However, the notes are still bound to relative positions within the digital document instead of referring to structural elements within the document. ProofRite refers to the position captured from paper, while digital-only systems retrieve the position returned by the stylus used on a display. Due to the lack of a binding between the structural elements of a document on paper and those in digital form, existing systems do not support the automatic integration of edits and only provide visual clues. Instead they simply copy the annotations into the digital document and still require editing operations to be performed manually.

In contrast, annotations and editing operations are interpreted and executed by PaperProof. Our system is unique in analysing paper-based edits and transforming them into operations and textual annotations based on gesture recognition and ICR technologies. To the best of our knowledge, our system is the only paper-based editing solution that also exploits change tracking functionality offered by digital authoring tools.

The automatic publishing of interactive paper documents is also supported by PaperProof which makes use of the iDoc Virtual Printer [16] to support the production of an augmented document based on Anoto technology. The virtual printer can be selected from the list of available devices and the digital document is automatically analysed and augmented with Anoto pattern. The user does not have to perform any additional steps and can start the proof-editing session as soon as the document is retrieved from the printer.

PaperProof also reveals some issues to be solved in terms of usability. It is often difficult to understand a user's intentions by using only pen-based gestures. The PaperProof gestures have been selected in a way that should feel natural to the end user. However, we plan to do some user studies in order to evaluate the usability of the paper-digital user interface.

The automatic granularity check presented in Section 6 enables a much smoother interaction with the system, without the user having to explicitly select the granularity level. However, dealing with precise gestures may lead to a lack of accuracy on the user side, which may easily cause unexpected results. For example, if the pen accidentally touches the next word while deleting a word, that would also be deleted. Since the user intention is not known a priori, the system should automatically select the best solution. What does it mean, for example, if a user crosses out all the characters in a word except the last one? Is there only a lack in precision or should the last character really not be removed? Our algorithm for the automatic granularity detection works fine but the analysis of user intention is still a source of potential misinterpretations. User studies will also be required to determine the best way to handle these issues.

While the retrieval of digital elements based on the selection of the corresponding element on paper works well, there are still some problems in terms of performance. This is probably due to the current implementation which uses the XPS file as the repository for elements. This is costly in terms of performance since parsing an XML file and searching for an element is a rather slow operation. Moving to the document database provided by the iDoc framework should lead to faster element retrieval.

The lookup of digital PaperProof elements is based on the relative position of an element with respect to its parent element. This is a generic approach which could also be used in the development of different plug-ins, for example a PaperProof authoring tool for Microsoft Word. A future Microsoft Word plug-in might be completely based on the existing infrastructure. The implementation of the iPublish plug-in would be straightforward since the XPS documents created from a Microsoft Word document already contain the semantic information required by our infrastructure. Therefore, the only parts that would have to be changed are the PaperProof application itself and the iDoc extension with the implementation of a simple snippet extractor adapted to the semantic information stored within the XPS files generated by Microsoft Word.

From this brief analysis, we see how our PaperProof application and the related technologies go a step further in the definition of an effective paper-digital proof-editing system and provide a solution for some of the open issues presented in Section 3. The PaperProof application further demonstrates the feasibility of a bidirectional paper-digital mapping.

## 10. CONCLUSIONS

We have presented a novel approach for capturing the structural model of a digital document during the printing process in order to maintain a logical, bidirectional mapping between digital and paper instances of a document. Our approach has been validated by PaperProof, a paper-digital application for document proof-editing that, not only enables comments to be captured as annotations, but also provides digital editing functionality based on the interpretation of handwritten gesture-based commands.

We plan to extend PaperProof in order to fully support multiple simultaneous reviews of the same document. This requires the definition of a version model which has to be shared between the paper and digital document instances. We also plan to carry out user studies and investigate various issues that may affect system usability, including feedback mechanisms and the gesture-based interaction design. Last but not least, we aim to develop an alternative PaperProof implementation based on Microsoft Word.

## 11. REFERENCES

- [1] Adobe Acrobat Professional, <http://www.adobe.com/products/acrobatpro/>.
- [2] Anoto AB, <http://www.anoto.com>.

- [3] K. Conroy, D. Levin, and F. Guimbretière. ProofRite: A Paper-Augmented Word Processor. Technical Report HCIL-2004-22, CS-TR-4652, Human-Computer Interaction Lab, University of Maryland, USA, May 2004.
- [4] C. Decurtins, M. C. Norrie, and B. Signer. Putting the Gloss on Paper: A Framework for Cross-Media Annotation. *New Review of Hypermedia and Multimedia*, 9:35–57, 2003.
- [5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [6] F. Guimbretière. Paper Augmented Digital Documents. In *Proceedings of UIST 2003, 16th Annual ACM Symposium on User Interface Software and Technology*, pages 51–60, Vancouver, Canada, November 2003.
- [7] H. Ikeda, N. Furakawa, and K. Konoishi. iJITinLab: Information Handling Environment Enabling Integration of Paper and Electronic Documents. In *Proceedings of CoPADD 2006, 1st International Workshop on Collaborating over Paper and Digital Documents*, pages 25–28, Banff, Canada, November 2006.
- [8] Proofreading Mark Revision Suggestions. Technical report, The Publisher Association, 2003.
- [9] ISO 5776:1983, Graphic Technology – Symbols for Text Correction. Technical report, International Organization for Standardization, 2005.
- [10] C. Liao, F. Guimbretière, and C. E. Loeckenhoff. Pen-top Feedback for Paper-Based Interfaces. In *Proceedings of UIST 2006, 19th Annual ACM Symposium on User Interface Software and Technology*, pages 201–210, Montreux, Switzerland, October 2006.
- [11] C. C. Marshall. Toward an Ecology of Hypertext Annotation. In *Proceedings of Hypertext '98, 9th ACM Conference on Hypertext and Hypermedia*, pages 40–49, Pittsburgh, USA, June 1998.
- [12] Microsoft Corporation. *XML Paper Specification*, 1st edition, October 2006.
- [13] Sanford Brands - Mimio, <http://www.mimio.com>.
- [14] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *Proceedings of ER '93, 12th International Conference on the Entity-Relationship Approach*, pages 390–401, Arlington, USA, December 1993.
- [15] M. C. Norrie, B. Signer, M. Grossniklaus, R. Belotti, C. Decurtins, and N. Weibel. Context-Aware Platform for Mobile Data Management. *ACM/Baltzer Journal on Wireless Networks (WINET)*, pages 855–870, 2007.
- [16] M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proceedings of CoPADD 2006, 1st International Workshop on Collaborating over Paper and Digital Documents*, pages 9–12, Banff, Canada, November 2006.
- [17] M. C. Norrie, B. Signer, and N. Weibel. Print-n-Link: Weaving the Paper Web. In *Proceedings of DocEng 2006, 6th ACM Symposium on Document Engineering*, pages 34–43, Amsterdam, The Netherlands, October 2006.
- [18] Openoffice.org, <http://www.openoffice.org>.
- [19] B. Plimmer and P. Mason. A Pen-Based Paperless Environment for Annotating and Marking Student Assignments. In *Proceedings of AUIIC 2006, 7th Australasian User Interface Conference*, pages 37–44, Hobart, Australia, January 2006.
- [20] B. N. Schilit, G. Golovchinsky, and M. N. Price. Beyond Paper: Supporting Active Reading with Free Form Digital Ink Annotations. In *Proceedings of CHI '98, ACM Conference on Human Factors in Computing Systems*, pages 249–256, Los Angeles, USA, April 1998.
- [21] A. J. Sellen and R. Harper. *The Myth of the Paperless Office*. MIT Press, November 2001.
- [22] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. PhD thesis, ETH Zurich, 2006.
- [23] B. Signer, U. Kurmann, and M. C. Norrie. iGesture: A General Gesture Recognition Framework. In *Proceedings of ICDAR 2007, 9th International Conference on Document Analysis and Recognition*, pages 954–958, Curitiba, Brazil, September 2007.
- [24] B. Signer and M. C. Norrie. PaperPoint: A Paper-Based Presentation and Interactive Paper Prototyping Tool. In *Proceedings of TEI 2007, 1st International Conference on Tangible and Embedded Interaction*, pages 57–64, Baton Rouge, USA, February 2007.
- [25] B. Ullmer and H. Ishii. mediaBlocks: Tangible Interfaces for Online Media. In *Extended Abstracts of CHI '99, ACM Conference on Human Factors in Computing Systems*, pages 31–32, Pittsburgh, USA, May 1999.
- [26] VisionObjects, MyScript Builder, <http://www.visionobjects.com>.
- [27] W3C, World Wide Web Consortium. *XML Inclusions (XInclude) Version 1.0*, November 2006. W3C Recommendation.
- [28] Wacom Graphic Tablet, <http://www.wacom.com>.
- [29] N. Weibel, M. C. Norrie, and B. Signer. A Model for Mapping between Printed and Digital Document Instances. In *Proceedings of DocEng 2007, 7th ACM Symposium on Document Engineering*, pages 19–28, Winnipeg, Canada, August 2007.
- [30] P. Wellner. Interacting with Paper on the DigitalDesk. *Communications of the ACM*, 36(7), July 1993.
- [31] R. B. Yeh, C. Liao, S. R. Klemmer, F. Guimbretière, B. Lee, B. Kakaradov, J. Stamberger, and A. Paepcke. ButterflyNet: A Mobile Capture and Access System for Field Biology Research. In *Proceedings of CHI 2006, ACM Conference on Human Factors in Computing Systems*, pages 571–580, Montréal, Canada, April 2006.