A Model for Mapping between Printed and Digital Document Instances

Nadir Weibel
Dept. of Computer Science
ETH Zurich
8092 Zurich, Switzerland
weibel@inf.ethz.ch

Moira C. Norrie
Dept. of Computer Science
ETH Zurich
8092 Zurich, Switzerland
norrie@inf.ethz.ch

Beat Signer
Dept. of Computer Science
ETH Zurich
8092 Zurich, Switzerland
signer@inf.ethz.ch

ABSTRACT

The first steps towards bridging the paper-digital divide have been achieved with the development of a range of technologies that allow printed documents to be linked to digital content and services. However, the static nature of paper and limited structural information encoded in classical paginated formats make it difficult to map between parts of a printed instance of a document and logical elements of a digital instance of the same document, especially taking document revisions into account. We present a solution to this problem based on a model that combines metadata of the digital and printed instances to enable a seamless mapping between digital documents and their physical counterparts on paper. We also describe how the model was used to develop iDoc, a framework that supports the authoring and publishing of interactive paper documents.

Categories and Subject Descriptors

H.1.2 [**Information Systems**]: Models and Principles - User/Machine Systems; D.2.11 [**Software Engineering**]: Software Architectures; H.4.1 [**Information Systems Applications**]: Office Automation

General Terms

Design, Algorithms, Experimentation, Human Factors

Keywords

Interactive paper, document integration, document model, structured documents, page description languages

1. INTRODUCTION

Paper still plays an important role in the document life cycle. While the digital revolution has had a tremendous impact on how we author, publish, distribute and manage documents, the actual reading and annotation of documents still tends to happen on printed versions of these documents.

Therefore, despite predictions of the paperless office, instead of removing paper and printing from the document life cycle, they are now mainly associated with stages that involve reading and annotation. Several studies within the human computer interaction (HCI) field have highlighted the affordances of paper that may explain why it persists and remains a preferred medium for note-taking as well as reading [28].

The first steps towards bridging the paper-digital divide have already been taken and a range of technologies developed that allow printed documents to be linked to digital content and services. The most advanced of these is *Anoto technology* [3] that uses a digital pen with an integrated camera to track the positions of the pen on paper through a special tiny dot pattern printed on the paper to encode position data. The technology was developed for the digital capture of handwriting, but nowadays digital pens are available that can deliver the position information in streaming mode, enabling paper to be turned into an interactive medium. Links on paper can be defined in terms of active areas and when the pen touches the paper within an active area, the link is activated and a request sent to a server to activate the target resource. In the case of a media file such as an image, video or web page, the resource will be retrieved and displayed. However, if the resource is some form of digital service, a request will be forwarded to that service. Handling interactions between paper and digital documents in a dynamic and flexible way is complex and a general infrastructure for this is desirable. The *iServer* infrastructure [20] together with its interactive paper plug-in (*iPaper*) [22] introduce such an approach and have been shown to support the rapid development of interactive paper documents and paper-based interfaces for a wide-range of applications.

As outlined in [23], the publishing of interactive paper documents can be a complex task including the authoring of the source documents, followed by the definition of the digital resources and services to be bound to the specific active areas. The last step involves the mapping of the digital document to its physical counterpart together with the generation of the link definitions that specify physical regions on paper as active areas. But so far this tends to be a one-way process used to generate printed instances of documents that are interactive rather than a process that supports the seamless transition between digital and printed instances of the same document within the document life cycle.

The static nature of paper and limited structural information encoded in classical paginated formats make it difficult to map between parts of a printed document instance and logical elements of a digital instance of the same document, especially taking document revisions into account. In this paper, we present a solution to this problem based on a model that combines metadata of the digital and printed instances to enable a seamless mapping between digital documents and their physical counterparts on paper. We also describe how the model was used to develop *iDoc*, a framework that supports the authoring and publishing of interactive paper documents.

We begin in Section 2 with a more detailed discussion of the issues and also related work. Section 3 examines approaches to structured documents, first looking at logical models of structure and then physical representation based on different Page Description Languages (PDL). Section 4 then presents our mixed physical and digital model which stands as a bridge between logical and physical document representations. A particular extension of the model investigating the structural interrelation of the objects composing it is presented in Section 5. Section 6 presents iDoc, the framework enabling the publishing of interactive paper documents and outlines how our model supports the publishing process of an interactive document in terms of a concrete example. We discuss the advantages that our model brings in publishing interactive paper documents in Section 7. Concluding remarks are given in Section 8.

2. INTERACTIVE DOCUMENTS

There are many situations where it is necessary to map from a printed instance of a document back to a digital instance. For example, handwritten gestures such as circling words or phrases can be used to select objects from the digital document. This feature could be used in applications ranging from dictionary lookup or translation services to actual editing operations specified on paper. Another example is the capture and possible integration of annotations. The mapping back from physical positions on printed documents to elements of the source digital documents is even more problematic if the digital source is under revision.

There are three main categories of digital documents to be made interactive: (a) documents generated with classical authoring tools (e.g. Microsoft Office, OpenOffice, LaTeX), (b) documents automatically generated using data from a database or content management system (CMS) and (c) existing documents in a paginated form, with no access to the source document. These different kinds of digital documents are stored in a wide range of formats and representations which depend on how they have been produced. In any case, the final step towards the physical representation of the document is usually bound to a paginated format such as PDF or Postscript.

The use of paginated formats at publishing time allows access to information about the layout of the printed version of the document. This enables a correct mapping from the paper document and specifically from the (x,y) positions of the individual active areas to the specified digital service. However, this step breaks the correlation between the digital source document and the printed version. Since the authoring of the active areas is bound to the physical representation of the document, the process of publishing an interactive document is bound to the static concept of (x,y) positions. Products based on Anoto technologies such as the Forms Automation System [11] take the input of the user on paper and map it to a digital service based on the (x,y) position extracted from the paginated version of the digital document. More sophisticated projects like ProofRite [7] track the exact position of annotations on paper and anchor them to the words rendered at the same position within the source authoring tool. Similarly, as done in [15] with digital documents, paper annotations may be repositioned if the layout changes by computing the (x,y) displacement of the anchored word. These approaches do not take into account the semantic information provided by the user at authoring time. In other words, once a digital document is printed on paper, it is no longer possible to directly access the "meaning" of the augmented physical element in its digital counterpart, nor is it possible to programmatically establish a link between the printed elements and the original objects within the source document. Moreover, if the digital document changes, its mapping with the printed information is not preserved, and the entire process must be repeated.

Our vision is to enhance the integration between the physical world of paper documents and their digital counterparts by providing extended semantics at the physical level, transforming a static paper sheet into an active and reactive object tightly bound to its digital source. The provision of a clear model for mapping between printed and digital document instances is the key to support a seamless transition between the paper and digital worlds within the document life cycle. The interplay between paper and digital document instances is achieved by defining a common component where physical and digital information is exchanged freely and where every physical object may always be paired with its digital source. The core model stores a combination of physical and digital information needed to map paper documents to the different kinds of possible digital sources, thereby dealing with all three kinds of documents and existing authoring tools, but being general enough to support other kinds of documents or authoring tools in the future.

3. STRUCTURED DOCUMENTS

The increasing popularity of Personal Computers (PCs) for typesetting documents raised issues of document exchange between users at different sites. This created the need to define common document interchange formats. As defined by Kimura and Shaw [16], three representations of a document may be identified: (a) the document expressed in terms of its logical but abstract structure, (b) the document defined by its concrete appearance on a page-based representation and (c) the document as it may be viewed by the user by means of a display (monitor or printer). The first two representations are often referred to as the *logical* and the *physical structure* of a document respectively.

In this section we describe some models for structured documents in terms of both the logical and physical structure. We particularly focus on innovative tree-based approaches for modelling the logical structure of a document and paginated representation of their physical structure. The described approaches drove the development of our new mixed physical and digital model, presented in Section 4. A survey of other models and approaches as well as a review of the literature can be found in [12].

3.1 Logical Structure

The logical structure of a document may be seen as a collection of atomic and higher-level objects combined together to define the highest level of abstraction: the document. These atomic and composite elements may depend on the document class currently instantiated. Examples of such classes include books, technical papers, letters, newspapers or conference proceedings.

Reid defined a document model with hierarchical nesting that was used in the *Scribe* word processor [25]. Scribe introduced named environments which had the role of containers (e.g. ordered lists, tables). Environments could be nested and any kind of hierarchical structure can be defined through relationships between environments. Exceptional types of environments with constrained nesting behaviour could also be defined. The appearance of the document's objects (e.g. font, size, colour) was defined at the level of environments so that the style was completely independent from the content, clearly separating structure from presentation.

One of the most interesting models based on Reid's approach is *tnt* [13] which uses a forest of ordered trees to represent the different document parts. Atomic values are defined in a heterogeneous way at a higher-level of granularity. Instead of storing single characters as leaves of a tree, the atomic values might be represented as a whole text string. By encapsulating non-tree structures into the leaves, the primary logical structure remains relatively simple and easy to manipulate, while the storage of other structures might be addressed at another level in an easier way.

The approach of Dori et al. [9] is also based on a tree concept, combining low-level elements into higher level ones. Its innovation lies in the definition of a generic logical structure that can be applied to different document classes. In this model, every object is defined as *texton* or *graphon* at the highest level of granularity. Depending on the document class instantiated and the current granularity level, textons may be classified as paragraphs, sentences, words, characters, etc. while graphons may be instantiated as line drawings, images, charts, tables, etc.

Tree-like document models have been further developed in a range of different formats. Most of them were inspired by both Reid's work and the Standard Generalised Markup Language (SGML) [14] introduced by Goldfarb. Many languages for describing heterogeneous types of documents, for example the Text Encoding Initiative (TEI) and DocBook, have been defined. The strengths of such a document representation were recognised and they eventually developed into commonly used standards such as XHTML and XML.

The document models presented above were developed mainly for markup-based document editors. Moving on towards WYSIWYG editors and formatters such as Microsoft Word or OpenOffice Writer, we witness great improvements in terms of the user interface, but also limitations in the document structure representations. WYSIWYG editors are natural to use since they represent documents in a form close to the physical representation, but their need of an exact visual representation limits the possibility of describing complex structures. As a result, the document models behind modern document processors tend to be much simpler. Nevertheless, in the last few years, the development of document interchange formats based on XML demonstrated how complex structural information may be defined also within modern document processors. Recently, standards such as the Open Document Format for Office Applications (ODF) [24] and the Microsoft Open Office XML (OOXML) [10] opened the way for the XML-based exchange of documents between different office applications.

3.2 Physical Structure

The main objective of the physical representation of a document is independence from the printing device: a document should not differ when printed on different printers. Reid defined two representations of physical documents [26]. The first was based on a command-stream image description language, where virtual printer commands defined within the document are translated at printing time into actual printer command streams. The second one is a procedural page description language (PDL), where data structures are defined to capture the properties of the different document elements (e.g. font, colour, size). While independence from the printing device in terms of layout and content presentation is important for document exchange, the risk is to throw away the entire logical structure defined at authoring time.

The foundation for most of the paginated formats still in use today lies in Warnock and Wyatt's stencils and sources model [32]. This model was first used by the PostScript (PS) [1] language and describes how "ink" should be placed on paper. Sources define the properties of the ink currently in use and a set of stencils define the executable primitive procedures to create the objects. Every object is mathematically described using lines and curves, grouped into trajectories defined as a path. Alphabetic characters (glyphs) are also defined as trajectories and therefore fonts are represented by a collection of path elements. Dictionaries are available for storing key/value pairs of properties. If path elements have to be processed multiple times, the instructions may be grouped into procedures stored in the document's preamble.

Many properties defined by this model were innovative in the document engineering community. Its modularity and adaptability, together with its compactness, transformed PS into the de facto PDL standard during the 80's and early 90's, until it was supplanted by one of its descendants, the Portable Document Format (PDF) [2]. The PDF document model is based on a subset of PS, mainly required for the generation of graphics and definition of the layout. The main difference compared to PS is the removal of all flow control commands. As a matter of fact, PDF was born as a much simpler document format: in practice the graphic commands generated by the PS program page after page are collected, tokenised and transformed into a collection of pages forming the PDF file. A PDF page is a sequence of drawing instructions exactly defining a document's content and layout. Figure 1 shows an example of a `Hello World` text defined with an 11pt Times new Roman font.

```
14 0 obj
stream /GS1 gs
   BT
    /TT2 1 Tf
    10.98 0 0 10.98 72 759.8604 Tm
    0.0003 Tc 0.0011 Tw
    (Hello World )Tj
   ET
  endstream
 endobj
```

Figure 1: Text encoded in PDF format

The PDF snippet shows how the (x,y) position of the text relative to the page is encoded using the combination of the transformation matrix (`Tf`), the word spacing (`Tw`) and the character spacing (`Tc`) operators.

For a long time, no paginated format could compete with the predominance of PDF but new formats have been proposed in recent years. Scalable Vector Graphics (SVG) is an XML-based markup language proposed by the WWW community to improve the rendering of graphics objects within web browsers. The SVG model is similar to the one defined by PDF and PS and combines complex graphical representations and non-paginated text visualisation, defining a strict layout for them. The W3C consortium is working on an innovative SVG draft (SVG version 1.2) [31] which is trying to migrate SVG towards a model based on multiple pages. The SVG format allows different types of objects to be rendered including text, vector graphics shapes and raster graphics images which are normally defined as external resources referenced by the XML. Figure 2 shows the same "hello world" example encoded in SVG. The `viewBox` attribute of the `svg` tag represents the size of the page, while the `x` and `y` attributes of the `tspan` element specify the position of the string within the page.

```
<svg viewBox="0 0 2448 3168"
  preserveAspectRatio="xMinYMin meet">
 <style type="text/css"><![CDATA[
   text.t1 { font-family:'Times New Roman',serif;
   font-size:42px;fill:#000000}]]>
 </style>
 <text class="t1">
   <tspan  x="374,403,422,433,446,465,477,515,535,549,561"
     y="308">Hello World</tspan>
 </text>
</svg>
```

Figure 2: Text encoded in SVG format

The Microsoft XML Paper Specification (XPS) [18] is an XML-based paginated representation of a document based on the Microsoft Extensible Application Markup Language (XAML), a declarative markup language that essentially defines objects, their properties and relationships. XPS was released with Microsoft Vista and Office 2007 and is tied to Microsoft's *.NET Framework*. The single XPS pages are encoded as separate XML files and bundled together with all the necessary resources in a ZIP archive. Similar to PDF, XPS pages contain shared resources such as fonts and images referenced from the individual pages. The XPS model does not differ from the ones of PDF or PS. Content is defined for every page in different XML files, which are referenced from the main XML document. An extract from one `<FixedPage/>` element of an XPS document is shown in Figure 3. To obtain the positional information, the `OriginX` and `OriginY` attributes should be taken into consideration.

```
<FixedPage Width="816"Height="1056">
  <Glyphs Fill="#ff000000"
   FontUri="/Documents/1/Resources/Fonts/EDA847AC5DD8.odttf"
   FontRenderingEmSize="13.8106" StyleSimulations="None"
   OriginX="124.8" OriginY="102.72"
   Indices="43;72;79;79,29;82;3,24;58,93;82,51;85,34;79;71;3"
   UnicodeString="Hello World"/>
</FixedPage>
```

Figure 3: Text encoded in XPS format

On every page, two main types of objects may be defined to represent graphics or text: `<Glyphs/>` for text and `<Path/>` for graphics. Every element is self-describing in the sense that its properties (e.g. position, colour, size) are defined as attributes of the XML element. Similarly to SVG, the main advantages of XPS over PDF may be seen in its much more structured definition.

3.3 Annotating PDLs

As a result of the wide acceptance of PDF, nowadays documents are frequently exchanged using their physical, paginated representation. However, none of them actually includes information about the logical structure of the source documents and often metadata defined in the logical model is almost completely lost. In many of the most frequently used PDLs and also in evolving trends, we see a lot of potential for integrating semantically rich information about the logical structure of the source documents.

The extraction of logical structure from the physical representation of documents is an important research topic [9]. Given the widespread use of PDF documents, many approaches have investigated how structural information may be encoded within them. Most of the approaches assume that no source document is available and offer a solution to reconstruct the structure by analysing the content [8] or the graphical layout of the different elements within the document [6]. Other approaches define XML representations of PDF files suitable for annotation (e.g. XCDF [5]) or use XML-based standoff markup techniques to enhance digital documents [30]. However, in many cases, the source document is available and the paginated format is used only as an intermediate format. Technologies such as the Adobe Standard Structured Tagset (SST) [2] or approaches like the Component Object Graphic (COG) are interesting examples of how it is possible to add structural information to paginated formats starting from a source document. For instance, making use of COG-enabled authoring tools, both PDF and SVG documents have been successfully annotated with structural information [4, 17].

In a similar way, it would be possible to annotate XPS documents. Currently an XPS document can be created through a virtual printer or using a special Microsoft plug-in. Depending on the XPS creator used, it is possible to semantically group objects. All lines belonging to the same paragraph could be grouped together within a single `<Canvas/>` element, inserting semantics directly into the paginated document. Since XPS is an extensible XML file, it would even be possible to define structural information by grouping glyphs or path elements by means of specific tags such as `<Paragraph/>` or `<Section/>`.

From the presented PDLs, we chose three of them, namely PDF, SVG and XPS in order to show how these could be extended with logical information and use them within our platform. However, as we will see in Section 4, our model is extensible and could support any other PDL or extensions of existing ones.

4. MIXED PHYSICAL-DIGITAL MODEL

Both the logical and physical document representations outlined in Section 3 have been successful in defining models for presenting and exchanging documents among different users. With the introduction and extensive use of XML technologies, document engineers began to realise how logical

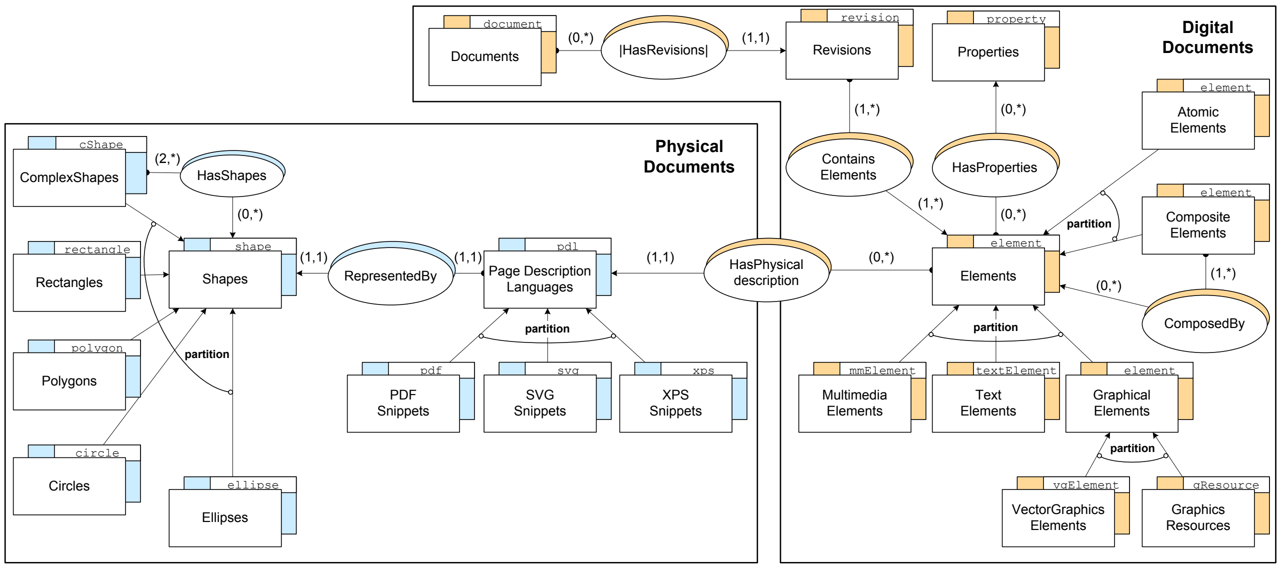

Figure 4: Mixed physical and digital model

structure could also be an important factor in a paginated version of a digital document. However, none of the existing technologies supports a full mapping between digital and physical document representations taking into account their structural objects. Our mixed physical and digital model stands in between these two document representations in order to enable this mapping. Our approach is to intercept the flow of data during the publishing process of a paginated document which may then be printed. At this point, we store all metadata needed in order to support the mapping back from the physical instance of the document to its structure defined at the logical level.

As already discussed, the logical structure may be defined at different levels of granularity (i.e. sections, paragraphs, words). Since we want to be able to refer to the logical representation of the objects at every level of granularity (e.g. at character level), this raises issues in terms of data storage. Should we store information about the position of every single character of a text document? Obviously this would be too much overhead even for a document with just a couple of pages. Another problem to consider is the static nature of a physical document. What happens if the source document is updated after its publication? How can we refer from paper to elements which have been moved to another position in the digital representation or to objects that have completely disappeared? In this section we show how our model addresses these and other issues, supporting the use of multiple physical and logical documents.

To describe our model, an extended entity-relationship model called OM [19] is used. OM is an object-oriented data model featuring powerful concepts such as object hierarchies, collection hierarchies, binary associations, multiple inheritance and multiple instantiation. It differs from commonly used object models such as UML in that it is designed as an operational model for data management as well as for conceptual modelling. OM therefore defines a full operational model over objects, collections and associations as well as constructs for their definition. Collections are used in OM for the classification of objects and multiple classification may be exploited by assigning the same object to different collections. Associations represent bidirectional links as a stand-alone concept and can be used to semantically connect and navigate large collections of objects. While collections are used to classify objects, types deal with their implementation. Graphically, the shaded rectangles shown in Figure 4 denote collections of objects and the shaded ovals represent associations. The shaded part of the collection box represents the member type of that collection. Since the OM model is designed to support database development from conceptual design through to implementation, we use an OM data management framework, OMS, to implement our model within the iDoc framework presented in Section 6.

4.1 Mapping from Digital to Physical

Our mixed model is based on two distinct parts representing metadata coming from logical structures, referred to as digital documents and represented in the upper-right part of Figure 4, and paginated formats, referred to as physical documents and represented in the lower-left part of Figure 4.

Every time a digital document is published, a new object of type `document` is added to the `Documents` collection. Documents are identified by a unique `ID` and information about the creator of the document is also stored. Since digital documents may be updated, we define `revision` objects bound to the source documents by an ordered association `HasRevisions`. In this way, the same document may have different revisions ordered by an increasing revision number, each of them defining different content. A revision may be seen as a milestone which is defined every time a digital document is printed. Through revisions, we actually keep a static representation of the document at a specific time. In Section 6, we will show what is involved in defining revisions.

Each revision of a document contains one or more `element` objects as defined by the `ContainsElements` association. Elements define a `version` attribute which, together with the element `ID`, uniquely identifies a version of the same element. The classification of elements is based on their type. Our model distinguishes three possible types of elements: text elements, graphical elements and multimedia elements (e.g. flash animations, videos). Multimedia elements are normally rendered as graphics in the printed version. For every type of element, a corresponding collection is defined. The collections of different types of elements are constrained by a partition constraint, which ensures that these collections have no common members. Because of the different structure of vector graphics elements and graphical resources (such as jpeg, tiff and gif), graphical elements are further divided into `GraphicsResources` and `VectorGraphicsElements`. While `element` objects define just basic fields which are applied to all kinds of elements, to classify the different types of elements, we define four new subtypes for the specific text, graphical or multimedia elements. These types extend the `element` type and contain a combination of the inherited fields and all attributes of the specific type, for example `text content` and `encoding` for text elements. Similarly, as in many of the systems presented in Section 3, our model enables the definition of more general attributes using key/value pairs stored in `property` objects within the `Properties` collection.

By analysing the digital source document at publishing time, it is possible to create elements for every defined object and place them into the `Elements` collection. Using annotation techniques to enrich PDLs like the ones defined in Section 3.3, it is possible to tag single elements of the digital document and retrieve their position by analysing their physical description. Every element may be described by a PDL snippet which defines its physical position, as shown in the examples presented in Figures 1–3. This information is stored as a `pdl` object in the `PageDescriptionLanguages` collection and is bound to the corresponding logical elements by the `HasPhysicalDescription` association. Even though the physical representation of the elements is encoded according to the PDL used, a shape defining the exact layout and position within the page is calculated for every element. Different shapes are defined and every PDL snippet is bound to the specific shape by the `RepresentedBy` association. To define different kinds of shapes, the `shape` type has several subtypes such as `rectangle`, `polygon` and `circle`.

To encode the physical representation of the elements, we selected three main types of PDL: PDF, SVG and XPS. Of course the model is extensible and it is possible to add other PDLs at any time. Depending on the PDL used, different information is stored and different objects created. For example, to store a PDF snippet, a `pdf` object must be created and placed into the `PDFSnippets` collection. The three PDLs define their own types extending the `pdl` object. To represent the fact that a `pdl` object must be encoded in one of the three defined formats and cannot be placed in multiple collections, we define a partition constraint over the PDL subcollections.

4.2 Element Granularity

One major issue of the logical models outlined in Section 3.1 was defining hierarchies of objects. Many of the proposed approaches use the concept of a tree where a high-level object may contain several lower-level objects. This approach may be taken further, resulting in a recursive element hierarchy. We represent this behaviour in our model through a composite design pattern. As illustrated in Figure 5, elements may belong either to the `AtomicElements` collection, if they do not contain lower-level objects, or to the `CompositeElements` collection in the case that they do. To model the fact that elements may not belong to both collections, we define a partition constraint over them. To build the hierarchical structure, the `ComposedBy` association is defined.

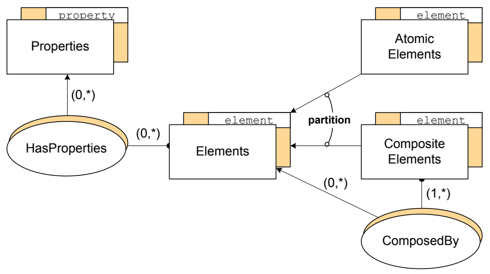

Figure 5: Elements

One of the key steps during the publishing process is the identification of high-level objects through the analysis of the source document. Based on the logical model of the source document, it is relatively easy to discover paragraphs, sections and even chapters of a document. Taking further the approach of non-tree structures stored in the leaves of the tree proposed in [13], we define elements at the highest possible level of granularity and store them as atomic elements within the model. This dramatically reduces the number of objects that we initially have to store in our model. However, we have already declared our intention to create a mapping from physical objects at any level of granularity. Because a low level of granularity, for example character level, is often not required, the idea is to store elements and their metadata at the highest possible level of granularity and to dynamically define composite elements only when needed. For example, if we want to access an element having a lower level of granularity than that of the current element, the physical description encoded in the PDL snippets lets us dynamically split the element into subelements transforming it into a composite element. This involves changing the classification of the main `element` object which is moved from the `AtomicElements` to the `CompositeElements` collection and defining a subelement hierarchy until the required level of granularity is reached. This approach supports the nesting behaviour described for example by Scribe [25] in a much more dynamic and space-saving manner.

5. STRUCTURED HIERARCHIES

In our main model, we distinguish between different types of elements (text, graphical and multimedia) and also between atomic and complex elements. However, within the same type of elements, different granularity levels may be defined. While the types introduced until now may be seen at a level of collection or class, when we speak of different granularity levels within the same type, we are actually defining typing at an instance level. A text element may be defined at different granularity levels such as a paragraph, a word or a character, but it still remains an object of type text. Similarly for vector graphics objects, a rectangle, a line and a point are different levels of granularity within the same type. This issue is addressed in our model by defining a special collection `Types` and an association `HasType` as shown on the left-hand side of Figure 6. By defining specific objects within the `ElementTypes` collection and linking them with the different `element` objects, we are able to assign the right level of granularity to every element instance. `TextElements` objects could be bound to section, paragraph, word or character types, while for `VectorGraphicsElements` there is the possibility to choose between rectangle, circle, line, point, etc.

This approach supports document structures in a similar manner as textons and graphons [9]. However, our solution supports any kind of dynamically generated hierarchy and the corresponding types and may therefore be seen as a generalisation of the textons and graphons approach.

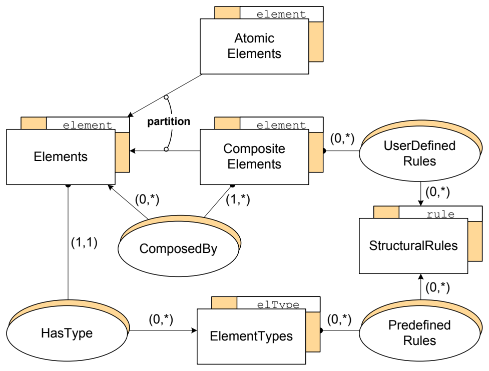

Figure 6: Structural rules

Once the element instances have been assigned to a type, the nesting behaviour represented by the composite pattern becomes more complex. The relationships between elements at different granularity levels must be properly specified. In other words, we must ensure that the nesting of elements occurs in a sensible way. This was achieved in Scribe [25] by defining constraints on how the different environments could be nested. In `pedtnt` [13], a grammatical specification of the object interrelationship was used. We also define such a mechanism for our model by introducing structural rules. Structural rules act on `ElementTypes` by means of the `PredefinedRules` association. Rules define the valid container and the valid content for specific element types. After defining objects of type `paragraph`, `word` and `character`, we can for instance specify a `rule` object allowing paragraphs as possible containers and characters as possible content of the element of type word. To model a greater flexibility in terms of constraints, the `userDefinedRules` association permits the specification of rules based on a specific instance of an element rather than on its predefined granularity level. Using user-defined rules, we could for example specify that a special rectangle cannot contain circle elements.

6. PUBLISHING FRAMEWORK

The model presented in Section 4 and 5 supports efficient storage of the document's information required to map its physical instance to the digital counterpart. To produce interactive documents and get access from paper to the digital objects, we provide the interactive paper publishing framework shown in Figure 7. Our framework is implemented in Java and is based on the storage of metadata about the physical and digital representation of the document by means of the OMS object-oriented database suite, which fully exploits the OM model described before. In the past there have been many proposals for handling the storage of structured documents within databases as described for example in [27]. However, our approach is different since the goal is not to store the whole document in the database, but just its metadata and define references to the original document. The usage of object-oriented concepts both in the programming language and database system allows us to radically decrease the impedance mismatch problems, thus reducing the need to continuously map objects back and forth from the application to the storage layer.

Our system is composed of a core component called iDoc that enables the mapping between paper and digital document instances. iDoc is tightly bound to the *iPublish* layer, which enables the authoring of the different kinds of documents, and to the iServer/iPaper framework [29], which supports interaction from paper documents.

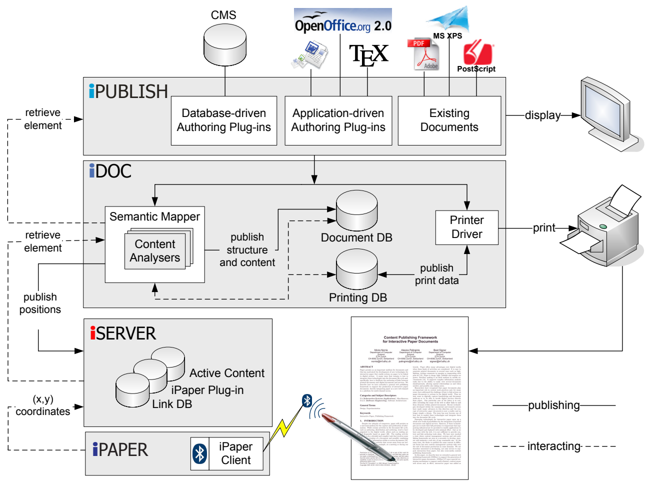

Figure 7: iDoc publishing framework

The iPublish layer consists of a series of plug-ins defined for different kinds of documents. As outlined in Section 1, we may classify documents in three categories. For every document category and its related authoring tool, we define an iPublish plug-in to track structured elements from within the authoring tool or by analysing existing paginated documents. In previous work, we already defined an iPublish plug-in for a database-driven publishing of interactive booklets for an arts festival [21] and a PDF-based text extractor for scientific publications [23].

After being printed on paper, a document loses all connections with the digital interface. The iServer and iPaper frameworks form the basis for building the bridge between the physical and digital world. As described in [22], iServer enables cross-media linking between physical and digital resources. The framework supports the definition of entities in the form of general resources and selectors to address parts of the resources as well as the definition of arbitrary links between these entities. The specific iServer resource plug-in for interactive paper documents is called iPaper [29]. The iPaper framework allows the definition of links between specific active areas on different pages of paper documents and digital functionality. The *iPaper Client* is responsible for communicating with the hardware device physically interacting with the paper sheet and transforming the captured data into the neutral page and (x,y) format to be handled on the server side. In the current implementation, the client is connected over Bluetooth with a digital pen based on Anoto functionality.

The iDoc framework deals with the interactive paper document publishing process as well as the mapping of objects back from paper documents to their digital counterpart. As we will see in the example presented later in this section, these two tasks are tightly coupled, since mapping back from paper depends on the publishing and printing process. The core element of the framework is the *Document DB*. This database implements the functionality needed to support the model defined in this paper. All the metadata about the logical and physical representations of the printed documents are stored in it. The *Printing DB* is responsible for storing information about the technology used to print the instances of the document. For documents printed with the Anoto technology, the specific pattern used and all related information are stored within this database. A *Printer Driver* is provided in order to enable the printing on-demand of the interactive documents. Depending on the technology used to print the interactive document, the printer driver generates the augmented version of the physical document. The architecture of the printer driver is flexible and allows the definition of interfaces for different augmenting technologies. The current implementation provides a driver for printing Anoto-enabled documents on the fly, as shown in [22], but could easily be tuned to print documents based on other technologies. The last element of the iDoc framework is the *Semantic Mapper*. The role of this component is to analyse the structure identified by the iPublish plug-ins during the publishing of the source document, to define the logical elements and store them within the document database and to track the elements within the physical representation of the document defined by its paginated version. The semantic mapper also computes the position of the different elements and their shape, publishes them within the iServer/iPaper framework and defines the link with the logical elements stored in the document database. Depending on the source document, the iPublish plug-in used and the paginated format provided, different *Content Analysers* may be provided. In [23], a specific analyser for scientific references within PDF documents was presented.

To better explain the process of publishing an interactive paper document and the mapping between its physical and digital instances, we present a practical example where a document is defined with the aid of a classic authoring tool such as OpenOffice and published using our framework and the XPS page description language. Document revisions are supported at the level of the authoring tool.

We outline how our system enables the interaction from an Anoto-enabled paper version of the document through a digital pen. The single physical elements are accessed from paper and the corresponding objects within the digital document instance are retrieved.

The publishing process starts within OpenOffice where the iPublish plug-in analyses the structure of the document and identifies text elements at paragraph level and graphics resources. It creates a paginated version of the document using XPS and encodes structural information enclosing the identified elements within two XML elements: `<Paragraph/>` and `<Graphics/>`. Information about the identity of the digital document (document ID, version, creator, etc.) is also stored within the XPS file. The paginated file is then forwarded to the iDoc framework which sends a copy of it to the printer driver and another copy to the XPS content analyser.

The printer driver retrieves information about the available Anoto patterns from the printing database and prints the Anoto-enabled version of the document. All information about the used pattern along with metadata regarding the source digital document are sent back to the printing database. At the same time, the semantic mapper contacts the document database and checks for the existence of the current document on the basis of the document identity stored within the XPS file. If the document already exists in the database, then a new revision is created. All the elements contained within the old revision are also associated with the new one and no elements are replicated. A new version of an `element` (that is a new object with the same `ID` attribute and a higher `version` number) is created for every element that was updated in the digital representation. In this case, the associations between the revision and the old elements are removed and new associations with the new elements are created. The semantic mapper then parses the XPS file identifying the newly defined elements, stores the corresponding XPS snippets within the document database, creates the logical elements assigning the relative type to them and creates the associations between the paginated snippets and the logical elements. It also defines the granularity level of the text elements as "paragraph". After that, the semantic mapper calculates the positions and shapes of the elements by means of the XPS snippets and publishes this information to the iServer/iPaper framework, creating a link between the physical element's position on paper and the element instance stored within the document database. An overview of the complete publishing process is summarised in Figure 8.

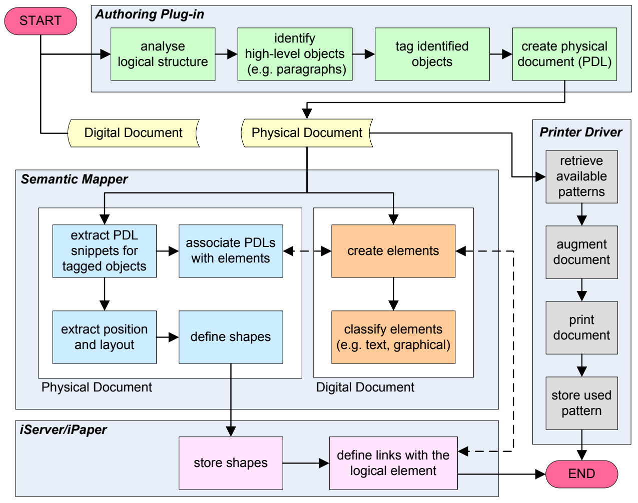
START
Authoring Plug-in
analyse logical structure
identify high-level objects (e.g. paragraphs)
tag identified objects
create physical document (PDL)
Digital Document
Physical Document
Printer Driver
retrieve available patterns
augment document
print document
store used pattern
Semantic Mapper
extract PDL snippets for tagged objects
associate PDLs with elements
create elements
extract position and layout
define shapes
classify elements (e.g. text, graphical)
Physical Document
Digital Document
iServer/iPaper
store shapes
define links with the logical element
END

Figure 8: Publishing process

Once the interactive Anoto-enabled paper document has been published, the user underlines a particular word on the paper document with a digital pen. The iPaper Client retrieves the position of the specified element and forwards it to iServer with its interactive paper plug-in which looks it up in its internal link database. iServer forwards the information about the document, the page and the position pointed to by the user to the semantic mapper. The semantic mapper analyses the metadata of the specified document, contacts the printing database in order to identify the right digital document and retrieves the information about the document from the document database. Based on the information stored in the document database at publishing time, the framework identifies the corresponding paragraph element within the document database. Since the user wants to retrieve a word within the source authoring tool, the semantic mapper splits the element into its subelements. It analyses the XPS snippet bound to the object retrieved, extracts the single words contained within the paragraph and calculates their positions and shapes. The element retrieved is moved to the `CompositeElements` collections and a set of atomic elements at word granularity is stored within the database. For every word element, a new XPS representation is created, stored in the `XPSSnippets` collection and associated with the new logical element. Once the document database has been updated, the semantic mapper retrieves the correct word element based on the relative position within the selected paragraph element and calls the OpenOffice iPublish plug-in responsible for the publishing of that document. The OpenOffice digital document is opened and the corresponding paragraph is found. Based on the word selected by the semantic mapper, the iPublish plug-in finally identifies the underlined word within the paragraph and highlights it.

7. DISCUSSION

Over the last 30 years, we have witnessed the effective exchange of documents between different people around the world using a wide range of software and operating systems. The exchange of documents was first supported by paginated document versions, but nowadays there is evidence that standards are also converging in the world of logical structures and models. Many of the models outlined in this paper successfully define the logical and the physical structure of a document and there is no reason to change them. The introduction of interactive paper technologies raises problems regarding the mapping between a physical version of a document printed on paper and its digital instance represented within the authoring tool. Even though both the physical and logical representation models of a document contain sufficient information about the two different instances, better integration of these models would be desirable.

The approach presented in this paper takes into consideration the existing logical document representations and describes how they could be integrated in order to bridge the paper-digital divide. The composite elements approach used in the tree-based models starting from Scribe has been integrated into our model, which further supports the dynamic specification of the granularity level. The typing hierarchy introduced by Dori et al. with textons and graphons has also been taken into consideration, enabling the application developer to define their own types at runtime, without having to specify them a priori. The clear separation of the content and style introduced by Reid in Scribe and further supported in standards such as XHTML and XML has been modelled in our approach by means of properties which may be associated to either individual elements or a group of elements depending on their granularity level. The regulation of the nesting behaviour outlined by both Reid and Furuta has been addressed by means of structural rules. Our approach acts on two levels—element instances and element classes—leaving it to the developer to decide how restrictive the constraint on the nesting behaviour of the elements at different granularity levels should be. The dynamic behaviour of digital documents has been addressed by means of document revisions which map the dynamic updates in digital documents to a static representation in paper documents. A solution for efficiently storing all of the metadata in a dynamic way has been proposed and implemented within an object-oriented DBMS.

Many of the features introduced by the physical models have also been taken into account and a flexible mechanism supporting the combined usage of many of them has been introduced. Emerging standards based on XML such as XPS as well as established standards such as PDF have been studied and are supported by our framework. Existing annotation techniques based on PDF and SVG and new approaches taking XML documents into account have been presented as a way to address the problem of mapping between a digital document and its physical instance.

The introduction of a framework supporting interactive paper documents from the authoring phase through to the publishing phase, and enabling an extended interaction with them, represents an important step towards a powerful infrastructure where users may freely move back and forth between the physical and digital worlds. The mapping from the physical to the digital world may finally be supported at a semantic level, expanding the ordered structure of logical documents also at physical level. The flexible architecture and plug-in mechanism of our framework enables multiple technologies to be supported in terms of authoring tools, physical representation and interactive techniques. The seamless transition from digital documents to interactive paper documents and back supports end-users by hiding all the unnecessary details about the underlying mappings.

We feel that our model and the related framework provide flexible, dynamic and complete support for documents spanning over the digital and paper world. Our solution further enriches interactive paper technologies, by supporting a wide set of formats, models and authoring tools.

8. CONCLUSIONS

We have presented a model and related publishing framework designed to support and enhance the mapping between paper and digital instances of a document. We outlined how a single model located between the logical and the physical representation of the document helps in maintaining this mapping. The framework allows a dynamic definition of interactive paper documents based on a range of existing and future authoring tools and document formats. It also enables printing-on-demand of these documents supporting their integration in a general link server dealing with their interaction at a physical level. The paper outlines how our solution is a step forward towards bridging the paper-digital divide by supporting the publishing of interactive documents in an extended way.

Based on the model and the framework defined in this paper, we are currently working on a set of plug-ins for different authoring tools (e.g. OpenOffice) which will support an enhanced authoring of interactive documents. We are also investigating how to deal with new logical structures (e.g. ODF) and how to better support new paginated formats (e.g. XPS). In terms of our document model, we plan to integrate it with the general link concepts defined by our cross-media platform, thus gaining greater flexibility in the entire link management.

9. REFERENCES

[1] Adobe Systems Inc. *PostScript Language Reference Manual*.

[2] Adobe Systems Inc. *PDF Reference, Adobe Portable Document Format*, February 2006. Version 1.6.

[3] Anoto AB, http://www.anoto.com.

[4] S. Bagley, D. Brailsford, and M. R. B. Hardy. Creating Reusable Well-structured PDF as a Sequence of Component Object Graphic (COG) Elements. In *Proc. of DocEng 2003, ACM Symposium on Document Engineering*, pages 58–67, Grenoble, France, November 2003.

[5] J. Bloechle, M. Rigamonti, K. Hadjar, D. Lalanne, and R. Ingold. XCDF: A Canonical and Structured Document Format. In *Proc. of DAS 2006, 7th IAPR Workshop on Document Analysis Systems*, pages 141–152, Nelson, New Zealand, February 2006.

[6] H. Chao and J. Fan. Layout and Content Extraction from PDF Documents. In *Proc. of DAS 2004, 6th IAPR Workshop on Document Analysis Systems*, pages 213–224, Florence, Italy, September 2006.

[7] 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.

[8] H. Déjean and J. Meunier. Structuring Documents According to Their Table of Contents. In *Proc. of DocEng 2005, ACM Symposium on Document Engineering*, pages 2–9, Bristol, UK, November 2005.

[9] D. Dori, D. Doermann, C. Shin, R. Haralick, I. Phillips, M. Buchman, and D. Ross. The Representation of Document Structure: A Generic Object-Process Analysis. In *Handbook of Character Recognition and Document Image Analysis*, pages 421–456. World Scientific, 1997.

[10] Ecma International. *Standard ECMA-376, Office Open XML File Formats*, December 2006.

[11] Forms Automation System (FAS), http://www.fasgroup.net/.

[12] R. Furuta. Important Papers in the History of Document Preparation Systems: Basic Sources. *Electron. Publ. Origin. Dissem. Des.*, 5(1):19–44, 1992.

[13] R. Furuta, V. Quint, and J. André. Interactively Editing Structured Documents. *Electron. Publ. Origin. Dissem. Des.*, 1(1):19–44, 1989.

[14] C. F. Goldfarb. A Generalized Approach to Document Markup. In *Proc. of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, pages 68–73, Portland, USA, June 1981.

[15] G. Golovchinsky and L. Denoue. Moving Markup: Repositioning Freeform Annotations. In *Proc.of UIST 2002, Symposium on User Interface Software and Technology*, pages 21–30, Paris, France, October 2002.

[16] G. D. Kimura and A. C. Shaw. The Structure of Abstract Document Objects. In *Proc. of the ACM-SIGOA Conference on Office Information Systems*, pages 161–169, Seattle, USA, June 1984. SIGOA Newsletter, 5(1-2).

[17] A. J. Macdonald, D. F. Brailsford, and S. R. Bagley. Encapsulating and Manipulating Component Object Graphics (COGs) using SVG. In *Proc. of DocEng 2005, ACM Symposium on Document Engineering*, pages 61–63, Bristol, UK, November 2005.

[18] Microsoft Corporation. *XML Paper Specification*, 1th edition, October 2006.

[19] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *Proc. of ER'93, International Conference on the Entity-Relationship Approach*, pages 390–401, Arlington, USA, December 1993.

[20] M. C. Norrie and B. Signer. Information Server for Highly-Connected Cross-Media Publishing. *Information Systems*, pages 526–542, November 2005.

[21] 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)*, 2007.

[22] M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proc. of CoPADD 2006, 1st International Workshop on Collaborating over Paper and Digital Documents*, pages 9–12, Banff, Canada, November 2006.

[23] M. C. Norrie, B. Signer, and N. Weibel. Print-n-Link: Weaving the Paper Web. In *Proc. of DocEng 2006, ACM Symposium on Document Engineering*, pages 89–96, Amsterdam, The Netherlands, October 2006.

[24] OASIS Consortium. *Open Document Format for Office Applications*, February 2007. Version 1.1.

[25] B. K. Reid. *Scribe: A Document Specification Language and its Compiler*. PhD thesis, Carnegie-Mellon University, Pittsburgh, USA, 1981.

[26] B. K. Reid. Procedural Page Description Languages. *Text Processing and Document Manipulation*, pages 214–233, April 1986.

[27] R. Sacks-Davis, T. Arnold-Moore, and J. Zobel. Database systems for structured documents. *IEICE Transactions on Information and Systems*, pages 1335–1342, 1995.

[28] A. J. Sellen and R. H. R. Harper. *The Myth of the Paperless Office*. The MIT Press, November 2003.

[29] B. Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Management*. PhD thesis, ETH Zurich, Switzerland, 2006.

[30] P. L. Thomas and D. F. Brailsford. Enhancing Composite Digital Documents Using XML-based Standoff Markup. In *Proc. of DocEng 2005, ACM Symposium on Document Engineering*, pages 177–186, Bristol, UK, November 2005.

[31] W3C, World Wide Web Consortium. *Scalable Vector Graphics (SVG) 1.2*, April 2005. W3C Working Draft.

[32] J. Warnock and D. K. Wyatt. A Device Independent Graphics Imaging Model for Use with Raster Devices. In *Proc. of SIGGRAPH '82, International ACM Conference on Computer Graphics*, pages 313–319, Boston, USA, July 1982.