Advanced Authoring of Paper-Digital Systems: Introducing Templates and Variable Content Elements for Interactive Paper Publishing

Signer, Beat; Norrie, Moira C.; Weibel, Nadir; Ispas, Adriana

Published in:
Multimedia Tools and Applications

DOI:
10.1007/s11042-012-1217-7

Publication date:
2014

License:
Other

Document Version:
Accepted author manuscript

Multimedia Tools and Applications manuscript No.
(The final publication is available at http://dx.doi.org/10.1007/s11042-012-1217-7)

Advanced Authoring of Paper-Digital Systems

Introducing Templates and Variable Content Elements for Interactive Paper Publishing

Beat Signer · Moira C. Norrie · Nadir Weibel · Adriana Ispas

the date of receipt and acceptance should be inserted later

Abstract Over the last decade, there has been an increasing interest in paper-digital systems that allow regular paper documents to be augmented or integrated with digital information and services. Although a wide variety of technical solutions and applications have been proposed, they all rely on some means of specifying links from areas within paper pages to digital services where these areas correspond to elements of the document's artwork. Various frameworks and tools are available to support the development of paper-digital applications, but they tend to either require some programming skills or focus on specific application domains. We present an advanced publishing solution that is based on an authoring rather than programming approach to the production of interactive paper documents. Our solution is fully general and we describe how it uses concepts of templates and variable content elements to reduce redundancies and increase the flexibility in developing paper-digital applications.

Keywords Multimedia authoring · Paper-digital systems · Interactive paper · Templates · Variable content · Cross-media publishing

1 Introduction

The availability of technologies such as Anoto's Digital Pen and Paper technology[1] has resulted in a tremendous growth in interest in various forms of interactive paper applications that can integrate paper with digital services and information [30]. While the commercial

Beat Signer
Web & Information Systems Engineering Lab, Vrije Universiteit Brussel,
1050 Brussels, Belgium
E-mail: bsigner@vub.ac.be

Moira C. Norrie · Adriana Ispas
Institute for Information Systems, ETH Zurich, 8092 Zurich, Switzerland
E-mail: norrie@inf.ethz.ch, ispas@inf.ethz.ch

Nadir Weibel
DCog-HCI Lab, University of California San Diego, La Jolla CA, 92093-0515, USA
E-mail: weibel@ucsd.edu

[1] http://www.anoto.com

sector as well as the research community have demonstrated how these technologies can support a wide variety of applications and studied the potential benefits of managing information across the paper-digital divide, the development of the necessary cross-media applications is more complicated and expensive than for purely digital applications.

A substantial amount of the additional effort required for developing interactive paper applications stems from the fact that current tools are not well integrated with existing document workflow processes. Instead, the realisation of the supplementary interactive paper functionality is often seen as an additional authoring step after the original document content has been created and the layout specified. This results in an unnecessary separation between the design of the artwork and the definition of any augmenting digital information and services.

This separation leads to an authoring process where different tools have to be used in different stages of the development. Typically, this implies that the artwork of a paper document has to be finished before the active paper regions and the corresponding interaction design can be defined. Later changes to the design and layout of an interactive paper document are not supported by most existing authoring tools and require a manual update of the associated interaction design.

Our goal was to develop an integrated authoring and publishing solution based on existing document editing processes. The resulting authoring tool enables users to easily author and edit interactive paper documents with links to a wide range of digital media and services. We discuss the requirements for the advanced authoring of interactive paper solutions and highlight how the introduction of the *template* and *variable content element* concepts leads to higher efficiency and flexibility in the interactive paper publishing process.

We start in Section 2 by describing existing toolkits and frameworks for the development of interactive paper solutions. In Section 3, we give an overview of our interactive paper authoring approach with a special focus on the new template and variable content element concepts. Our advanced visual interactive paper authoring tool is presented in Section 4, followed by a general discussion of our approach in Section 5. Concluding remarks are given in Section 6.

2 Background

A number of studies emphasise the benefits of enhancing paper-based practices with the aid of emerging digital technologies [24, 14, 15, 17]. New paper-digital solutions should respect the unique affordances of pen and paper for information capture while exploiting the superiority of digital tools in terms of information handling and management. In this way, natural forms of interaction are kept and minimal changes in current work practices are required. However, developing interactive paper applications does not always follow the same principle of naturalness. The development process is often cumbersome, inflexible and inaccessible to users without specific programming knowledge. The complex development process usually involves the authoring of the document content, the definition and implementation of digital services and interactions, as well as the association of paper documents with digital services. Earlier approaches exploited the structure or content of the source document and added interactive features at print time. For example, Books with Voices [12] provided paper-based links to digital resources by tagging the printed description with barcodes whereas Origami [23] made use of the hyperlinks stored in HTML documents to enable paper-based links to the linked documents based on the DigitalDesk [39].

A more effective solution that can be used for the capture of handwritten information and the realisation of paper-digital applications is Anoto's Digital Pen and Paper technology. Documents first have to be digitally authored before being printed with the Anoto pattern which is read by the digital pen's integrated infrared camera. The Anoto pattern consists of tiny black dots which encode a position and are almost invisible for the human eye. The pen calculates its position by decoding the recorded pattern as it is moved over the surface of the pages of a paper document. The positional data is transmitted to a computer via Bluetooth or a wired docking station to be processed in batch mode by Anoto software or third-party forms automation solutions.

In order to provide better access to the Anoto technology and offer the corresponding functionality for paper-based interactions, several frameworks have been introduced. The Paper Augmented Digital Documents infrastructure (PADD) [9] developed at the University of Maryland as well as iJITinOffice [11] introduced by the Hitachi Central Research Laboratory of Tokyo address the basic management of Anoto-based documents. To support the development of standard Anoto applications with real-time interaction, researchers at Stanford University developed PaperToolkit [42]. PaperToolkit is a Java framework based on standard Anoto tools. It makes use of these components in order to create an augmented version of a PDF document that can be printed and used to access interactive digital services in real-time through the digital pen. More recently, Livescribe[2] introduced the new Anoto-based Pulse and Echo digital pens. The major innovation introduced by Livescribe is the possibility to deploy software on the pen and support stand-alone applications—so-called penlets—that run on the digital pen. The Letras framework [10] takes an alternative approach towards the development of interactive paper applications. Applications are built based on a distributed processing pipeline abstracting different stages of pen-based input and associating them with different tasks.

These frameworks and tools have been successfully used over the last 10 years to explore digital pen and paper interactions and scenarios such as document editing [7], text processing [37], scientific field and laboratory work [41,34], collaborative annotations [33] and gesture-based interactions with paper documents [26,35,13].

All the presented software frameworks enable the creation of paper-digital applications. Anoto directly supports form-based approaches, PADD and iJITinOffice deal with the management of the documents and the assignment of the pen strokes to the corresponding pages, PaperToolkit and Letras support real-time interaction and linking to digital services from paper regions, and Livescribe introduces stand-alone pen-only applications. Despite their similar intent, current approaches are fundamentally different in the way they address the implementation, deployment and management of the paper-digital applications. In the following, we describe these approaches, outlining how they address the authoring of paper-digital applications and the implementation of the specific functionality. The process of creating a paper-digital application can be separated into a programming phase, a design phase and an authoring phase. Depending on the framework used, these three phases can be intertwined, dominant or non-existent.

Anoto provides specific tools to help designers and developers build applications. Figure 1 illustrates the interplay of the different Anoto tools, highlighting how the development of a form-based application requires the continuous and iterative interaction between designers and developers. The Anoto Forms Development Tool Kit (FDT) [2] allows graphical designers and software developers to create Anoto-enabled forms (PDF documents) by using an Adobe Acrobat plug-in. During the design phase, one can add *pidgets* or *user areas*

[2] http://www.livescribe.com

to the form layout. Pidgets are interactive page elements enabling the user to control specific Anoto functionality. The pidget functionality is predefined as part of the software deployed on a digital pen and cannot be changed. Examples of pidgets are buttons that will send the captured input to a mobile phone or signal the end of the interaction. While pidgets are interpreted by the pen, the application logic bound to active paper areas is defined in a separate Paper Application Definition (PAD) document. To offer a more flexible development process, Anoto released the Paper SDK and the PAD and Print Generation Module (PPGM) [3] enabling software developers to access core Anoto functionality. Once an Anoto-enabled document has been created, the Anoto SDK [1] can be used to implement the application logic, while the Pen API [4] manages the pen strokes that have been collected from the pen and stored in a dedicated Pen Generated Coordinates (PGC) file.

In order to develop a complete Anoto form-based application, the different Anoto tools have to be used in combination. As shown in Figure 1, the prerequisite for developing an application is a source PDF file (1). The development starts with the definition of the interactive fields via the FDT or by using the Paper SDK (2a). At the same time, developers implement the application logic based on the libraries offered by the Pen API and the Anoto SDK (2b). The PDF document is transformed into a PostScript file and augmented with the Anoto dot pattern (3a). A new PAD file is generated (3b) and made available to the application by storing it in a local folder that can be accessed by the Pen API (4). The end user interacts with the dotted paper (5) and the digital pen transmits the collected strokes via a PGC file to the application (6). The application processes the PGC file with help of the Pen API and the Anoto SDK (7) and stores the interpreted strokes in a repository or sends them to an external application (8).

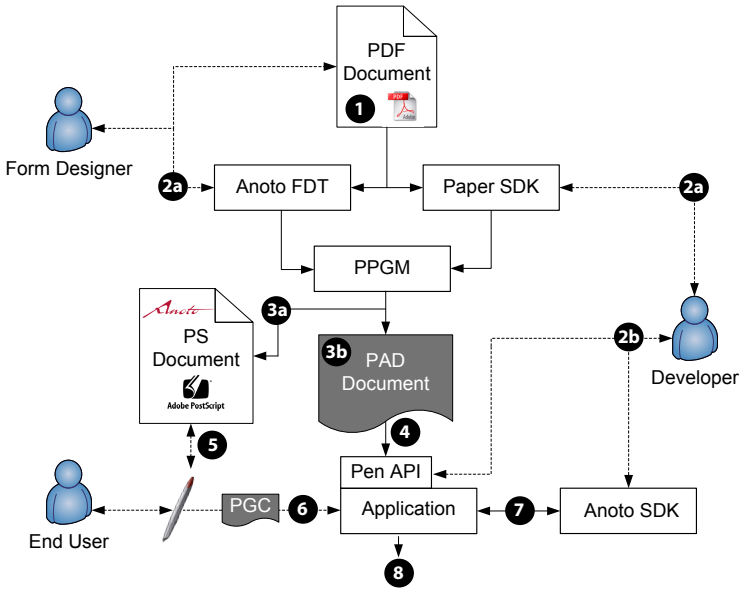


Fig. 1 Anoto tools

The PaperToolkit makes direct use of Anoto tools such as the Anoto SDK and PenAPI to support the management and binding of captured strokes to a range of digital services. It is

based on paper-based events that are handled by event handlers associated with pre-defined paper regions. The development of a new interactive paper application based on the PaperToolkit involves the programmatic definition, implementation and binding of active page areas to these event handlers. In order to work with PaperToolkit, the Anoto pages have to be previously registered within the framework by modifying the PAD file created through the Anoto FDT and including it as a resource in the framework. In order to give more flexibility to both developers and designers, PaperToolkit introduced the *Paper UI Sketcher* [40], a tool that allows designers to sketch simple interfaces on paper sheets and define active regions to which event handlers might be attached. These stroke-based interfaces are subsequently transformed into code snippets that have to be further processed by the application developers in PaperToolkit.

Livescribe supports the development of dedicated penlet applications via the Platform SDK[3], an Eclipse-based IDE that combines a number of specific SDKs. Similarly to PaperToolkit, it supports the development of event-based applications and the association of events to specific areas of a document. The Penlet SDK enables J2ME applications to be developed on the desktop and deployed to the pen. The Paper Designer supports the definition of specific active page areas that have to be defined on top of a static PostScript document loaded into the tool. Active areas can be linked subsequently to specific handlers defined by means of the Penlet SDK before the final deployment of the penlet to the Livescribe digital pen.

The Letras framework introduces a more structured approach to paper-digital application development. At the driver stage, Letras handles the connectivity with different digital pens and transforms raw input to pen-based strokes. The region processing stage deals with the definition of active page areas whereas the semantic processing stage transforms the collected strokes into higher levels of abstractions such as text/drawing segmentation. Last but not least, the application logic is defined at the application-level processing stage. Despite this separation of stages, similarly to the PaperToolkit and Livescribe SDKs, Letras also processes pen-based input based on active paper regions and links them to particular services or applications before deploying the software, making it hard to change any functionality without having to recompile and redeploy the application.

Even though they clearly advanced the state of the art in supporting interactive paper applications, a major drawback of the described development tools is that the design of the interactive paper document and the interaction design are not well integrated. Documents have to be finalised before exporting them to paginated documents which can then be augmented via the Anoto FDT or the Livescribe Paper Designer. If changes have to be made to the original design after the active regions have been defined, the shapes representing these active regions in the Anoto FDT, PaperToolkit, Livescribe Paper Designer or Letras have to be updated manually since there is no direct association between the graphical document elements and the corresponding application logic. Note that while PaperToolkit and Letras support the development of applications with real-time interaction, the Anoto FDT does not. Further, the Anoto Paper SDK, PaperToolkit and Letras all introduce some low-level functionality for application developers, but there is no high-level authoring functionality that allows the designers to create an interactive paper document without programming effort.

Existing interactive paper authoring tools lack a separation between the authoring of active areas and the implementation of the application logic. They follow a programming approach for the development of applications, meaning that the interactions and specific active paper regions have to be implemented as part of the different applications and designers

[3] Livescribe recently discontinued the developer program (see http://www.livescribe.com/developers)

are forced to work in strict collaboration with developers to create a new application or even change an existing application or interface.

The Livescribe Paper Designer and the PaperToolkit Paper UI Sketcher try to address this issue, but the tools provided for authoring interactive areas are still bound to static documents or hard-linked to functionality that has been developed as part of specific applications or penlets. Since the definition of active paper areas is further hard-coded in the application programs, changes in design require a recompilation. Moreover, in most of the existing tools, the implementation of the application logic is mixed with the interaction design which makes it difficult to reuse functionality across different applications. The pipeline approach of Letras tries to avoid this mix of interaction design and application logic and looks promising in that sense. Finally, the use of the low-level Anoto or Livescribe SDK functionality often makes the overall development process more vulnerable to errors than an approach at a higher-level of abstraction.

An alternative approach addressing the shortcomings of existing solutions is our iPaper framework that we describe in the next section. By introducing the concept of templates and variable content elements, we were able to realise a flexible and fully integrated paper-digital authoring tool, supporting an authoring rather than programming approach towards the development of paper-digital applications.

3 Interactive Paper Authoring

In contrast to most existing interactive paper development frameworks, our iPaper solution [21,25,38,36] is based on a data-centric approach with a clear separation of the application logic and the metadata defining active paper regions. The definition of active paper elements is no longer part of the program code but managed by iServer [19], a general cross-media link server in combination with the iPaper plug-in for interactive paper documents as shown in the UML diagram in Figure 2. iServer and its underlying resource-selector-link (RSL) metamodel [27] enable the definition of `Links` between arbitrary digital or physical `Entities`. While a `Resource` is used to represent a particular media type, a `Selector` can be applied to address parts of a resource. A selector always has to be associated with a single `Layer` to define the order in the case of overlapping selectors and our system supports an arbitrary number of layers. Furthermore, for each entity, there has to be exactly one `User` who created the entity and can define its access rights. Please note that Figure 2 only shows a simplified version of the RSL metamodel. Some of the components that are less relevant in the context of this paper (e.g. user management and access rights) are described in more detail in [27].

While the RSL model manages information in a media neutral format based on the main concepts of resources, selectors and links, it can be extended for different types of media. For instance, iPaper resources are represented by `Pages`, whereas the selectors are arbitrary shapes such as rectangles, circles and polygons. In addition to single pages, iPaper supports the concept of `Documents` consisting of multiple pages. Each page has a number of active areas defined in terms of shapes within the page and links to `Active Components`. An active component is a piece of program code that gets invoked when the related link is activated by selecting the corresponding active page region with the digital pen [29]. The active component subsequently becomes the handler for the pen input data until it is unloaded by the program logic.

A wide variety of interactive paper documents and applications have been developed based on iPaper [20,37,22,32]. Figure 3 outlines the authored interactive handouts for a

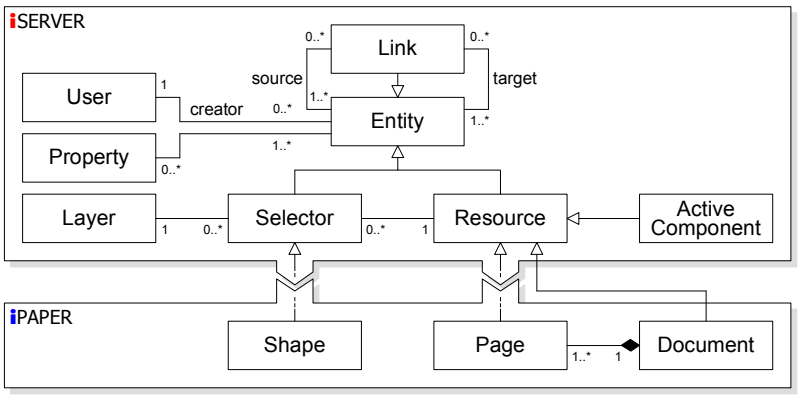


Fig. 2 iServer with iPaper plug-in

particular application called PaperPoint [28]. In the following, we use the PaperPoint application as an example to explain different steps of the authoring process. However, note that the interactive paper authoring solution presented is fully general and not limited to specific application domains. By having a modular system architecture with different device drivers for two-dimensional tracking solutions, we are further not limited to the Anoto technology for paper-digital interaction but can support other tracking mechanisms by simply implementing a new input device driver. In the case of PaperPoint, active components serve as a bridge to Microsoft PowerPoint. The areas corresponding to the six slides on a handout page are defined as capture areas, as shown on the right-hand side of Figure 3, and any handwritten annotations on a printed slide are transferred in real-time into the digital slideshow. The user can move to a specific slide by simply touching the slide thumbnail with the pen or by selecting the `Show` button printed below each slide. Furthermore, at the top of each page, there are active areas linked to actions such as changing the pen's digital colour and moving to the first, last, next or previous slide.

Developers can focus on the functionality to be handled by a single active component resulting in a component-based architecture where specific functionality is encapsulated in small modular units. In addition to application controls, we have developed a wide range of general active components, including paper-based GUI widgets such as buttons, input fields and slider components as well as active components without a particular relation to GUI elements. Active components can be reused across different applications and the growing library of available active components supports the rapid prototyping and authoring of new interactive paper applications. Furthermore, non-programmers can develop interactive paper applications by simply authoring active paper areas (arbitrary shapes) and linking them to existing active components via the visual authoring tool presented later in Section 4. In addition, each shape can be associated with an image to define a graphical page element. The image that is linked with a particular graphical page element is going to be used as the shape's background when the document is printed. This brings advantages compared to other approaches since it allows us to link the design of the page content and layout with the corresponding interaction design in a more flexible manner. If, at any stage in the development phase of an interactive paper application, the designer is going to rearrange some graphical page elements, all relevant information about the position of the associated active paper area (with its links to active components) is updated automatically.

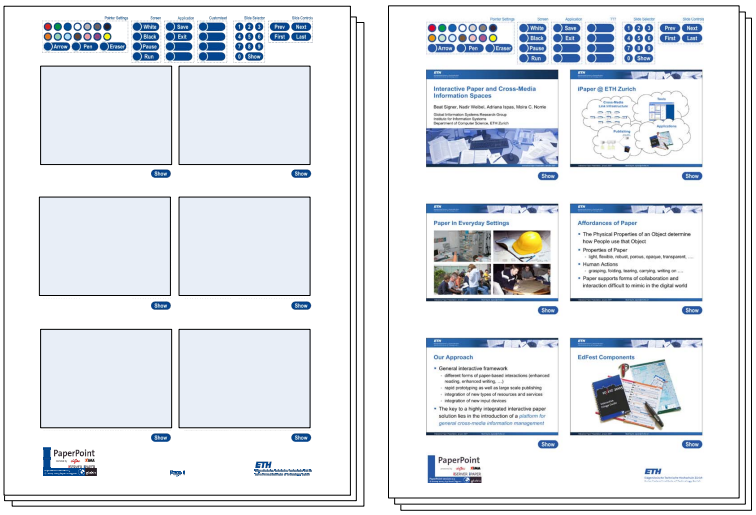


Fig. 3 Interactive PaperPoint handouts

Our iPaper platform supports different forms of automatic or manual link authoring but the resulting metadata about active paper regions and the corresponding linked services is always stored in an iServer/iPaper database. Print-n-Link [22] is an example of an application where paper links are generated automatically based on an analysis of arbitrary PDF documents. However, for most applications, links need to be authored manually either by editing an XML file or using our visual authoring tool. In applications such as PaperPoint, where the number of pages and active areas can be large and designers want to experiment with different layouts and functionality, this authoring process can be tedious and consist of many repetitive steps. To simplify the authoring process, we therefore introduced the notions of templates and variable content elements as described in the remainder of this section.

3.1 Templates

We introduced a template mechanism so that active paper elements that occur on multiple pages can be defined once and be applied several times rather than having to repeat their definition on every single page. Our solution supports templates during the design of the graphical and textual page content as well as during the related interaction design. By adding templates to the RSL model, each resource can be associated with one or more template instances. For the iPaper plug-in, this implies that templates can be defined on the document level as well as on the page level, since both of them are resource subclasses. While the definitions on a document template are applied to every single document page, a page template might be associated with individual pages within a document. In addition to the extension of the RSL model, we also had to define the behaviour in the case that multiple templates are associated with a single page. When a specific page of an iPaper document is loaded, the system first checks whether there are any document-level templates and applies them to the current page. In a second step, all associated page-level templates are processed for the given page. Both document-level and page-level templates further make use of the

layer concept and in the case of overlapping shapes, the layers define the precedence rules necessary to resolve "conflicting" active page areas.

Note that our template approach not only simplifies the creation of the artwork, but also supports the definition of active areas (shape) together with links to relevant digital services. This implies that the application logic of active components that are linked from a template has to be aware of the new template mechanism. To explain why we need template-aware active components, let us have a closer look at one possible interaction with the PaperPoint application. Each PaperPoint handout page shows six slides. If a user starts to draw with the digital pen on one of these slides, the PowerPoint presentation immediately switches to that slide and any paper-based annotations are mapped to the digital slide version in real-time. In order to do that, each active component that is linked to one of the six slide selectors also needs the corresponding slide number as a parameter. In a non-template-based PaperPoint implementation, each handout page would have a new set of six slide shapes with links to different active components with individual metadata such as the slide number. For example, the active component that is linked to the upper left slide region on the first handout page has a `slide number` parameter with value '1', whereas the active component that is linked to the upper left slide region on the second page has a value of '7' for the slide number. If, on the other hand, the six slide regions are defined in a document template, we can no longer provide the slide number as a parameter of the active components. The six active components that are linked from the PaperPoint document template will only provide their relative positions, for example a number between '1' and '6', as their parameters. The newly designed template-aware active component has to perform some additional computation to get the required slide number. It first retrieves the current page number p, multiplies the number of slides s on a single page with $p-1$ and finally adds its relative position to compute the slide number. While a template can be used across multiple documents, we currently mainly use them for reducing the redundancy within single documents. However, templates that are going to be used across multiple documents might demand even more for different user roles. The template designer is going to develop the document or page templates and normally needs some basic programming skills whereas the end user of these templates only has to provide some configuration parameters in order to make use of them.

Templates support the development of an interactive paper application in two different ways. We can use them as initialisation templates when a new page is created. This means that, at page construction time, the user can select a specific template and all the selectors and links defined on the template will be *copied* to the new page. This approach may be used if the page layout or content is not completely identical but a developer wants to start from a common source defined by the template. Note that since all elements are copied from the template, this use of templates does not help to avoid any redundancies.

In a second approach, templates are associated with a given document or a number of pages and the elements that have been defined in the template are shown on every single page that has been associated with the template. Unnecessary redundancy is avoided by no longer copying the template metadata, such as the shapes of active page regions and their associated links, to every single page. Templates can be defined on the document as well as on the page level, which gives a developer the freedom to add templates to a selective number of pages or to associate specific pages with multiple page templates.

When we developed the original PaperPoint version, the presented template mechanism was not yet available and the shapes for buttons and slide regions had to be stored redundantly for each of the ten slide handout pages. A later template-aware PaperPoint version makes use of a single document template that is automatically applied to all of the slide handout pages. A first positive effect of the re-authoring based on the new template mech-

anism is that the size of the iServer/iPaper database could be reduced by almost a factor of the number of slide handout pages. Even if the database size was not a critical issue, the introduction of the document template mechanism implied that less redundant information has to be stored in the database which enhances the overall data management. The use of templates further simplified the addition of new slide handout pages when we decided to increase the maximum number of PaperPoint slide handout pages to sixteen (e.g. a maximum of 96 slides), since we only had to add some empty pages making use of the same document template. Note that the decision to currently support up to sixteen slide handout pages is mainly based on the fact that most presentations do not contain more than 96 slides and we did not want to reserve too much of the Anoto pattern space. However, the maximal number of slides could easily be extended if necessary.

Another advantage of the new template-based PaperPoint implementation is that it has become much easier to change the design of the slide handouts such as the position of a single button. By simply moving the picture of a button on the document template, its associated active region is updated in the database and all handout pages are automatically aware of the template update without any further manual intervention. At some point, we wanted to increase the maximal number pages by adding six handout pages. In an earlier non-template-based version, this would have required us to copy and paste the shape information from an existing page followed by a manual update of the active component parameters such as the slide number parameter. However, in the new template-aware solution, we only had to add six new pages to the document and no additional authoring was necessary at all since the new pages immediately inherited the shape and link content from the existing document template.

3.2 Variable Content Elements

The production of interactive paper documents often involves the authoring of pages that are similar in their structure and layout, but different in terms of their content. An example of these similar layout components for the example of PaperPoint is shown in Figure 3. The different paper buttons to control the application are repeated on every handout page as shown on the left-hand side of Figure 3. On the other hand, the content of the empty boxes representing the slots for the resized slide thumbnails, might differ for every new printed job. From a content publishing point of view, the repeating elements have to be seen as *static content* which is defined at authoring time, whereas the content of the handout regions to be added only at printing time can be referred to as *variable content*.

A major issue in personalised content publishing is the integration of static and variable content at printing time. To solve this problem, one could use an approach based on the overprinting of preprinted forms with variable data. Such an approach was actually used in earlier versions of the PaperPoint application. However, this introduces a number of problems ranging from practical printer paper loading issues to calibration problems between the preprinted and variable content. Furthermore, the printing of an interactive paper document requires an exact mapping between the shapes defined in the database and the printed artwork and even a small displacement during the printing process can negatively affect the paper-digital interactions.

A better approach is to automatically combine static and variable content elements at print time. While the static content is usually defined and instantiated at authoring time, variable content is defined at authoring time but instantiated only at printing time. As we briefly introduced earlier, static content can be represented by shapes that are associated

with the desired graphical content—such as the image representing the `Next` button in the PaperPoint example—through the definition of shape properties. However, to support variable content, the iPaper data model has been slightly extended as highlighted in Figure 4. The new `VariableContent` (VC) resource can be linked to a pre-defined shape and represents the variable information that is filled into the shape at printing time. The abstract `VariableContent` class defines the overall structure of this resource which is based on two main components: a `ContentExtractor` able to interact with different content resources at their APIs and a `process()` method that knows how to process the resources fetched by the extractors. The `VariableContent` class can be extended by concrete classes that instantiate specific `ContentExtractor` classes and implement the processing logic. Figure 4 shows three examples of specific variable content implementations: a `DocumentPageVC` handling content extracted from a document in a page by page manner, a `PictureAlbumVC` dealing with content extraction from picture albums such as Picasa, Flickr and Facebook, and a `DatabaseVC` retrieving content from a database. Thanks to its generality and extensibility, the software infrastructure supports the definition of arbitrary variable content elements able to process new content resources or combinations of different sources. Note that the layers introduced earlier can be used to control the order of shapes in the case of overlapping active regions. If static content has been linked to shapes on different layers, the ordering of the layers defines the order of the graphical elements in the final artwork.

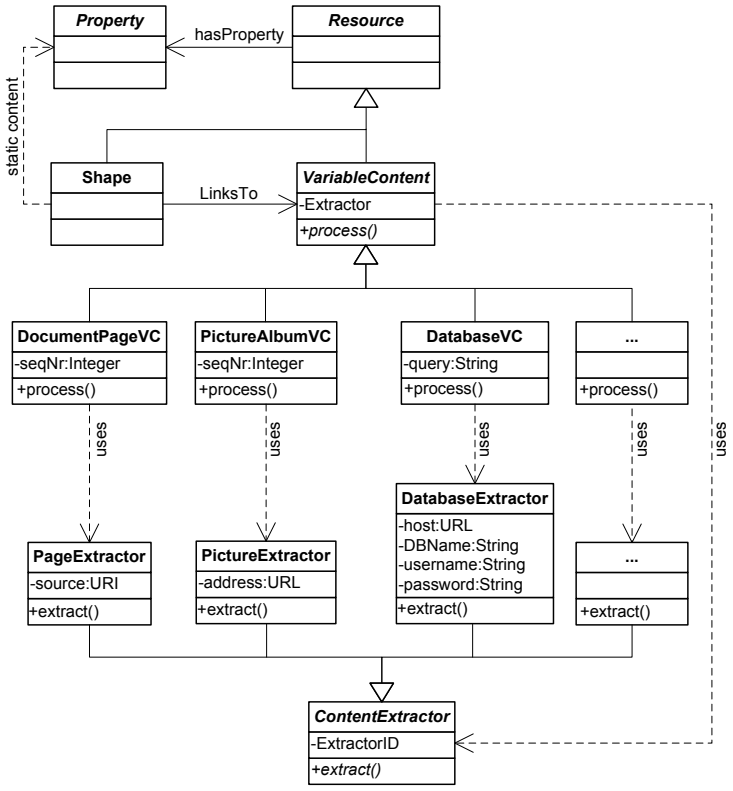


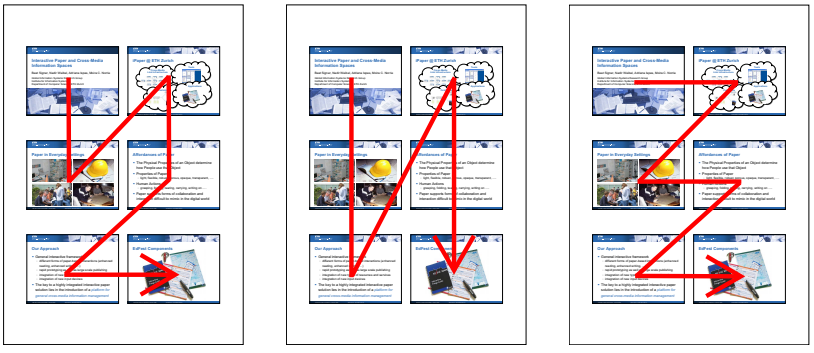
Fig. 4 Variable content concepts

In order to insert data into the placeholders defined by the authored shapes, the variable content must be extracted from different sources. The `ContentExtractor` elements outlined in the UML diagram shown in Figure 4 implement this functionality by means of an Abstract Factory design pattern. Every variable content element makes use of a specific content extractor and delegates the content extraction to it. The extractor knows exactly how to deal with a given source and delivers the necessary content to the variable content element. Each instance of a content extractor contains some metadata about the resource to be accessed such as the URI of a file or information on how to establish a connection to a specific database. The `DocumentPageVC` variable content element and the corresponding `PageExtractor` are used in the PaperPoint application to render content extracted from a source PowerPoint presentation by iterating over it page by page.

Since multiple shapes on a single page might link to variable content elements associated with the same source, variable content elements can define specific parameters to drive the extraction of the content and add it to the correct placeholder at print time. This enables the designer to specify how data from the source should be fed into the variable content placeholders. For example, in the case of the `DocumentPageVC` and the `PictureAlbumVC`, a `seqNr` attribute has been defined by the designer to specify the order in which the source pages have to be rendered within a single page of the artwork. Based on these additional parameters, different variable content element layouts can be defined at authoring time.

Figure 5(a) shows an example of a specific layout, emphasising how designers can define an arbitrary ordering of the variable content elements. In most cases, a more regular layout is used and tools that automate the definition of the sequencing are required. Figure 5(b) and Figure 5(c) show such a possible automatic approach for the definition of standard *vertical* and *horizontal* slide ordering. Note that more complex ordering patterns might be defined based on additional variable content elements and content extractors.

Having defined all necessary data for static and variable content elements at authoring time, we have to assemble it in the final paginated document which is then augmented with the Anoto pattern and sent to the printer. We use the Apache XML Graphics Commons Java library [5] to create the PostScript file and then add the Anoto pattern in a subsequent step by using our special Anoto Virtual Printer [21]. As shown in Figure 6, our publishing engine first fetches all the shape information from the iServer/iPaper databases. Depending on the number of pages defined within the document, a new empty document is created (1). Based on the static information associated with shapes, the publishing engine retrieves the



(a) Sparse (b) Vertical (c) Horizontal

Fig. 5 Variable content ordering

required file or web resources and inserts it at the shape's exact position (2). In a next step, any variable content is retrieved and inserted into the document. In the case of PaperPoint, the content source consists of a single PowerPoint document. Since PaperPoint defines variable content by using the `DocumentPageVC`, the publishing tool extracts the variable content page by page from the PowerPoint document. The single slides are resized to the slide shape's dimension and added to the paginated document (3). After all the static and variable content has been inserted into the document, the Anoto pattern is added (4). Finally, the newly created interactive document is printed (5) and used to interact with the PaperPoint application (6).

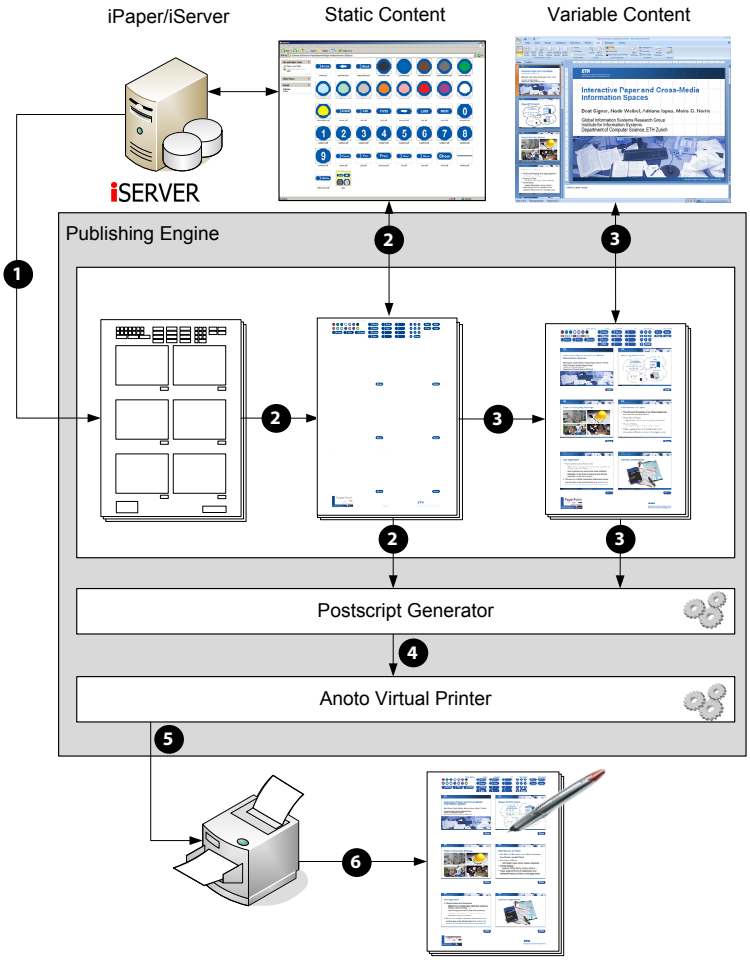


Fig. 6 Printing architecture

4 Authoring Tool

The authoring of interactive paper applications should be possible for users with different backgrounds, including interaction designers and artists, and not just programmers. We therefore developed a visual authoring tool based on the concepts presented in the previous sections. The tool supports the complete interactive paper workflow, covering document design, the definition of interactive paper areas and their associated digital services, as well as the printing of Anoto-enabled documents. Active page areas can be defined based on a WYSIWYG approach in the form of various geometrical shapes superimposed on document pages. The active areas can be linked to a wide range of digital resources such as videos or pictures, as well as to active components for further processing of the pen input. The authoring of interactive paper documents is implemented as a plug-in for a more general cross-media authoring tool [31] as shown in Figure 7.

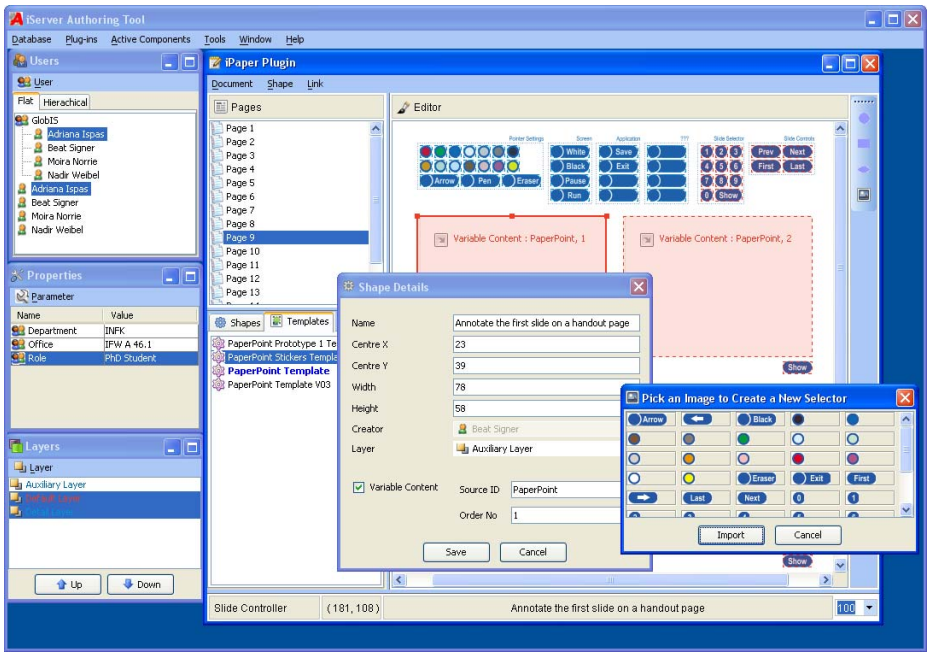


Fig. 7 iServer authoring tool with iPaper plug-in

The authoring of an interactive paper document starts with the definition of a new document, which is done via the `Document` menu. The menu provides support also for the handling of existing interactive paper documents, such as opening an existing document, modifying some document properties, as well as printing an Anoto-enabled document version. The upper left panel of Figure 7 shows all the pages of the currently edited document and various features for managing single document pages are provided via this page panel. The selection of a page via this panel results in the rendering of the page content in the central panel.

The `Templates` tab, shown in the lower left panel of the plug-in window, lists the names of all the templates defined in the database and highlights the ones that are applied to the

currently selected page (in the Pages panel), including the document-level templates. A template can be added or removed at the page or document-level by right clicking the desired template in the Templates tab and invoking the corresponding action from a drop-down menu. The selection of an already associated template in the Templates tab results in showing its shapes superimposed on the current page view in the central panel, with the shapes highlighted in accordance with their corresponding layers which have to be selected from the Layers tab. When adding a template via the contextual menu, the template is implicitly selected and its shapes are highlighted in the page view. In this way, users can quickly inspect a template's structure and subsequently remove it from the same contextual menu in the case that the template should not satisfy their needs. Finally, the complete set of shapes that have been defined for the currently selected page—either on the page itself or via a template—is shown in the the Shapes tab. Note that templates can be defined by the end users but we might also introduce the role of a template designer who has detailed knowledge about the document structure. While templates are currently selected based on their name, in the future the template selection process might be enhanced by a textual template description or a visual preview of a template's content.

If none of the existing templates suits a user's design intentions, they can simply create a new template by switching to a template mode from the Templates tab and defining its content in the central panel. Several buttons shown in the right-hand side toolbar enable the insertion of various types of predefined active area shapes. Once a new shape has been added, it can be moved around, resized, attached to different layers or, very importantly, linked to other content such as active components. The Shape Details dialogue window presents a shape's attributes, all being configurable except for the creator who is fixed and defaults to the editing user. Further, shape-based active areas can be marked as variable content as shown in the lower part of the dialogue. Information about existing variable content resources is fetched from the underlying database and the available options are presented to the user. Depending on the type of variable content chosen, additional parameters have to be specified. For example, in the case of our PaperPoint application, users have to specify a source PowerPoint document identifier and the sequence information. To create a link, a similar dialogue presents a list of available resources which become the link's target when selected.

The tool also provides support for adding predefined artwork on a document page. Imported artwork snippets define implicitly active page areas, which can be further linked to active components or other resources. A designated button on the toolbar is used to bring up a dialogue window showing all artwork snippets already attached to the current document and additional images can be imported and automatically inserted into the document's resource collection. The URI of an inserted image is saved along with the shape implicitly created based on the imported content. The URIs are used at printing time when rendering the interactive paper document. Creating active areas based on this approach has the benefit that modifying the artwork and the corresponding active area is done simultaneously. If an artwork element is used for design purposes only, then the author has nothing to do and no resource link is created. The buttons mentioned in the case of our PaperPoint example application, which are shown in the upper part of the page currently loaded in the authoring tool in Figure 7, are active page areas created based on such an authoring approach.

While a regular shape may be edited via its visual representation in the main panel without any constraints, the modification of a template-based shape results in different alternative operations. As soon as a user attempts to modify a shape that is part of a template, a dialogue window prompts the user to switch to the template mode. Furthermore, if the template has been either applied at document level or used for other pages than the currently edited page,

the user is informed that changes will affect other pages that are associated with the template and they are asked whether that is their intention. If a user only wants to modify the current page, there are two possibilities. They can either create a new template based on the previously applied one, or the template can be unwrapped and its shapes rearranged as standalone active areas that are no longer associated with a template. The first option corresponds to a kind of *copy-on-work* with a clone of the original template being created for the user to work on. In the case that the starting template was previously applied at the document level and the user's intention is to change the structure of a single page, the starting template will have to be removed from the document level and reapplied to each page except for the modified one. Please note that a clone of the template is created every time that the user attempts to modify a template associated with other documents than the one currently being edited.

5 Discussion

The concepts of templates (sometimes referred to as pagemasters) and variable content elements are well known in both document engineering and web engineering. They are supported not only in existing word processors, but also in web development tools and Web 2.0 platforms supporting personalised websites. However, they have not yet been applied to interactive paper development processes.

We note that word processors and web-based systems that support some form of template mechanism tend to give limited control to the users over the ordering of graphical elements within the template when it comes to the final rendering step. Often the content on template pages is just overlaid by any specific content on the page level. Our layering concept for shapes, which is also available within templates, enables the designer to define the exact ordering of elements within a template by assigning them to the corresponding layers. Our iPaper template mechanism also reduces redundancy in the definition of repeating resources.

Most existing tools for the authoring of interactive paper applications tend to be monolithic in nature and this severely limits their flexibility in terms of support for rapid prototyping as well as extensibility and reusability. By developing tools, such as the one presented in this paper, that support the entire workflow in an integrated manner based on a document-driven approach, it should be possible for non-programmers to not only design interactive paper documents but also develop entire paper-digital applications through an authoring process. Templates and variable content elements, together with an extensive library of active components, provide the building blocks for the authoring process.

Our approach is not only document-driven but also data-centric. This means that in contrast to most of the existing frameworks and tools presented in Section 2, all features of a document including the definition of active areas and their relationship to both the artwork and digital services are defined explicitly as data based on the RSL data model. This makes it easy to dynamically update arbitrary document features at any time.

As we have seen previously, creating an interactive paper application often requires the interplay of different actors. A graphical designer usually creates the paper interface by defining the content and the appearance of the document. After that, the interaction designer normally defines the interactive behaviour that should be supported by the paper-digital interface. Finally, an application developer develops the code necessary to execute specific commands or interactions on the computer. Our solution fully supports this interplay of roles by decoupling the design of the interface, the definition of the active areas, the binding to the digital functionality and the development of the active components. In this way, the

various team members can focus on their own tasks. By using the visual authoring tool described in Section 4, changes in the document layout are automatically propagated and updated through the whole interactive paper design path and there is no need to build the applications anew.

In the last few years, similar problems have been addressed in the document engineering community. With the advent of high quality digital printing, new technologies addressed the increasing need for personalised printing [18] by introducing new forms of handling *variable* information that is assembled at print time. This approach combines the production of *static* elements through standard authoring tools with *dynamic* content-based retrieval of variable information coming from a range of different sources, such as content management systems, databases and web pages. Such a personalised printing workflow is currently implemented for the large-scale publishing and mass customisation of documents and it is known as *Variable Data Printing* (VDP). Novel approaches in this domain support a more segmented workflow, where multiple designers can use different tools to author the artwork and combine them at a later stage. For example, Anvil [8] proposes such a segmented workflow, supported by the use of the specialised standard XML exchange format PPML[4]. A similar segmented workflow occurred during the development of the publishing engine for our interactive paper documents. Unlike Anvil, we wanted to support any tool used by the designer, without having to create plug-ins or add-ons for these tools. In our case, the variable part of the document is represented by means of the stand-alone variable content elements which are created within the authoring tool and instantiated at print time. One key point is that our variable content elements might be complex—it is up to the developer to implement new variable content elements—and combine content coming from different sources in a similar way to that done by Document Description Framework (DDF) documents [16].

The presented solution addresses the issues related to the handling of variable content for interactive paper applications based on a relatively small number of printed copies. On the other hand, existing VDP systems mainly focus on the *efficient* and *fast* publishing of hundreds or thousands copies of documents containing variable content. In order to speed up the printing process, different solutions have been proposed, including the one used by the Component Object Graphic (COG) extractor for VDP [6]. In this case, static content that is repeated on several pages might be extracted from different types of paginated documents. For documents with highly redundant content, this approach might effectively reduce the amount of data to be processed by the printer, resulting in a faster printing process. Even if the variable content mechanism increases the flexibility of our infrastructure, any resources defined within a document will be instantiated for every page at print time, which again increases the amount of data to be processed by the printer. To solve this problem, we could extend our Anoto-based publishing engine to support the COG approach on PostScript documents for static, repeating content.

Even though the large-scale publishing of interactive paper documents is not yet a major application domain, we see a lot of potential for introducing interactive paper documents into existing publishing channels such as for education and news. The approach described in this paper represents a necessary step in this direction and could open up a range of new opportunities for paper-digital systems in different domains. However, we can imagine that as soon as interactive paper technologies become more widely adopted, it might also make sense to support the mass production of interactive paper documents. For this reason, standards such as PPML or PPML-T[5] should be taken into consideration. Due to the data-

[4] Personalized Print Markup Language 2.2 (PPML), http://www.podi.org

[5] Personalized Print Markup Language: Templating 1.0 (PPML-T), http://www.podi.org

driven nature of our solution and the introduction of the template mechanism, supporting such formats would definitely make sense in the further development of our authoring and publishing infrastructure.

6 Conclusions

We have presented a novel approach for supporting the development of paper-digital systems and highlighted the advantages of an authoring rather than programming approach for developing interactive paper documents and applications. We have shown how templates and variable content simplify the authoring process and introduce some interesting issues not present in current digital publishing systems. Last but not least, we have provided an overview of a visual authoring tool that can be used to design, assemble and re-assemble applications and described how this supports a clear separation of roles at authoring time. While the active component developer is in charge of implementing the application logic, the graphical template designer defines pieces of artwork which can later be reused at the page or document level. In a similar way, the developer of the interaction templates defines active page areas together with the corresponding linked services. Last but not least, the end user of our paper-digital authoring tool can define their own active page areas and link them to existing active components or reuse some of the graphical and interaction design templates.

The presented advanced visual authoring tool for paper-digital systems simplifies the development and maintenance of digital pen and paper-based applications. While the presented solution has been used to realise some of our more recent paper-digital applications, in the future we plan to do a more detailed evaluation of the innovative functionally offered by our paper-digital authoring tool. Thereby, we will also investigate the gain in flexibility by using an authoring rather than programming approach when developing paper-digital systems. The presented solution has to be seen as a first important step towards advanced interactive paper publishing solutions and some of the introduced features might be adapted by other paper-digital authoring systems.

References

1. Anoto SDK for PC Applications, 2006. Version 3.3.
2. Form Design Tool User Guide, February 2006. Version 2.1.
3. Paper SDK Specification and Description, February 2006.
4. Anoto AB. *Pen API programmer's guide (Java)*, 2006.
5. Apache XML Graphics Commons, http://xmlgraphics.apache.org/commons/.
6. Steven R. Bagley, David F. Brailsford, and James A. Ollis. Extracting Reusable Document Components for Variable Data Printing. In *Proceedings of the ACM Symposium on Document Engineering (DocEng 2007)*, pages 44–52, Winnipeg, Canada, November 2007.
7. Kevin M. Conroy, Dave Levin, and François 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. Fabio Giannetti and Royston Selmann. Anvil: VDP Segmented Workflow Toolset. Technical Report HPL-2007-18, HP Labs, 2007.
9. François Guimbretière. Paper Augmented Digital Documents. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST 2003)*, pages 51–60, Vancouver, Canada, November 2003.
10. Felix Heinrichs, Jürgen Steimle, Daniel Schreiber, and Max Mühlhäuser. Letras: An Architecture and Framework for Ubiquitous Pen-And-Paper Interaction. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2010)*, pages 193–198, Berlin, Germany, June 2010.

11. Hisashi Ikeda, Kosuke Konishi, and Naohiro Furukawa. iJITinOffice: Desktop Environment Enabling Integration of Paper and Electronic Documents. In *Poster Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST 2006)*, pages 94–95, Montreux, Switzerland, October 2006.
12. Scott R. Klemmer, Jamey Graham, Gregory J. Wolff, and James A. Landay. Books with Voices: Paper Transcripts as a Tangible Interface to Oral Histories. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2003)*, pages 89–96, Fort Lauderdale, USA, April 2003.
13. Chunyuan Liao, François Guimbretière, Ken Hinckley, and Jim Hollan. PapierCraft: A Gesture-Based Command System for Interactive Paper. *ACM Transactions on Computer Human Interaction*, 14(4):1–27, January 2008.
14. Min Lin, Wayne G. Lutters, and Tina S. Kim. Understanding the Micronote Lifecycle: Improving Mobile Support for Informal Note Taking. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*, pages 687–694, Vienna, Austria, April 2004.
15. Paul Luff, Guy Adams, Wolfgang Bock, Adam Drazin, David Frohlich, Christian Heath, Peter Herdman, Heather King, Nadja Linketscher, Rachel Murphy, Moira Norrie, Abigail Sellen, Beat Signer, Ella Tallyn, and Emil Zeller. Augmented Paper: Developing Relationships Between Digital Content and Paper. pages 275–297, 2007.
16. John Lumley, Roger Gimson, and Owen Rees. A Framework for Structure Layout & Function in Documents. In *Proceedings of the ACM Symposium on Document Engineering (DocEng 2005)*, pages 32–41, Bristol, United Kingdom, November 2005.
17. Wendy E. Mackay. Is Paper Safer? The Role of Paper Flight Strips in Air Traffic Control. *ACM Transactions on Computer-Human Interaction*, 6:311–340, 2000.
18. D. McDowell. Variable Data Printing What? Where? Standards! *IPA Bulletin*, 2005.
19. Moira C. Norrie and Beat Signer. Information Server for Highly-Connected Cross-Media Publishing. *Information Systems*, 30(7):526–542, November 2005.
20. Moira C. Norrie, Beat Signer, Michael Grossniklaus, Rudi Belotti, Corsin Decurtins, and Nadir Weibel. Context-Aware Platform for Mobile Data Management. *ACM/Baltzer Journal on Wireless Networks (WINET)*, 13(6):855–870, October 2007.
21. Moira C. Norrie, Beat Signer, and Nadir Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proceedings of the 1st International Workshop on Collaborating over Paper and Digital Documents (CoPADD 2006)*, pages 9–12, Banff, Canada, November 2006.
22. Moira C. Norrie, Beat Signer, and Nadir Weibel. Print-n-Link: Weaving the Paper Web. In *Proceedings of the ACM Symposium on Document Engineering (DocEng 2006)*, pages 34–43, Amsterdam, The Netherlands, October 2006.
23. Peter Robinson, Dan Sheppard, Richard Watts, Robert Harding, and Steve Lay. Paper Interfaces to the World-Wide Web. In *Proceedings of the World Conference on the WWW, Internet & Intranet (WebNet 1997)*, Toronto, Canada, November 1997.
24. Abigail J. Sellen and Richard H. R. Harper. *The Myth of the Paperless Office*. MIT Press, November 2001.
25. Beat Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. Books on Demand GmbH, May 2008.
26. Beat Signer, Ueli Kurmann, and Moira C. Norrie. iGesture: A General Gesture Recognition Framework. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR 2007)*, pages 954–958, Curitiba, Brazil, September 2007.
27. Beat Signer and Moira C. Norrie. As We May Link: A General Metamodel for Hypermedia Systems. In *Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007)*, pages 359–374, Auckland, New Zealand, November 2007.
28. Beat Signer and Moira C. Norrie. PaperPoint: A Paper-Based Presentation and Interactive Paper Prototyping Tool. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI 2007)*, pages 57–64, Baton Rouge, USA, February 2007.
29. Beat Signer and Moira C. Norrie. A Framework for Developing Pervasive Cross-Media Applications based on Physical Hypermedia and Active Components. In *Proceedings of the 3rd International Conference on Pervasive Computing and Applications (ICPCA 2008)*, pages 564–569, Alexandria, Egypt, October 2008.
30. Beat Signer and Moira C. Norrie. Interactive Paper: Past, Present and Future. In *Proceedings of 1st International Workshop on Paper Computing (PaperComp 2010)*, Copenhagen, Denmark, September 2010.
31. Beat Signer and Moira C. Norrie. A Model and Architecture for Open Cross-Media Annotation and Link Services. *Information Systems*, 36(6):538–550, May 2011.
32. Beat Signer, Moira C. Norrie, Michael Grossniklaus, Rudi Belotti, Corsin Decurtins, and Nadir Weibel. Paper-Based Mobile Access to Databases. In *Demo Proceedings of ACM International Conference on Management of Data (SIGMOD 2006)*, pages 763–765, Chicago, USA, June 2006.

33. Jürgen Steimle, Oliver Brdiczka, and Max Mühlhäuser. CoScribe: Integrating Paper and Digital Documents for Collaborative Knowledge Work. *IEEE Transactions on Learning Technologies*, 2(3):174–188, 2009.
34. Aurélien Tabard, Wendy E Mackay, and Evelyn Eastmond. From Individual to Collaborative: The Evolution of Prism, a Hybrid Laboratory Notebook. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2008)*, pages 569–578, Chicago, USA, 2008.
35. Theophanis Tsandilas and Wendy E. Mackay. Knotty Gestures: Subtle Traces to Support Interactive Use of Paper. In *Proceedings of the International Conference on Advanced Visual Interfaces (AVI 2010)*, pages 147–154, Roma, Italy, May 2010.
36. Nadir Weibel. *A Publishing Infrastructure for Interactive Paper Documents: Supporting Interactions across the Paper-Digital Divide*. PhD thesis, ETH Zurich, 2009. Dissertation ETH No. 18514.
37. Nadir Weibel, Adriana Ispas, Beat Signer, and Moira C. Norrie. PaperProof: A Paper-Digital Proof-Editing System. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2008) (Interactivity Track)*, pages 2349–2354, Florence, Italy, April 2008.
38. Nadir Weibel, Moira C. Norrie, and Beat Signer. A Model for Mapping between Printed and Digital Document Instances. In *Proceedings of ACM Symposium on Document Engineering (DocEng 2007)*, pages 19–28, Winnipeg, Canada, August 2007.
39. Pierre Wellner. Interacting With Paper on the DigitalDesk. *Communications of the ACM*, 36(7):87–96, July 1993.
40. Ron Yeh, Scott Klemmer, Andreas Paepcke, Marcello Bastéa-Forte, Joel Brandt, and Jonas Boli. Iterative Design of a Paper + Digital Toolkit: Supporting Designing, Developing, and Debugging. Technical Report 2007-10, Stanford Computer Science, March 2007.
41. Ron B. Yeh, Chunyuan Liao, Scott R. Klemmer, François Guimbretière, Brian Lee, Boyko Kakaradov, Jeannie Stamberger, and Andreas Paepcke. ButterflyNet: A Mobile Capture and Access System for Field Biology Research. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2006)*, pages 571–580, Montréal, Canada, 2006.
42. Ron B. Yeh, Andreas Paepcke, and Scott R. Klemmer. Iterative Design and Evaluation of an Event Architecture for Pen and Paper Interfaces. In *Proceedings of the 21th Annual ACM Symposium on User Interface Software and Technology (UIST 2008)*, pages 111–120, Monterey, USA, October 2008.