Web Application for Analysis, Manipulation and Generation of Accessible PDF Documents

Alireza Darvishy, Hans-Peter Hutter, and Oliver Mannhart

ZHAW Zurich University of Applied Sciences
InIT Institute of Applied Information Technology
P.O. Box 805
CH-8401 Winterthur, Switzerland
{alireza.darvishy,mano}@zhaw.ch

Abstract. This paper presents a software architecture for a web-based service that checks the accessibility of PDF documents and is capable of rendering them accessible. Users will be able to detect accessibility issues related to a PDF document and use a web browser to fix them. The implementation includes a user interface component as well as a PDF analysis and tagging engine (work in progress). The user interface component should be intuitive, in order to support users who do not have extensive knowledge regarding the accessibility of PDF documents. No additional software and installation is needed to use the system, the software should be available for use in educational and e-government applications.

Keywords: Accessibility, Document accessibility, Accessibility Analysis and manipulation of PDFs, visual impairment, tagged PDF, software architecture, user interface design, web content accessibility guidelines.

1 Introduction

The PDF standard is used in a variety of fields and preferred by many authors and authorities. Many PDF documents are available on internet platforms, unfortunately most of them are not accessible for users with disabilities. All major word processors (such as Microsoft Word [1], OpenOffice [2], LaTeX [3], etc.) support a way to export a document to a PDF file. Yet most of these methods do not produce tagged PDFs (which are required for accessible PDFs), or the resulting PDFs are tagged incompletely (not compliant with WCAG 2.0 [10]) and have to be corrected manually. Obviously this is a tedious work and requires specialist know-how. Consequently, the majority of the publishing community is unable to cope with this task. Two key-problems must therefore be addressed:

1. The lack of accessibility knowledge on the part of authors.
2. Different word processors are used for the generation of PDFs.

This project aims to develop a tool, which can be used with very limited knowledge of accessibility and does not require any installation on the local computer.

C. Stephanidis (Ed.): Universal Access in HCI, Part IV, HCII 2011, LNCS 6768, pp. 121–128, 2011.

2 Our Solution

One approach to solve these problems is to extend the functionality of the word processors [9]. However, this is a complex and time-consuming task since every tool has to be adapted.

Our solution therefore works directly on the PDFs themselves, since all major word processors are able to generate PDF documents. Our system is capable of checking PDFs as well as the correctness and completeness of false or missing accessibility tags. The following diagram shows a simplified outline of the architecture used for the web application.

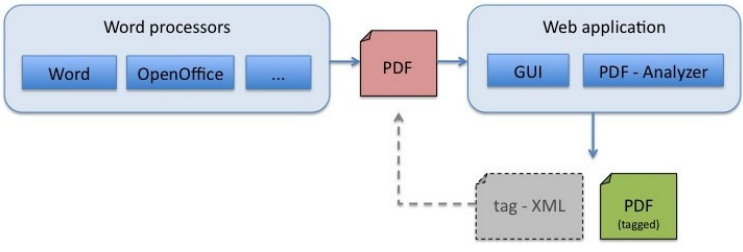

Fig. 1. Architecture of web application

As shown in Fig. 1, the application's output contains a fully accessible PDF and an optional file that contains tag information in XML format. XML is useful in several situations. For example, a student might use the web application to create an accessible PDF, and discover shortly thereafter spelling mistakes in the document. Thanks to XML, he or she does not have to reenter all of the accessibility information again. Instead, he or she can directly generate a new, fully accessible PDF. The XML file can also be used with a document that has a similar structure, a different number of images, or even new tables compared to the original PDF. The application will detect such discrepancies and ask only for the additional information it needs (e.g. the description for a new image).

Based on its concept and architecture, the web application liberates authors from secondary accessibility tools (e.g. accessibility checkers) and from the need for specific accessibility and tagging knowledge. Authors neither need to know how to tag PDFs nor what exactly has to be tagged. All this know-how is held by the application and is presented through a dynamic web page (GUI).

3 Problem Domain

Because a PDF does not necessarily have to be enriched with structural information, it can be very difficult to assemble this information retrospectively (a table could comprise only positioned text). It is therefore crucial to integrate the user, as only the user can reliably provide information on the logical structure of the PDF.

The system reads the PDF in two different ways. Firstly, it extracts existing structural information (tags), and secondly it proposes potential tags according to

given rules. It can thus identify text blocks as paragraphs, for example. Such potential tags are presented to the user as errors because the system deems these tags to be missing. Of course, original tags with missing attributes are also displayed to the user as errors. In addition to marking and defining structural information, the reading order can also be derived from the tag tree of the PDF. A logical representation of a document (in this context, a PDF) has therefore been defined for the web application. It should be noted that although this representation does have similarities with the structure of a tagged PDF from ISO 32000-1 [12], it is not the same but only an abstraction of it. When a PDF is processed by the web application, however, all "standard structure types" defined in ISO 32000-1 must be mapped accordingly. Fig. 2 shows this logical representation as an example. It should be noted that neither types nor their attributes are shown in full detail. The graphic serves simply to illustrate the concept. The reason behind defining a logical representation of a document is that we want to separate the web application from the PDF standard. As a result, changes or extensions to the PDF standard lead only to reimplementations in the PDF parser.

3.1 Representation of a Document

A document comprises various elements. Derived from these basic elements are various types (section, heading, paragraph, etc.). Certain elements (e.g. section) can contain further elements. This results in a structure tree of the document. If it is processed depth-first, the result is the order in which the elements appear in the document.

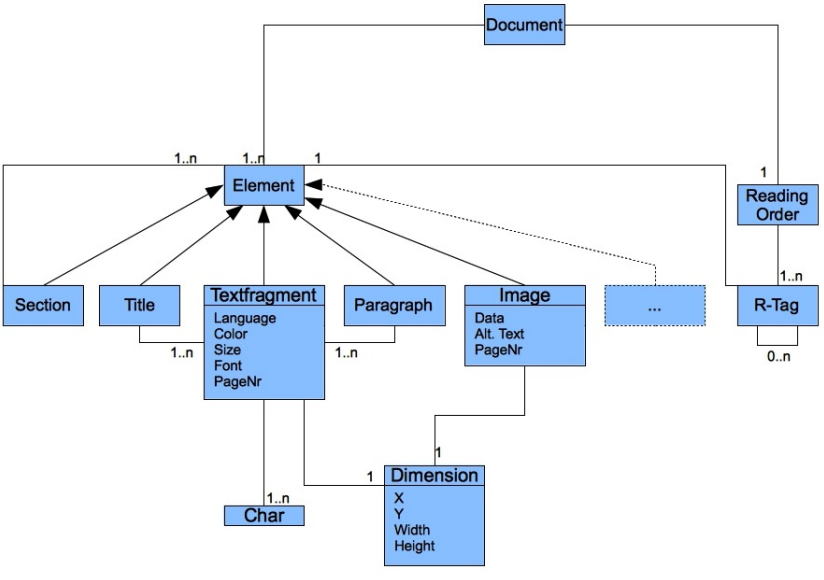

Fig. 2. Logical Structure

The important thing is that the page and position of each element can be determined. For a heading it can be determined via its text fragments. In the case of an image, on the other hand, the information pertaining to page and position can be determined directly (by means of dimension). These attributes are used primarily for the graphic presentation in the GUI.

In addition to the elements, a document also has a reading order which can differ from the absolute positioning of an element in the document. The reading order is also structured as a tree, whereby each R-tag of this reading order (see Fig. 2) references one element. Consequently, the reading order can be arranged freely without having to make any changes to the elements. If a tag does not reference an element, nothing is marked during synchronization with the PDF. The same occurs if an element without a counterpart is created in the reading order.

In order for the system to have this functionality, it has to be able to export the tags and errors from the PDF. To do so, it outputs the Reading Order tag tree, if available. For every tag, there must be a corresponding element in the PDF. So, an element tree can also be constructed (users can view incomplete elements displayed as errors in the GUI). Additionally, the system tries to detect elements not yet included in the tag tree. These are then added to the element tree (these tags are displayed as errors in the GUI because the system considers the elements still missing). Users can now modify or expand the tag or element tree via the GUI (e.g. correcting incomplete elements). Finally, the tag tree as well as the element tree is written back to the PDF.

4 User Interface Concept

This section explains the GUI concept in detail. Only the main components of the GUI will be examined in detail here and underlying ideas will be highlighted.

The user's main task is to correct errors (e.g. inserting an alternative text), ensure a logical reading order, and adjust, insert or delete existing tags. In the GUI presented here, these functions are shown within one view. Thus, the work to be done is immediately apparent. Additionally, discrepancies (incorrect reading orders, for example) are easy to recognize, as the GUI intuitively highlights whether or not the PDF is coherent.

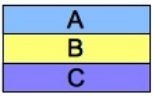

Fig. 3. Main Layout

The main GUI is divided into three sections (A, B, C), which will be discussed here in detail. These three sections are arranged on top of each other, as shown in Fig. 3.

4.1 A – Status Section

This section of the main GUI provides information on the status of the PDF. Thus, it is evident here whether or not it was possible to check the PDF and how many errors

were detected. The user gets an overview of the PDF's status (from the accessibility point of view).

4.2 B – Unvisualized Events

A PDF not only displays errors or tags that concern a certain section of the PDF. The main language of the document cannot, for example, be assigned to a specific part of the PDF. Such errors therefore cannot be visually marked in the PDF and are thus itemized in a list (see Fig. 4).

Fig. 4. Unvisualized events

It should be noted that such errors or tags will not be translated to the PDF as tags. Most of them are PDF meta information. However, this is irrelevant for the user and introducing further terms (such as meta information) would only confuse the user.

Each line in the table shown in Fig. 4 comprises four cells. In addition to the type of error or tag (e.g. main language), the description, and the status, the user is given a number of options for dealing with tags or errors in the PDF. An error (see lines B1) can be corrected (option C in Fig. 4) or deleted (option D in Fig. 4). If an error is corrected, it does not disappear from the table, rather its status is changed and it becomes a full tag (see B2). It is important to leave corrected errors in the table so that the users can clearly see the results of their actions (e.g. correct an error).

If this logic is followed, it makes complete sense to also show tags in the same table. Unlike errors, however, tags can be edited not corrected.

The delete option, however, applies to both errors and tags. At first glance, it might appear counterproductive to delete an error. However, it must be assumed that the web application falsely identified the PDF structures (e.g. a table instead of simple text). In these cases, the user must be able to delete the incorrect error message.

4.3 C – PDF Visualization

Section C visualizes the PDF and the elements important in relation to accessibility. Section C primarily comprises three columns (see Fig. 5): The reading order on the left, the page thumbnails, and the actual tags and errors in the PDF on the right. Across these three columns, the correspondences of the "tagged PDF" are evident and can be intuitively manipulated. In the case of PDF documents with many pages, the PDF visualization is split over several pages. The user can then toggles between the individual pages or skip straight to the next error via the corresponding links.

Visualization: Reading Order. All tags/errors are listed in the reading order. Additionally, each element in the reading order displays a reference number that

appears again both in the thumbnails and also in the tag/error visualization. This enables the user to quickly find the origin of a tag.

Certain tags (the section tag in Fig. 5, for example) can contain further tags. Such tags are displayed in the reading order with two tags. One of the tags marks the start and the other marks the end of the tagged section. These special tags, however, are used only for approximate structural markings (part, section, article, etc.). A table, by contrast, is displayed as a tag although strictly speaking it comprises several tags. The reading order within a table is then processed by means of sub-dialogs that have a structure identical to the GUI in Fig. 5.

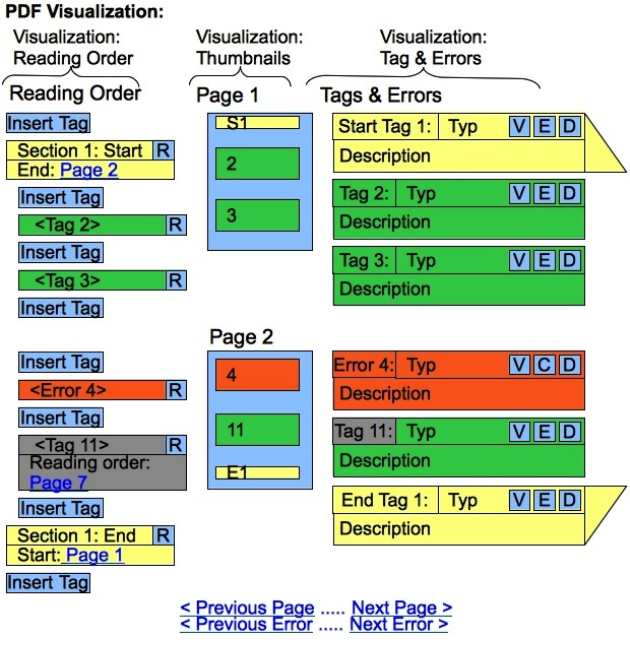

Fig. 5. PDF Visualization

The user is also to be given the option of inserting new tags into the reading order (marked in Fig. 5 with "Insert Tag") or changing the reading order (marked in Fig. 5 with "R").

However, the reading order can differ from the actual sequence of document parts. This aspect is also considered in the GUI calculation. Tag 11 in Fig. 5 depicts such a case. This tag is not supposed to appear in the reading order until much later (e.g. on page 7). Such a tag is shown twice in the GUI . Once to denote the absolute positioning of the tag with a reference to the page in the reading order (page 7 in this example). It has a distinctly different color to normal tags. The tag would also appear in the reading order on page 7 (see Fig. 6). Of course, the tag can now no longer be visualized in the thumbnails because it is actually located on page 2. Therefore, the user is again given a reference to the actual tag page (page 2 in our example).

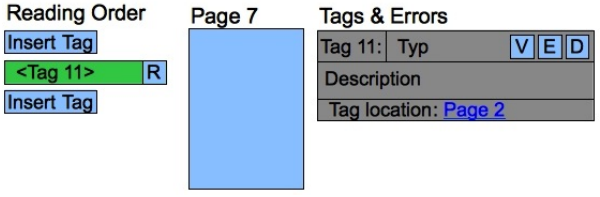

Fig. 6. Reading Order

Visualization: Thumbnails. The thumbnails are the key element of section C. They allow the user to quickly find things in the document and adjust the reading order and tags accordingly.

A thumbnail is a miniature view of a PDF page. One thumbnail is therefore created for each page. In the thumbnails (see Fig. 5), the individual tags or errors are highlighted in a different color. An error (e.g. image without alternative text) appears in red. A tag (e.g. a heading), on the other hand, is shown in green. In addition, each color marking is assigned a unique tag number that also appears in the reading order.

Visualization: Tag & Errors. The third and last part of section C contains the tags and errors that were found in the PDF. It should be noted that the errors are displayed only when the PDF has been checked. The GUI described in this section, however, can also be used to show tags only. As clearly shown in Fig. 5, the visualization of tags & errors corresponds to a detailed view of the PDF sections marked in the thumbnails. Besides various information on the tags or errors, it is particularly important to highlight the functions accessible to the user via this view. In Fig. 5, these are symbolized by the letters V, E, D, and C. V (View) is used to select a sub-dialog that displays the full view of the corresponding page and the tag (without thumbnail). E (Edit) can be used to edit a tag (e.g. change the type or an attribute). Other dialogs are of course selected, depending on the tag. However, listing them all would exceed the scope of this work. D (Delete) deletes the corresponding tag or error. C (Correct) allows an error to be corrected and then converted into a correct tag. The Tag & Error view also contains the nested structural tags (e.g. section tag 1).

5 Conclusion

The software architecture presented in this paper is suited for a broad audience (with no specific know-how required) and could therefore make a difference in the quality and quantity of fully accessible documents. Based on the proposed architecture, a prototype application is currently being implemented. The main advantage of the proposed concept is that it is independent from the authoring software (such as e.g. MS Office [8]), which results in lower maintenance and support costs.

References

1. Microsoft Office Word, http://office.microsoft.com/word/
2. OpenOffice, http://www.openoffice.org/
3. LaTeX, http://www.latex-project.org/
4. PDF Accesibility Checker, http://www.access-for-all.ch
5. Freedom Scientific JAWS for Windows Screen Reading Software, http://www.freedomscientific.com/products/fs/jaws-product-page.asp
6. NetCentric PDF Accessibility Wizard for MS Office, http://www.net-centric.com/products/PAW.aspx
7. Virtual508.com Accessible Wizard for Microsoft Office, http://www.virtual508.com
8. Microsoft Office (2010), http://us1.office2010beta.microsoft.com
9. Darvishy, A., Hutter, H.-P., Horvath, A., Dorigo, M.: Dorigo: A Flexible Software Architecture Concept for the Creation of Accessible PDF Documents. In: Miesenberger, K., Klaus, J., Zagler, W., Karshmer, A. (eds.) ICCHP 2010. LNCS, vol. 6179, pp. 47–52. Springer, Heidelberg (2010)
10. WCAG 2.0, http://www.w3.org/TR/WCAG20/
11. eGov PDF Checker, http://accessibility.egovmon.no/en/pdfcheck/
12. ISO 32000-1:2008. First Edition 2008-7-1: Document management - Portable document format - Part 1: PDF 1.7, pp. 573-610, http://www.iso.org/