DocHandles: Linking Document Fragments in Messaging Apps

Laurent Denoue, Scott Carter, Jennifer Marlow, Matthew Cooper
FX Palo Alto Laboratory
Palo Alto, CA
{denoue,carter,marlow,cooper}@fxpal.com

ABSTRACT

In this paper, we describe DocHandles, a novel system that allows users to link to specific document parts in their chat applications. As users type a message, they can invoke the tool by referring to a specific part of a document, e.g., "@fig1 needs revision". By combining text parsing and document layout analysis, DocHandles can find and present all the figures "1" inside previously shared documents, allowing users to explicitly link to the relevant "document handle". Documents become first-class citizens inside the conversation stream where users can seamlessly integrate documents in their text-centric messaging application.

CCS CONCEPTS

• **Document preparation Multi/Mixed Media Creation • Computer Graphics • Image Manipulation • Image Processing**

KEYWORDS

Document capture; image processing; enterprise messaging; interactive documents

1 INTRODUCTION

Over the last few years, messaging applications such as Slack have been adopted by millions of users. With these new communication tools, users frequently share documents they are working on, either as links for example to Google Docs, or by directly attaching document files.

After a file is linked to or attached, users start discussing its contents: They give each other feedback about what needs to be changed, or refer to a specific table or figure that others should pay attention to, for example a link to a PDF paper that is relevant to their research. Unfortunately, making sense of what is referred to can be cognitively expensive, and hinders the normal flow of messages. Misunderstanding can be further exacerbated for teams distributed across time zones who frequently read messages hours after they have been posted.

Figure 1: When the user types "@fig2", she is shown a list of figures extracted from recently shared documents; each suggestion corresponds to the "figure 2" in that document, along with the document filename or title and caption.

This work is inspired by behaviors that we observed in the chat logs of teams of knowledge workers using Slack to communicate and co-edit a variety of documents. We found that small groups often make edit documents such as research papers in Microsoft Word, or presentations in PowerPoint, and used natural language to guide their collaborators to specific parts of a document.

For example, we noted that people often added text comments after sharing links, documents or images. Comments included specific parts of the documents (e.g., "introduction section") or mentioned aspects of their structure (e.g., "slide 1"). People also spent a fair amount of time telling others where files are, what has been uploaded, commented on, or edited. Finally, different versions of the same file were uploaded with text describing what was done. People used numbered lists or references to page, figure, and slide numbers to clarify what they were referring to, especially if they occurred in between other messages.

While typing comments about the document into Slack may have been easy, there remained a disconnect between the discussion in Slack and the relevant parts of the document that needed attention. In other words, collaborators had to switch between the Slack conversation and the document itself in another

DocEng '17, September 4-7, 2017, Valletta, Malta

ACM ISBN 978-1-4503-4689-4/17/09…$15.00
https://doi.org/10.1145/3103010.3121036

program, or move the conversation into a document application with integrated chat.

Previous systems have attempted to augment chat spaces with recommended content [1] without directly linking to documents or their fragments. Other systems augment web clients with chat capabilities, but require manual selection of the appropriate fragment of the currently rendered page [2].

To better integrate chat-based conversations with documents, we describe the design and implementation of DocHandles where users can explicitly link specific parts of documents they are referencing. When a user types "@" in her message, the system matches the subsequent characters to determine the relevant document fragment. For example, in Figure 1, the user typed "@fig2" and the system immediately shows her a list of possible figures corresponding to "figure 2" in recently shared documents. The user simply clicks on a suggested figure to link it to her message. With this tool, recipients immediately understand more of the message intent and document context. They can see the figure by hovering over the link or access it within the original document page by clicking on it.

We describe below how the system analyses documents shared in the chat application, and recommends document fragments based on the user's message.

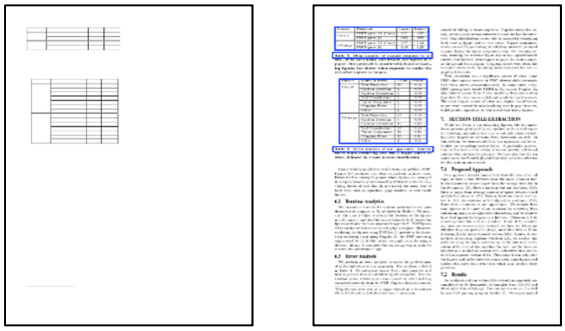

Figure 2. Left: the binary image without text rendered; right: the page with detected captions and tables.

2 SEMANTIC LAYOUT ANALYSIS

To link "@fig2" typed by a user to the actual figure 2 of a document, we developed a new document layout engine that finds images in documents and associates each with its nearby caption (if any). With the caption, the system matches the figure/table number with the message content.

To deal with various file formats used in enterprise chat messaging applications, documents are first converted to PDF. LibreOffice[1] is used to convert Microsoft Office documents such as Word and PowerPoint. Links to Google Docs are also quite common, so we export them as PDF by modifying the link to obtain their PDF versions (see [4]).

2.1 DOCUMENT IMAGE PROCESSING

Similar to the processing flow of PDFFigures2 [3], our document layout engine renders each document page image without text elements via a modified version of XPDF. A binary image is then obtained by thresholding the grayscale page image (we use 168 out of 255 gray values) and the bounding boxes of connected components are found. For each page, we also use XPDF to extract each word and its bounding box on that page.

For business documents seen in our Slack teams, this binary image clearly shows areas of the page where images, graphs or tables are present, see Figure 2. We now explain how the text and connected components of binary images are used to form the final document layout. Extracting graphical content from document images is an active area of research [3,6,7].

2.1 FINDING CAPTIONS

Word bounding boxes are grouped into text lines, and the following regular expression is used to find likely captions:

```
/^(graph|chart|figure|table|tab\.?|algorithm|alg\.?|fig\.?)\s*\d+\s*(\.|:|–)/
```

Each caption bounding box is retained and used later to associate it with its corresponding image. Figure 2, right, shows that the captions for each table were found. Note that for our purposes, we do not need to detect the complete caption, because the first line alone links users' messages to potential captions. Also, the first line is an informative cue for users in the interface showing suggestions (see Section 4).

2.1 DETECTION OF FIGURES

The boxes of connected components are recursively merged if they are close to each other, and are not separated by the bounding box of a caption, as found in the previous step. We use 10 pixels on document images rendered at 72 DPI to decide whether to group two boxes.

Graphs or flowcharts often contain text elements near graphical elements. Since the binary image was obtained without rendering text, the next step grows each region box to include nearby words, using the bounding boxes of text elements found previously.

2.2 FINDING TABLES

The previous step fails to detect tables that might contain horizontal lines that were too far from each other to be grouped. We further group those if they are vertically aligned, i.e., if their left and rightmost positions are equal within a 2-pixel threshold to accommodate binarization errors. Tables without rules could be detected using dedicated methods such as [6,7].

[1] https://www.libreoffice.org/

2.3 FINETUNING WITH CAPTIONS

For many images, the previous step fails to group distinct elements that appear together in one figure. For example, a figure might span the entire width of a document page, but the individual components are not all within the 10-pixel threshold used to group them in step 2.1. To address this issue, we observed that caption text usually spans the whole width of the corresponding figure. We thus use this box width as a heuristic to group figure elements that are above this caption box. The result is a better grouping of the actual figure elements that constitute one figure, as shown in Figure 3.

2.4 LINKING IMAGES TO CAPTIONS

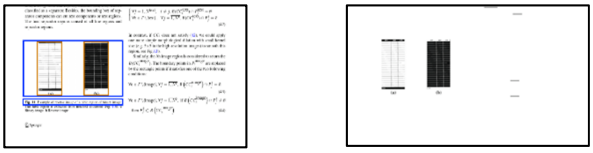

Figure 3. Left: the binary image without text rendered; right: the page with detected caption and figure. Note that the initial boxes (orange) were merged to span the width of the nearby caption box.

After finding the bounding boxes of detected images, graphs and tables, the system then associates the captions with their corresponding figures. We use Euclidian distance to determine which caption should be linked to a given image.

The result of this step is a JSON file that contains the bounding boxes of likely figure areas along with their captions. This structure is used in the next step to match a user's message to specific content in documents. To detect whether a file is likely a presentation file (e.g., PowerPoint), we simply use the aspect ratio of the page dimensions. These results are filtered to generate the candidate lists for users to interactively review to create links.

3 TEXT MESSAGE PARSING AND MATCHING OF RELEVANT DOCUMENTS

To inform our implementation, we analyzed the message logs of three Slack teams over a period of four months and extracted the most frequent words people use when referring to document fragments in their messages, namely "page", "slide", "figure", "table" and "algorithm". Other words such as "paragraph" or "introduction" are also common, but we decided to focus our implementation efforts on the aforementioned keywords for now. People also often cite passages of documents in their messages, e.g., "please change XXX into YYY". We have not yet implemented this detection.

Users often abbreviate keywords, e.g., "fig" for "figure", "tab" instead of "table", "p" for "page" and "alg" for "algorithm". These variations are captured by two regular expressions that match the root of the keyword, followed by a number. For figures, tables and algorithms, we use the following expression:

```
/^(figure|table|tab\.?|algorithm|fig\.?)\s*(\d+)\s*(\.?|:?|-?)/gi
```

To detect mentions of a page or slide number, we use this second expression:

```
/^(page|slide|p|pg)\.?\s*(\d+)/gi
```

Given a user's message prefix (the text typed after the "@" sign), these two expressions thus yield null, or a couple {keyword, number}. If a match is found, the JSON file generated in the document analysis step (see Section 2) allows us to find the corresponding page, slide, or figure in each document that was shared in the conversation. If a slide keyword is detected, only documents classified as presentations are kept.

The list of matching document page regions is sorted by most recent document first, because our analysis of the Slack logs indicates that users are more likely to refer to a document when it was most recently shared in the conversation. We include only documents shared within the current channel.

4 INTEGRATION IN CHAT APPLICATIONS

To test our algorithms, we integrated them into our custom enterprise chat application called DocuChat and instantiated them as a Slack "bot".

4.1 INTEGRATION IN DOCUCHAT

We developed DocuChat, a custom web-based chat application like Slack, to quickly test new techniques for interacting with documents inside chat applications. Our web-based interface gives full access to users' actions, especially in our case keyboard events, providing in turn a more integrated user experience. All screenshots in this paper come from DocuChat.

In DocuChat, users can quickly pick or drag and drop documents from their desktop onto the web application. Similarly, if they link to a known document type in their message, DocuChat automatically follows the link, downloads the document, converts it to a PDF and previews it below the message. Clicking on the thumbnail shows users the document inside the browser window, avoiding unnecessary downloads and application switching. We use Mozilla's PDFJS to render PDF files inside the web browser. The layout analysis is performed on the server on each document, yielding a JSON file that contains all words as well as image positions and associated captions and stored in a database.

When the user types a "@", the system looks for characters that follow and tries to match this suffix against a caption, e.g., "fig1" will find the couple {"figure","1"} as explained in section 3.

The list of all figures appears in an overlay window, along with the document title and associated caption if found, as shown in Figure 1. Since chat applications are text-centric, we allow users to TAB or use arrow keys to pick a suggestion. They can also click with their mouse. Upon picking a suggestion, the system creates a hyperlink that encodes the document identifier along with the page number and image number on that page. For example, "@fig2" typed by the user is converted into "@fig2#390-4-5" if the figure was the 5th image on page 4 of document 390.

Figure 4. When a user clicks the link, the system shows them the document

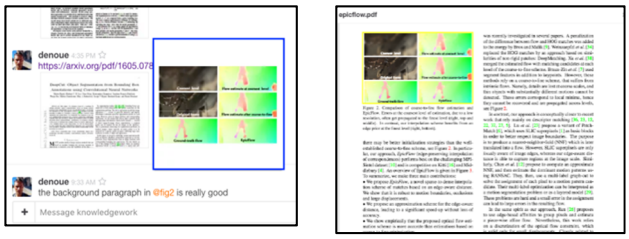

Figure 5. Left: clicking the DocHandle link in the chat message, shows the original document automatically scrolled to the area of interest (here "figure 2").

Because such hyperlinks would clutter the text box and perhaps even confuse users, the caption of the hyperlink only shows the original user's content, here "@fig2" only, as shown in Figure 4. When the message is posted, any user can hover over the link to view the corresponding region of interest (ROI) in the document image, as shown in Figure 5 left, inside the blue border.

Finally, if a user clicks on the hyperlink, they are shown the original document, scrolled to that exact location, see Figure 5 right. The interface supports a seamless transition between text-centric chat messages and the specific document-centric context of the conversations.

4.2 INTEGRATION IN SLACK

Slack's popularity stems in part from its rich set of APIs, allowing third-party services to write "bots". Once a bot is invited to a channel, it can receive all posted messages, including attachments. We wrote DocuBot, a Slack bot that users can invite to their channel. DocuBot indexes all attached documents posted on the channel, and looks for links to documents in messages. Like in DocuChat, each document is indexed by our document layout analysis system.

If a message is found to match a caption, the bot retrieves a list of suggestions (as in the case of DocuChat) and posts them as a Smart Slack Message that includes image thumbnails of the relevant document fragments. Only the user who typed that message sees this first message. She can ignore it, but she can also click on a button associated with one of the images to pick a suggestion. Upon receipt, the bot then posts the selected figure to the channel so that all users can see the explicit link between the previous message and the document fragment. Unlike DocuChat, this Slack bot does not have fine control over the previous message, but the integration is nonetheless important because DocHandles can be deployed and tested by many more teams that already use Slack.

5 CONCLUSIONS

We described DocHandles, a new technique that enriches text-based chat conversations about documents by allowing users to quickly link specific and relevant portions of documents inside their messages. Anecdotal evidence with a few users from our laboratory and friends shows that users like to create these DocHandles links because they effectively augment their messages while preserving the ease of text-centric communication in this medium. Additionally, users like to read messages that contain explicit mentions of document fragments because they reduce ambiguity and link conversations to relevant content.

Future work will focus on extending the layout analysis and matching functions presented in this paper. In particular, PowerPoint slides typically lack numbering of their figures. The algorithm will thus need to classify each detected image area as a type, e.g., table, figure, algorithm, graph, pie-chart. One method would be to use neural networks to classify detected images as shown in [5].

Finally, analysis of Slack message logs has shown that users copy verbatim some passages in documents. We can readily use the text extracted from the PDF documents to match those. But users will not necessarily type "@" before the text excerpt and the user interaction will need to change: the system could for example monitor the current message's contents and pro-actively insert an icon or hyperlink where it thinks that content relates to a document fragment, letting users quickly validate with a button press or TAB.

REFERENCES

[1] S. Loh, D. Lichtnow, A. Justin C. Kampff, J. de Oliveira. Recommendation of Complementary Material during Chat Discussions. Knowledge Management & E-Learning: An International Journal (KM&EL), Vol 2, No 4 (2010)

[2] Kifi. https://techcrunch.com/2016/07/12/google-acquires-deep-search-engine-kifi-to-enhance-its-spaces-group-

[3] chat-app/
C. Clark and S. Divvala.. PDFFigures 2.0: Mining Figures from Research Papers. Proc. of ACM/IEEE-CS on Joint Conf. on Digital Libraries. ACM, 2016.

[4] Labnol https://www.labnol.org/internet/direct-links-for-google-drive/28356./

[5] Y. Rangoni, A. Belaid. Document Logical Structure Analysis Based on Perceptive Cycles. IAPR Workshop on Document Analysis Systems. Springer, 3872, , 2006.

[6] M Fan, DS Kim. Detecting Table Region in PDF Documents Using Distant Supervision. 2015, arXiv:1506.08891.

[7] L. Hao, L. Gao, X. Yi and Z. Tang, "A Table Detection Method for PDF Documents Based on Convolutional Neural Networks," IAPR Workshop on Document Analysis Systems (DAS), 2016