Optical Character Recognition for Apache PDFBox

Short Description

Apache PDFBox is a widely used library to extract text from PDF files, manipulate existing PDF files and create new PDF files. The current functionality of text extraction is done by fetching data stream of the particular PDF file and passing through several filters and algorithms of PFDBox. This method does not work in situations where PDF file has malformed character encoding and text embedded in images. PDFBox currently have tools to create images from a given input PDF file. In this GSoC project, a new approach is proposed to extract text from a PDF file through Optical Character Recognition by using the image generated using PDF box.

Proposal Title : Optical Character Recognition for Apache PDFBox
Student Name: Dimuthu Upeksha
Student Email : dimuthu.upeksha2@gmail.com
JIRA Issue : https://issues.apache.org/jira/browse/PDFBOX-1912

Deliverables

- This improves the acceptance of PDFBox among community because there are lots of PDF files that has scanned images which can not be extracted using normal text extracting libraries. Specially Governmental documents

- Accuracy of the text extraction can be improved.

- Provide the flexibility to users to choose best suiting method (OCR or normal method) or both in their applications.

- Even documents with wrong character encoding can be used to extract its text using OCR algorithms because OCR doesn't depend on encoding of the characters but the shape of characters.

Detailed Description

Apache PDFBox currently can extract text by analyzing it's COS model. A PDF consists of a COS model which includes different variable types like Boolean, Number, String and etc. Current text extraction algorithm go through this model and fetch text + location data. Coming up with an accurate text extraction is very complex to automate because model of PDF files are defined not in machine readable manner. It's structured in human readable way. So it needs a lot of processing to do a correct text extraction considering placement of each word and character of the PDF file. What PDFBox does is mapping whose word + location data into meaningful output text. But this method does not work in some use cases.

1. When PDF file contains text in image format
2. When PDF file contains corrupted character encoding.

Problem in both scenarios is, we can't get text + location data from the COS model of the PDF file. Although we can get location data in second scenario, we can't get actual words from it because its character encoding is corrupted. So there should be another approach to feed those details to PDFBox. That's where Optical Character Recognition is used. In Optical Character Recognition input is an image and it returns word + location data after processing pixel details of the image. Tesseract is a such OCR library that was written in C++ and currently managed by google. The input image is generated by PDFBox's utilities using input PDF file and pass it to the OCR library.

Scope of the project

- Provide ability to PDFBox to invoke third party OCR libraries to fetch word + location data of PDF files which belongs to above use cases.

- The coupling between the third party library and PDFBox should be minimal in order to make ability to plug another library with minimal changes in the code

- Currently Tesseract OCR library is one of the best open source OCR libraries available. But it is written in C++. So it is required to implement a java wrapper using jni to provide support to java to invoke native methods.

- Implement a consumer at PDFBox side which uses above wrapper to map fetched word + location data into meaningful data. Just like what PFDToTextStripper is doing.

Data flow

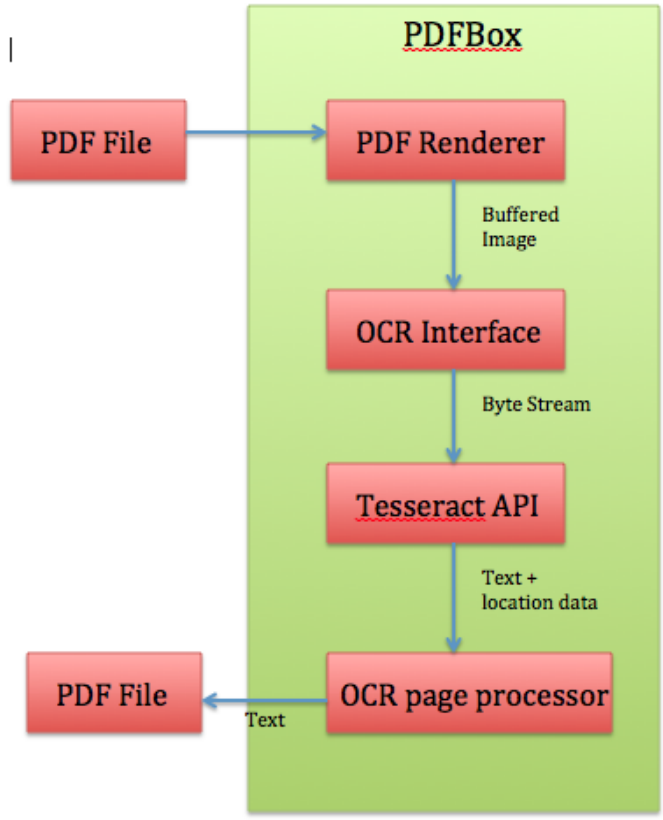
1. PDF file is loaded into PDFBox as a PDDocument object

2. It is parsed into a Bufferedimage object using PDF Parser classes

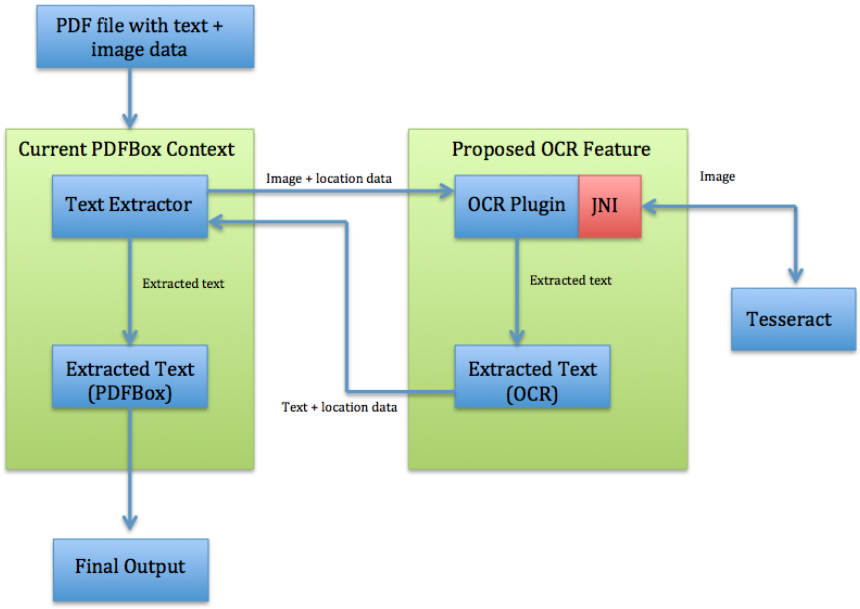
3. Created BufferedImage object is fetched by a generic OCR interface and converted into a byte stream which consists 24 bits per pixel RGB data.

4. Byte stream with its metadata is passed to Tesseract api (Wrapper for native Tesseract api)
5. Tesseract api passes recognised word + location data to OCR page processor to organize that raw data into meaningful strings.

6. Final text is given as output



Top view architecture



Considering the top view architecture there are three main parts

1. Jni wrapper for Tesseract c++ library

Tesseract is an open source C++ library for OCR. It has several command line tools and in built C++ api to communicate with other applications. In this project, the C++ api is used. http://zdenop.github.io/tesseract-doc/group___advanced_a_p_i.html. To consume that api there should be a java wrapper in order to call them in java. There is a JNI wrapper for Android and we can port its architecture to this project.
https://code.google.com/p/tesseract-android-tools/source/browse/tesseract-android-tools/jni/com_googlecode_tesseract_android/
Although the api provides lots of operations we need only few of them.

a .

```
void   tesseract::TessBaseAPI::SetImage (const unsigned char *imagedata, int width, int
       height, int bytes_per_pixel, int bytes_per_line)
```

b.

```
char *  tesseract::TessBaseAPI::GetUTF8Text ()
```

c.

```
ResultIterator
*                tesseract::TessBaseAPI::GetIterator ()
```

d.

```
void   tesseract::TessBaseAPI::End ()
```

Currently, the implementation of this wrapper has been finished.
https://github.com/DImuthuUpe/Tesseract-API

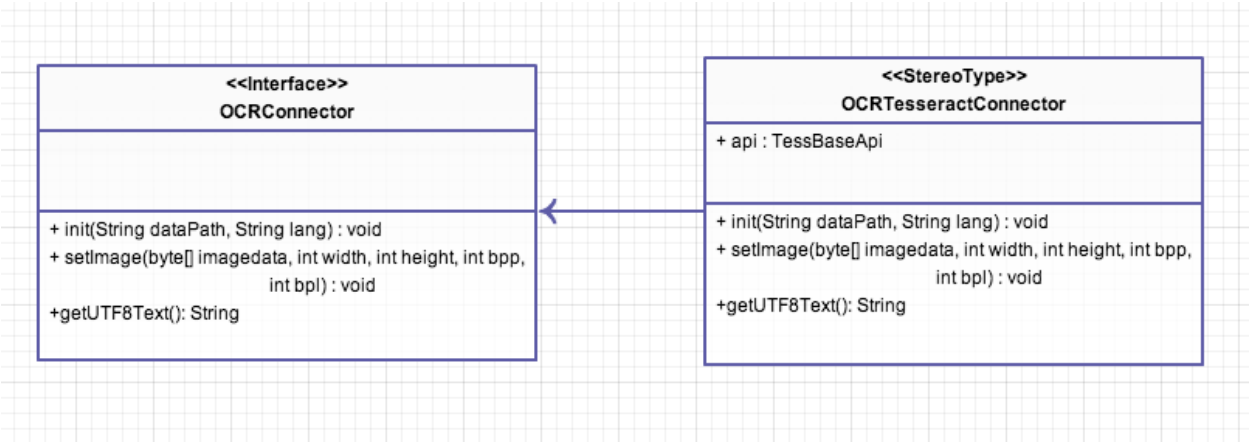
Tesseract uses a image processing library called Leptonica. So it is required to build leptonica and link to Tesseract before Tesseract is built. JNI wrapper uses the static libraries of Tesseract and Leptonica. Those libraries are platform dependent because they access system libraries of the operating system. So it is a must to map correct compatible libraries to jni wrapper before building the project. This can be achieved by two ways.

a. Use pre built libraries
b. Build them from the source

Project has both features (pre built libraries and cmake files to build them from the source). So it's not hard for users to configure the project.

2. Plugin to connect Tesseract wrapper to PDFBox

This is the gateway to connect any OCR library to PDFBox. It includes a simple interface named "OCRconnector" which contains basic operations that are needed in a OCR library. PDFBox doesn't know the underlying technology at OCR side. It simply uses this OCRConnector to get required data. To connect Tesseract wrapper we need to create a sub class of OCRconnector interface and call Tesseract wrapper's code inside it. This allows us to easily intergrate another OCR library into PDFBox easily.



3. OCR Text Processor at PDFBox.

This is simply like PDFTextStripper which currently available in PDFBox. From OCR libraries what we get is recognised words with their location data(bounding box's pixel locations). We need something like processing engine to use those raw data and generate meaningful text. So this needs to implement some algorithms to effectively join recognised data into final output.

Implementation Approach

Phase 1 : Developing JNI wrapper for Tesseract

- Writing makefiles to automate Leptonica build
- Writing makefiles to automate Tesseract bulid
- Writing makefiles to build native api with linking Tesseract and Leptonica
- Automate above makefiles using maven ant plugin
- Generating and placing platform dependant static libraries of above libraries at maven build time
- Writing unit tests

Phase 2 : Developing plugin interface

- Coming up with a consistent interface that can cover all requirements of a OCR library
- Writing Tesseract plugin for PDFBox
- Writing integration tests

Phase 3 : Developing OCR Text Processor at PDFBox

- Going through current algorithms used in PDFTextStripper and try to re-use suitable algorithms
- Coming up with a model to figure out special features of the PDF. Ex : Columns

Phase 4 : Finalizing

- Acceptance Testing
- Bug fixing

Time Frame

Time Period	Expected outcome
April 21 - May 19	Discussing with the mentors about further details of the project and setting up the development environment
May 19 - June 2	Phase 1 Fixing possible memory overflow situations Finishing JNI wrapper Tests

June 2 - June 16	Phase 2 Developing plugin interface
June 16 - July 21	Phase 3 Developing OCR Text Processor at PDFBox
July 21 - August 11	Phase 4 :Finalizing

Remarks :
1. Phase 1 is already finished. There are small bug fixings and test implementations to be done.

2. Need more time to phase 3 because it consumes a lot of time to R&D stuff. So I hope to finish phase 1 as soon as possible and find more time to phase 3.

Additional information

Bio

Email : dimuthu.upeksha2@gmail.com
Mobile : +94719084638
Skype : dimuthu_upeksha
Blog : http://dimuthuupeksha.blogspot.com/
Linkedin : http://www.linkedin.com/profile/view?id=196355296

I'm DImuthu Upeksha, a 3rd year undergraduate at Department of Computer Science and Engineering - University of Moratuwa Sri Lanka. I have hands on experience of open source projects like Moodle, Docbook, Apache ISIS, Apache Synapse, Apache Axis2, WSO2 ESB, WSO2 Application Server and WSO2 App Factory. Although I'm a computer engineering student I would like to contribute with other engineering activities also ,specially in robotics. I'm a sharp learner and interested to learn new technologies. Currently I'm doing my internship at WSO2 which is a leading company in open source development.

Contribution to open source development

- I have successfully completed my GSoC project last year with Apache software foundation. I worked on Apache ISIS. The project was building a generic Android application which consumes RESR API of ISIS.

  Blog :
  http://google-opensource.blogspot.com/2013/10/google-summer-of-code-veteran-orgs.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+GoogleOpenSourceBlog+%28Google+Open+Source+Blog%29

Project : https://github.com/DImuthuUpe/ISIS_Android_Viewer

- I have actively participated in writing TCK tests for ISIS restful object viewer

  Issue : https://issues.apache.org/jira/browse/ISIS-421

- I have involved in moodle code development and provided several bug fixes in some versions of moodle code

  http://tracker.moodle.org/browse/MDL-27731

  http://tracker.moodle.org/browse/MDL-30144

  http://tracker.moodle.org/browse/MDL-30157