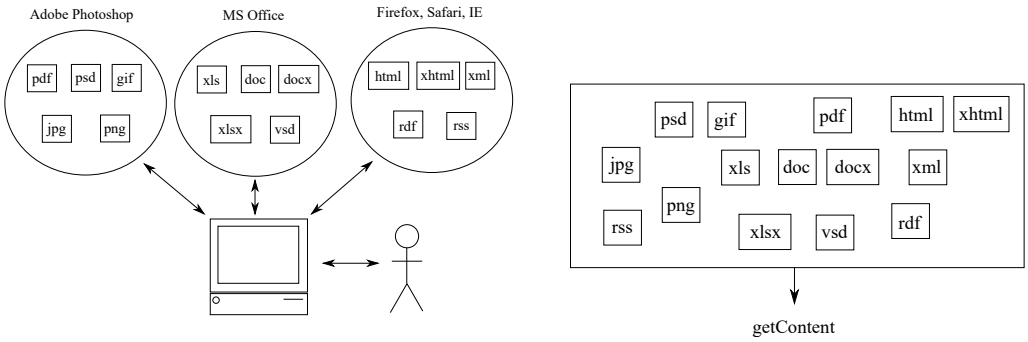
Lab 2 – Data Extraction: Apache Tika

Deadline: +1 week

1 Motivation

- Most of applications process only dedicated file formats.
- Programs need converters to read unsupported file formats (e.g. Microsoft Office – Office Open XML format and OpenOffice OpenDocument format).



2 Purpose

- An application that processes multiple commonly used file formats.
- Search engine processing documents stored on a shared drive: Excel spreadsheets, Word documents, text files, PDFs, images, audio files, ZIP archives, etc.

3 File types

- About 51 000 different file types.
- How to recognize a file format? By file extension (.xls, .html) – often depends on the system and installed applications.

- MIME – Multipurpose Internet Mail Extension – to help applications "deal with the data in an appropriate manner".
- Identifier: type/subtype and optional parameters (attribute = value).
- More than 1000 of official types (text/plain, text/html, image/jpeg) and thousands of unofficial ones.

4 Metadata

- "data about data".
- Dublin Corestandard – 15 attributes (data format, title, creator, etc.).
- Multiple dedicated metadata formats, e.g. Adobe – Extensible Metadata Platform (XMP).

5 Apache Tika

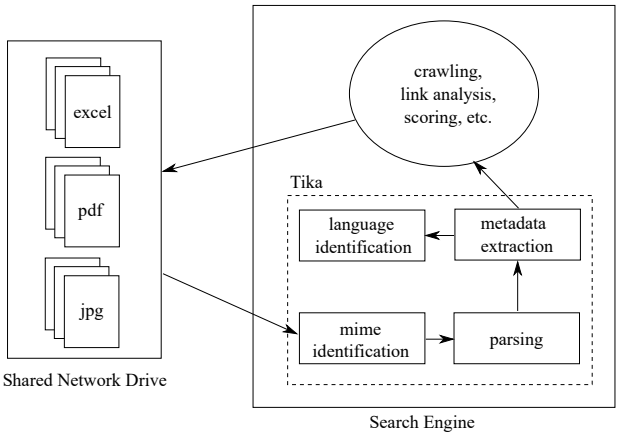
Toolkit for detecting and extracting metadata and text content from over a thousand different file formats (such as DOC, PPT, XLS and PDF; http://tika.apache.org).

5.1 General purposes

- Unified parsing: a set of functions and Java interfaces for parsing files.
- Parses integration: easy access to libraries parsing different file formats.
- Low memory requirements.
- Fast processing and file type detection.
- MIME detection.
- Understanding multiple metadata formats.
- Language detection.

5.2 General applications

- Tika is widely used for content indexing.
- Document content analysis – finding key entities (people, places, etc.) and relations between them (with Apache UIMA and Mahout).
- Digital Asset Management.



5.3 Document processing

Stages of document processing:

1. File type identification.
2. Parser selection (e.g. PDFBox for PDF files).
3. Text content and metadata extraction.
4. Language identification.

6 Programming assignment

Useful imports are provided at the end of this pdf file

6.1 Exercise 1

The purpose of this exercise is to compare the existing libraries for obtaining content from different file formats. Write a program for retrieving phone numbers from files stored in an archive **exercise1.zip** (**do not** unpack the file; it contains one PDF file and one XML file). Use the following files provided in the exercise bundle: **Exercise1.zip**, **pdfbox-2.0.8.jar**, **tika-app-1.17.jar**, and **Exercise1.java**. Create a java project, using any IDE (IntelliJ IDEA, NetBeans, etc.) and add the downloaded files.

Exercise 1.a

1. Your task is to finish the **exercise1a** method for getting the phone numbers from the files stored in the archive. You can use the **ZipFile** and the **ZipEntry** classes to get the files from the archive. See the below code:

```
ZipFile file = new ZipFile(''Exercise1.zip'');
Enumeration entries =  file.entries();
while(entries.hasMoreElements())
{
ZipEntry entry = (ZipEntry) entries.nextElement();
InputStream stream = file.getInputStream(entry);
}
```

2. Check the file type by reading the file name extension (use **entry.getName()**) and choose a correct method for content extraction:

   - For the PDF file, use the **PDFBox** library. Use a static method **PDDocument.load()** and a method **getText()** of **PDFStripper** object to get the text content. It is suggested to use regular expressions to derive the phone numbers from the text. See the below piece of code:

     ```
     Pattern pattern = Pattern.compile(\\([0-9]{3}\\) ?[0-9-]+);
     Matcher matcher = pattern.matcher(content);
     while (matcher.find())
         String text = matcher.group();
     ```

   - For the XML file, use Java build-in classes:

     ```
     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
     DocumentBuilder db = dbf.newDocumentBuilder();
     Document document = db.parse(stream);
     ```

     Then, use a **getElementsByTagName()** method of the **Document** object to get all nodes having the specified tag name. **Use tag = "Phone"**.

3. The **exercise1a** method has to return a list of retrieved phone numbers (**LinkedList** <**String**>).

4. Run the **Exercise1.java** class. Some statistics should be printed and you may see that some of the phone numbers were not found by the method. Why?

Exercise 1.b Now, your task is to complete the **exercise1b** method for retrieving the phone numbers. This time, however, use Apache Tika.

1. Firstly, create an **AutoDetectParser** object. It, as the name suggests, automatically detects the file type (e.g., zip) and uses a proper method for data extraction.

2. Then, create a **Metadata** object (a multi-valued data container).

3. You can use a **PhoneExtractingContentHandler** object to derive the phone numbers when parsing the files. Use a **parse** method of the **AutoDetectParser** object.

4. Lastly, use a **getValues** method of the **Metadata** object to get a list of values associated with a provided metadata name. Use **name = "phonenumbers"**.

5. Uncomment the line (print results) in the main method. Run the program and analyze the results. Did Tika extract more phone numbers stored in XML file than Java build-in class? Why Tika failed (missed) to extract some of the phone numbers? This is a tricky question.

6.2 Exercise 2

Given is a a collection of documents stored in **documents.zip** file. This collection contains documents being written in different languages and saved in different file formats. Your task is to use Tika to extract metadata and text content from these files. Furthermore, use Tika to identify the language each file was written in.

Start with a scratch of code provided in **Exercise2.java** file. For each file in the collection create a new text file with the same name (but with extension ".txt") with information about the language, creator (author), creation and modification date, MIME type and the text content of this file. Metadata should be saved in separate lines, followed by a blank line and the text content. See the below example:

```
Exemplary output for file test.docx (available in documents.zip):
Name: test.docx
Language: fr
Creator:
Creation date: 2018-01-30
Last modification: 2018-01-30
MIME type: application/vnd.openxmlformats-officedocument.wordprocessingml.document

test file content
```

6.3 Useful (and sufficient) imports

```
importorg.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.text.PDFTextStripper;
import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.sax.BodyContentHandler;
import org.apache.tika.sax.PhoneExtractingContentHandler;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.util.Arrays;
import java.util.Enumeration;
import java.util.LinkedList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.zip.ZipEntry;
import java.util.zip.ZipFile;
```