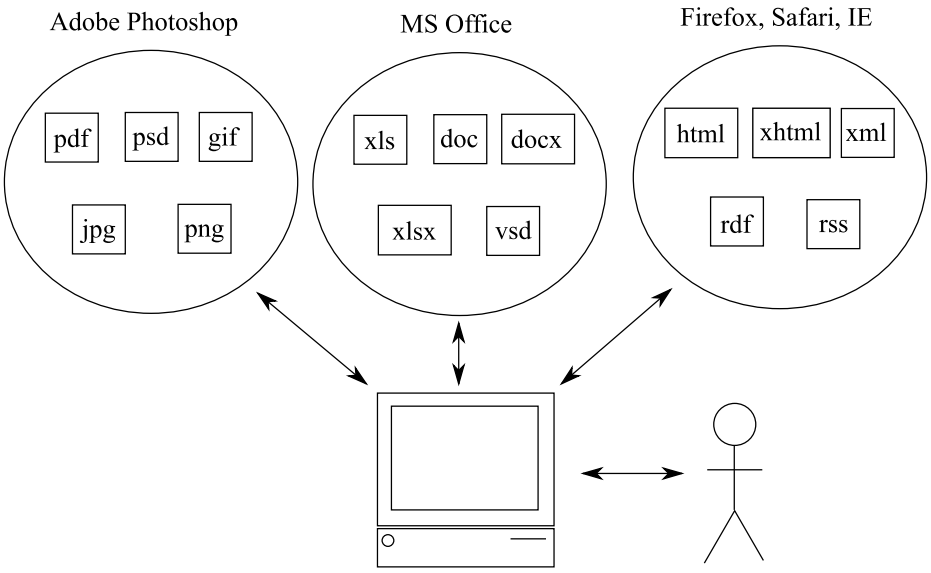
LAB 2: Apache Tika

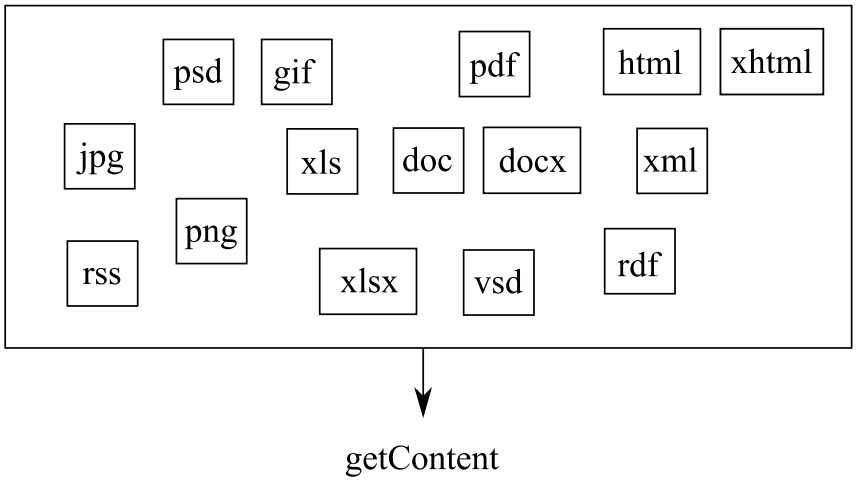
1. Motivation:

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



2. Purpose:

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



3. File types:

- About 51 000 different file types**.**
- Different formats, encoding, methods of keeping the content**.**
- How to recognize used file format? By file extension (.xls, .html) – often depending 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 types.

4. Metadata:

- "data about data"**.**
- *Dublin Core* standard – 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 from over a thousand different file types (such as DOC, PPT, XLS and PDF; http://tika.apache.org):

- Apache Tika provides methods for automatic file type detection.
- Tika provides support for the most popular metadata formats and tools for defining own ones.

General purposes:

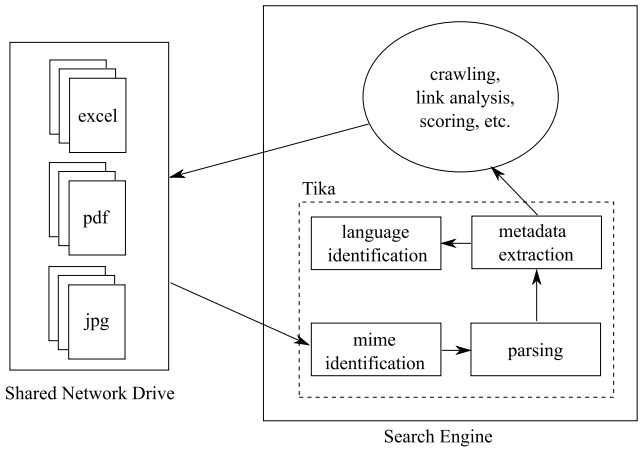
- Unified parsing: set of functions and Java interfaces, which allow using multiple libraries (e.g. org.apache.tika.Parser).
- Parses integration: easy access to libraries parsing different file formats.
- Low memory requirements.
- Fast processing and file type detection.
- MIME detection.
- Understanding multiple metadata formats.
- Language detection.

General applications:

- Indexing text content from documents in multiple formats.
- Document content analysis – finding key entities (people, places, etc.) and relations between then (with **Apache UIMA** and **Mahout**).
- Digital Asset Management.

Apache Tika is accessible as an application with GUI and command-line interface.

Document processing by Apache Tika:



Stages of document processing:

1. File type identification.
2. Choosing the appropriate parser (e.g. *PDFBox* for PDF files).
3. Text content and metadata extraction.
4. Language identification.

Programming assignment (Java, deadline +1 week)

Exercise 1: Compare the existing libraries for obtaining content from different files.

(1) Write a program for getting phone numbers from files in archive **exercise1.zip (without unpacking the file; it contains one PDF file and one XML file)**. Download the following files from http://www.cs.put.poznan.pl/mtomczyk/ir/lab2/: Exercise1.zip, jar files pdfbox-2.0.8.jar, tika-app-1.17.jar, and Exercise1.java. Create java project using any IDE for JAVA (IntelliJ IDEA, NetBeans, etc.) and add the downloaded files.

Exercise 1a

(2) Your task is to implement **exercise1a (Exercise 1)** method for getting the phone numbers from the files in the archive. You can use Java classes ZipFile and ZipEntry to get the files from the Exercise1.zip. 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);
}
```

(3) Detect input file type by its extension (entry.getName()) and choose correct method to parse the content**:**

a. For PDF file use PDFBox library:

i. Use static method PDDocument.load() and method getText() of PDFStripper class to get text content.

ii. It is suggested to use regular expressions to get 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();
```

b. For XML file use Java build-in classes to parse XML:

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

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

(4) The method exercise1a has to return a list of derived phone numbers (LinkedList<String>). Don't forget to update this list every time your method finds a phone number.

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

Exercise 1b

(6) Now, your task is to complete the method **exercise2b (Exercise 1).** It should do the same what **exercise1a** does, but this time you have to use Apache Tika to derive phone numbers.

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

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

(9) You can use **PhoneExtractingContentHandler** to derive the phone numbers when parsing the files. Use parse method of **AutoDetectParser**.

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

(11) Uncomment the line (print results) in the main method. Run the program and analyze the results.

a. Did Tika extract more phone numbers of the XML file (contained in the zip file) than used Java build-in class?

b. **Why Tika failed (missed) to extract some of the phone numbers?** This is a tricky question.

Exercise 2: Given is a collection of documents. This collection contains documents written in different languages and saved using different file formats. Your task is to use Tika to extract metadata of these files and their text content. Furthermore, use Tika for language identification.

For each file of the collection create a new text file with the same name (but with extension ".txt") with information about language, creator (author), creation and modification date, MIME type and 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

(12) Download template file **Exercice2.java** and add it to your project.

(13) Download the collection **documents.zip** and extract it to the same directory (or to the root directory of your Java project).

(14) **Exercice2.java** loads files of the collection (**main()**), parses the files (**processFile()**), and saves the results (**saveResult()**). However, the implementation is incomplete.

(15) Your task is to finish **initLangDetector()** (initializes language detector) and **processFile()** methods in order to extract text content, metadata, and language from a given file. Some useful classes: **AutoDetectParser, BodyContentHandler, Metadata, OptimaizeLangDetector, LanguageResult, TikeCoreProperties.**

```
import org.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;
```