Deliverable

Grant Agreement Number	270137
Full Project Title	Scalable Preservation Environments
Project Acronym	**SCAPE**
Title of Deliverable	Characterisation technology, Release 1 + release report
Deliverable Number	D9.1
Work-package	WP9, PC.WP.1 Characterisation Components
Dissemination Level	**PU** = Public
Deliverable Nature	**O** = Other
Contractual Delivery Date	2012-03-31
Actual Delivery Date	2012-03-31
Author(s)	Markus Radtisch (ONB), Peter May (BL), Asger Askov Blekinge (SB), Per Møldrup-Dalum (SB)

Abstract

This report sums up the work done during the first year of the SCAPE project in the Characterisation Components work package of the Preservation Components sub projects. The report documents three commonly used tools from the Digital Preservation Community: DROID, Apache Tika™, and FIDO. The tools have been evaluated against the GovDoc1 document corpora and the enclosed ground truth. The report also documents the amount of work that has been carried out to extend especially two of the selected tools, namely Apache Tika™ and DROID. Finally a road map for the next year is laid out. No conclusions regarding the quality of the tools are presented in this report.

Keyword list

digital preservation, characterisation, identification, signature files, PRONOM

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).


SCAPE

Authors

Person	Role	Partner	Contribution
Per Møldrup-Dalum	Deliverable Manager	SB	chapter 1, 2, and 6
Peter May	author	BL	chapter 5
Markus Raditsch	author	ONB	chapter 3
Asger Askov Blekinge	author	SB	chapter 4
Alan Akbik	author	TUB	text-mining issues of chapter 6, section 5.1.4

Document Approval

Person	Role	Partner
Dave Tarrant	Reviewer	OPF

Distribution

Person	Role	Partner
Project Office		SCAPE

Revision History

Version	Status	Author	Date	Changes
0.1		Per Møldrup-Dalum	2012-01-02	Initial draft/outlining
0.2		Per Møldrup-Dalum	2012-01-26	draft of chapter 1,2, and 6

0.3		Per Møldrup-Dalum	2012-02-13	added drafts of chapters 3 and 5. Harmonise layout
0.4		Per Møldrup-Dalum	2012-02-22	added drafts of chapters 4
0.5		draft completed	2012-02-29	made everything fit together
0.6		Per Møldrup-Dalum	2012-03-26	incorporated comments and suggestions from reviewer
1.0	Final	Per Møldrup-Dalum	2012-03-30	ready for publication

References

see section 8

Executive Summary

This report describes the work done during the first year in the Characterisation Components work package in the Preservation Components sub project of SCAPE.

The work package initially compiled a list of 16 tools for identification and charaterisation (van der Knijff, Askov Blekinge og Schlarb, WP 9: target characterisation tools 2011) . That list was reduced to five tools. These five tool were evaluated against the characterisation evaluation framework (van der Knijff, Askov Blekinge og Schlarb, Characterisation Evaluation Framework 2011). The result of that evaluation can be seen in (van der Knijff og Wilson, Evaluation of Characterisation Tools 2011).

To optimise our efforts we decided to focus on only file identification using a still smaller selection of tools. We narrowed the list of tools to DROID[1] and FIDO[2]. During the weeks leading up too this decision, the Apache Tika™[3] tool entered our discussions and we realised that it should have been part of our lists. As it quickly showed great promise for our work, we decided to inlcude it on our short list. These three tools are all documented in this report with regards to usage and deployment..

To evaluate the tools a new evaluation framework was implemented. The evaluation described in Evaluation of Characterisation Tools (van der Knijff og Wilson, Evaluation of Characterisation Tools 2011) used a private data set and it was required manual testing of the tools. To remidy these shortcomings, we created a extended testing framework. This framework evaluates the tools using the Govdocs1 digital Corpora and the characterisation results from Forensic Innovations, Inc. as ground truth. The Govdocs1 corpora was chosen as it is large, cirka a million files, and it is freely available for alle partners in the SCAPE project.

In this report we defer from presenting any conclusions based on the described evaluations. Instead we will present a partial conclusion on this work as part of the SCAPE-CP067 check point.

In parallel with the evaluations we examined the possibillities of extending DROID and Apache Tika™. We examined the feasibility of adding DROID functionality to Apache Tika™, added support for Apple's iBooks 2 format to Apache Tika™, started work on the language module of Apache Tike™, refactored the DROID code base into a Maven project and did some experiments with a Hadoop integration of DROID.

In the second year of the SCAPE project (Feb. 2012 to Feb. 2013), we will move from simple identification of digital objects to complex characterisation and expand upon the set of selected tools by considering more complex tools such as FITS and JHOVE2. At the same time, we will also look at more simplistic tools such as UNIX file, specialised tools as ffprobe (use for characterisation of multimedia files) etc., embedding, extending or learning from them as appropriate. We will also begin to use text-mining technologies for analysis of different kinds of text corpora e.g. web content.We will use these technologies for creating syntax analysis, trend analysis and complex characterisation of web content, both acquired from archives, harvests, and monitoring of text feeds like Twitter, forums, blogs etc. Most of the results will be fed directly into the Planning and Watch (PW) subproject for the Watch components.

[1] http://sourceforge.net/apps/mediawiki/droid

[2] http://www.openplanetsfoundation.org/software/fido

[3] http://tika.apache.org/

Table of Contents

1 Introduction

1.1 Scope of this Document

This deliverable report is part of SCAPE Work Package 9 (PC.WP.1) named Preservations Components —Characterisation Components. This report builds primarily on the following reports:

"WP 9: evaluation framework for characterisation tools" (van der Knijff, Askov Blekinge and Schlarb, Characterisation Evaluation Framework 2011),

"WP 9: target characterisation tools" (van der Knijff, Askov Blekinge and Schlarb, WP 9: target characterisation tools 2011),

"Evaluation of characterisation tools: Identification" (van der Knijff and Wilson, Evaluation of Characterisation Tools 2011),

"Tika Evaluation 0.1" (Askov Blekinge 2011),

"Experimental "real world ARC" extraction report" (Raditsch 2011)

1.2 Outline

Chapter **Fejl! Henvisningskilden blev ikke fundet.** gives the motivation for the scope of work done during the first year of the SCAPE project and thereby leading up to this report. Chapter 3 documents the three tools, how they can be deployed and details their usage in relation to SCAPE. Chapter 4 goes on to discuss a common evaluation of all three tools, briefly documenting the evaluation framework developed and used during the first year. Chapter 5 describes the development effort carried out on the actual tools. Finally, chapter 7, lays out the ground for the next year - what this work package intends to focus on in SCAPE year two. The actual details for this road map will be published in the checkpoint CP067, "Identification of target tools and tool-types for iteration 02" in month 15.

1.3 Terminology

In order to avoid any confusion regarding terminology, this report uses the same terminology as Evaluation of Characterisation Tools (van der Knijff og Wilson, Evaluation of Characterisation Tools 2011) described as follows:

In practice the term "characterisation" is often used to indicate quite different things. To avoid any confusion, this document follows the terminology used in the JHOVE2 project. Here, characterisation is loosely defined as the process of deriving information about a digital object that describes its character or significant nature[4]. This process is subdivided into four aspects:

[4] JHOVE2 actually uses 2 separate definitions: "(1) Information about a digital object that describes its character or significant nature that can function as an surrogate for the object itself for purposes of much preservation analysis and decision making. (2) The process of deriving this information. This process has four important aspects: identification, feature extraction, validation, and assessment."

- *Identification - the process of determining the presumptive format of a digital object;*
- *Feature extraction - the process of reporting the intrinsic properties of a digital object that are significant to preservation planning and action;*
- *Validation - the process of determining the level of conformance of a digital object to the normative syntactic and semantic rules defined by the authoritative specification of the object's format;*
- *Assessment - the process of determining the level of acceptability of a digital object for a specific purpose based on locally defined policy rules.*

2 Selected Tools

To optimise our efforts we decided to focus on file identification and a small selections of tools. We used the Identification report (van der Knijff and Wilson, Evaluation of Characterisation Tools 2011) as the basis for the tool selection with one addition. During the weeks leading up the compilation of the final tool list, the Apache Tika™[5] tool entered our discussions and we realised that it should have been part of our lists from the beginning. During these discssions it quickly became apparent that this tool showed great promise for our work. We therefore decided to include it on our short list. This led to the following list:

- DROID (Digital Record Object IDentification) [6]
- FIDO[7]
- Apache Tika™[8]

This document focuses exclusively on these three tools, how they perform with regards to the SCAPE project, and what improvements have been made to them during the work leading up to this report.

[5] http://tika.apache.org/

[6] http://sourceforge.net/apps/mediawiki/droid

[7] http://www.openplanetsfoundation.org/software/fido

[8] http://tika.apache.org/

3 Documentation of selected tools

3.1 DROID 6.0

3.1.1 Overview

DROID is a tool that attempts to identify digital objects using PRONOM[9] format signatures ('magic numbers') and/or known file extensions. The identification results are reported as PRONOM-compliant Persistent Unique Identifiers (PUIDs). DROID is an open-source, platform-independent Java application. It can be used directly from the command line, or, alternatively, using a graphical user interface. We evaluated the most recent version of DROID, which is DROID 6.0 (released March 2011).

3.1.2 Tool interface

DROID can be invoked through a graphical user interface and a command line interface. For the command line interface a Windows batch file and a Unix shell script are provided. The batch file only appears to work if it is invoked directly from the application directory (i.e. the directory in which the batch file and the two main JAR files are installed). Since this is not very practical, calling the command-line interface JAR directly will often be more convenient. It is possible to tell DROID to analyse all files in a given directory (or set of directories) using the "-R " command-line switch that activates recursive processing of all subdirectories (and sub-sub directories, and so on).

3.1.3 License type

DROID is released under an open-source BSD License, which allows "[…], inclusion in other products and redistribution as long as the terms of the license are adhered to" (The National Archives n.d.). DROID uses a number of third party components, which are all released under a variety of open source license types (The National Archives n.d.).

3.1.4 Language

DROID is written in Java (version 6). The user documentation states that DROID has been tested on Java 6 update 17 to update 22, while it should run under Java 6 update 10 onwards.

3.1.5 Platform dependencies

According to the user-documentation, DROID runs on Windows, Linux, and Mac.

3.1.6 Coverage of file formats

DROID tries to identify files using file signatures, or, alternatively, known extensions. These are defined in a signature file, which is regularly updated by The National Archives. As an indication, version 45 of this signature file contains 766 file type definitions and 327 file signatures.

[9] http://www.nationalarchives.gov.uk/PRONOM

3.1.7 Extendibility

Since the number of formats that DROID can handle is defined by the information in the signature file, new formats can be added by modifying the signature file (or downloading the latest version from The National Archives). Unlike earlier versions, in DROID 6 the signature file is always stored in a separate and configurable directory, which is not necessarily the location of the DROID application files. In addition to the regular signature file, DROID 6 also uses a separate file with 'container signatures ', which is used for formats that may contain other files within themselves, for example ZIP, which is used as a physical container for the Open Document and Microsoft Office Open XML formats, and OLE2 that is a container for the Microsoft Office formats.

3.1.8 Output format

Older versions of DROID (e.g. DROID 4) used to write output directly to an XML file. In DROID 6, things are slightly more complicated. Central to its input and output handling is the concept of profiles. A profile is defined as "the files and folders you want to find out about, and the results of profiling them" (The National Archives n.d.). A profile uses the ZIP format as a physical container, and all relevant DROID in- and output results are stored in this container. Crucially, the characterisation results are stored in a database format that is not human readable, therefore in order to use the results, the information from the profile has to be exported. DROID 6 can only export the profile information to CSV (comma-separated text) format. By default, each row in the exported CSV file represents one file object. However, if a file object results in multiple format matches, the last four columns of the CSV output (PUID, MIME type, format name and format version) are repeated for each match. This means that the number of columns per row is variable. Alternatively, DROID offers a 'one row per format' option, where each row represents one unique format identification (which has the implication that one file objected may be represented by multiple rows in the CSV file). Exporting is done as a separate step, which means that two separate invocations of DROID are necessary to produce the CSV output (i.e. first scan, then export).

In addition to this, DROID 6 also has options to generate reports that can be in either XML or PDF format, but this report functionality is mainly useful for generating aggregate statistics of the identification results, e.g. the number of files for each format. None of these output formats are particularly suitable for use in automatic workflow systems[10].

3.1.9 Unique output identifiers

Identification results are reported as PRONOM Unique Identifiers (PUID) and MIME types. In addition, DROID also provides a textual description of the format name and, if applicable, the format version number as they are given in the signature file.

[10] Contacted The National Archives about these output format issues, and inquired whether it would be possible to re-introduce the 'old' XML output format. In a reply to this (e-mail Andrew Fetherston, 11-4-2011) TNA confirmed that the XML output was dropped in recent DROID releases. He did however offer to add a suggestion for unified XML output of results for any future development of DROID.

3.1.10 Granularity of output

The use of PUID as the primary identifier ensures that DROID's output can be mapped directly to the PRONOM registry[11] as well as the nascent OPF registry[12].

3.1.11 Deployment and usage

Getting started with the DROID tool requires some minor preparation steps. This section will shortly describe how to deploy and use the DROID tool to create a simple report with DROID.

The DROID tool package can be downloaded from: http://www.nationalarchives.gov.uk

After downloading the ZIP package, which contains JARs, batch scripts, and some other files, create a tool folder of your choice and extract the ZIP package to it. On a windows machine that could be, for example, c:\programs\droid601\.

The deployment of the tool is more or less done with this step.

3.1.11.1 GUI mode

As described above, DROID can be used in GUI and command line mode. A detailed description on how to work with the tool can be found in the documentation included in the tool ZIP package.

GUI mode is a good starting point to get to know the tool. The general workflow and output options are more or less the same for GUI and the command line usage, so you can use the GUI mode to get a feeling for how the tool works and apply your knowledge later if you prefer to use the command line version.

Simply start the scripts (droid.bat or droid.sh) inside the DROID tool folder to start the GUI version.

3.1.11.2 Command Line Interface

Command line mode is your choice if you are planning DROID to run automated and/or combined with other tools.

DROID follows a two-step approach when identifying a set of files. First add a set of files or folders to a DROID "profile", then, in the second step, start analysing the files and create your report file.

The following command line recursively adds the content of the "myFolderToAnalyze" folder to a droid profile temporary stored in droid.tmp:

```
java -jar c:\programs\droid601\droid-command-line-6.0.jar -R -a c:\myFolderToAnalyze -p
%TMP%/droid.tmp
```

[11] http://www.nationalarchives.gov.uk/PRONOM

[12] http://wiki.opf-labs.org/display/TR/Formats

In the second step DROID analyses the content of the droid profile stored in droid.tmp and outputs the result in a CSV file:

```
java -jar c:\programs\droid601\droid-command-line-6.0.jar -p %TMP%/droid.tmp -e
c:\myFiles\myResult.csv
```

Both steps could be easily combined in a script or, for example, in a Taverna external tool plugin if you like to run it within a Taverna workflow. More options and ways to combine them can be found in DROID's documentation.

3.2 FIDO 0.9.6

3.2.1 Overview

Fido (Format Identification for Digital Objects) is an identification tool that uses the PRONOM format signatures. It is essentially a DROID clone. The identification results are reported as PRONOM-compliant Persistent Unique Identifiers (PUIDs). Fido is an open-source command line application written in Python. There is also a Jython-based version of Fido (Jython is a Java implementation of the Python language), which provides Fido as a JAR file. The following two versions will be evaluated in this chapter:

- Fido 0.9.6 (released 4. October 2011)[13]
- Fido_jar 0.9.5 (Jython version, released March 2011)

Note here seems to be no Jython 0.9.6 version of Fido. Since version 0.9.5 is the first (and currently only) Jython release, it was not possible to use the same version of both releases.

3.2.2 Tool interface

Fido can be invoked through a command line interface. Installing Python under Windows will automatically associate ".py" files with the Python interpreter, whereas on Linux-based systems the command-line interpreter can be explicitly declared on the first line of the script. The command line parameters are well documented.

3.2.3 License type

Fido is open-source software that is released under Apache version 2.0 license.

3.2.4 Language

Fido is written in Python. It uses a Python syntax that, as of this writing, is compatible with Python versions 2.6 and 2.7.1. It is not compatible with the Python 3.x range of interpreters. Even though the 2.x range of Python interpreters are still widely used, it may be worthwhile to consider upgrading to Python 3.x–compatible code in order to make it more future-proof. In many cases, it is possible to produce code that works under both Python 2.6 (and 2.7) and Python 3.x. It should be noticed that

13 http://www.openplanetsfoundation.org/blogs/2011-10-04-new-fido-version-096

Python programs can be compiled to native code linking against static C libraries. This last option have not been explored during this work, but would be a candidate for futher investigations.

3.2.5 Platform dependencies

Fido runs on any platform for which Python 2.6 or Python 2.7 is available (it is currently not compatible with the Python 3.x range of interpreters). There are no other dependencies.

3.2.6 Coverage of file formats

Fido tries to identify files using file signatures, or alternatively, known extensions. These are defined in two files, which are based on information from the PRONOM database. So in principle all formats of the PRONOM database are covered, provided that they have a valid signature.

3.2.7 Extendibility

As with DROID, new formats can be added by modifying or updating the signature and extension files. Although Fido's documentation describes how to add new formats to these files, it does not describe how to convert the PRONOM/DROID information to Fido-compatible format. Not having such an option would severely limit Fido's use in any operational setting. In a response to an earlier version of this report, Maurice de Rooij (OPF / NANETH) comments that this conversion can be done with the (undocumented) 'prepare.py' script that is part of Fido. A first test of this script wasn't successful. This issue has not been investigated further.

3.2.8 Output format

Output is written to the standard output device, which can be redirected to a text file. The format is largely configurable using two user-defined command line parameters (matchprintf and nomatchprintf). These parameters are Python string-formatting commands. They define the output in case of a match or no match respectively. The default values of these parameters result in output in comma-delimited format, where each line represents one unique format hit (if one file object gives two format hits, they are both written on separate lines). The default implementation use this output format

Output when the format of a file is matched:

```
OK,time,pronomUID,formatname,signaturename,filesize,filename,mimetype,matchtype
```

Output when theformat of a file is not matched:

```
KO,time,,,,filesize,filename,,matchtype
```

Fido contains a little python script ”toxml.py” that can parse this format into xml. The format of the xml is as below; no schema is available.

```
<fido_output>
   <versions>
          <fido_version>0.9.6</fido_version>
          <signature_version>52</signature_version>
      </versions>
      <file>
          <filename></filename>
          <status></status>
          <matchtype></matchtype>
          <time></time>
          <puid></puid>
          <mimetype></mimetype>
          <formatname></formatname>
          <signaturename></signaturename>
          <filesize></filesize>
      </file>
</fido_output>
```

3.2.9 Unique output identifiers

Identification results are reported as PRONOM Unique Identifiers (PUID) and MIME types. In addition, Fido also provides a textual description of the format name and (if applicable) a signature name.

3.2.10 Granularity of output

The use of PUID as the primary identifier ensures that Fido’s output can be mapped directly to the PRONOM registry (as well as the nascent OPF registry).

3.2.11 Deployment and usage

Getting started with the Fido tool requires some minor preparation steps. This section will shortly describe how to deploy and use the Fido tool.

The evaluated version of the Fido tool package can be downloaded from:
https://github.com/openplanets/fido/zipball/0.9.6

After downloading the ZIP package, create a tool folder of your choice and extract the contents to it. This completes the deployment.

3.2.11.1 Command Line Interface

FIDO can only be used from the command line. To analyse a set of files located in a tree of folders starting in “myFolderToAnalyze”, issue the following command:

```
python fido/fido.py -recurse  myFolderToAnalyze
```

This commands prints the results of the analysis to standard out, which can be piped through the 'toxml' script to format the results as XML:

```
python fido/fido.py -recurse myFolderToAnalyze | python fido/toxml.py
```

3.3 APACHE TIKA™ 1.0

3.3.1 Overview

The Apache Tika™ toolkit detects and extracts metadata and structured text content from various documents using existing parser libraries. The libraries work on magic numbers, file extensions, and structural analysis as described in the Apache Tika™ project[14]:

```
<!--
  Description: This xml file defines the valid mime types used by Tika.
  The mime type data within this file is based on information from various
  sources like Apache Nutch, Apache HTTP Server, the file(1) command, etc.
-->
```

Apache Tika™ can be used directly from the command line, as a Java library, or in GUI mode. Evaluated here is the most recent version of Apache Tika™, which is Apache Tika™ 1.0 (released September 2011).

3.3.2 Tool interface

Apache Tika™ can be invoked through a graphical user interface and a command line interface, or through a Java API.

It is developed as a Maven 2 project, and will thus integrate naturally in maven based Java projects.

3.3.3 License type

Apache Tika™ is an Apache project, and thus uses the Apache 2.0 License. It depends heavily on other parsing libraries, each with their own licenses. We have examined the list of dependencies, and they all seem to be available under the Apache 2.0 License, or derivatives of the MIT Software License, which is compatible with the Apache 2.0 License.

3.3.4 Language

Apache Tika™ is written in Java (version 5). It seems to function well on Java 6 update 22, which is the version used by the author of this evaluation.

3.3.5 Platform dependencies

There are no overtly listed platform dependencies; as such Apache Tika™ should run on any system that is able to run Java 5.

[14] https://github.com/openplanets/tika/blob/master/tika-core/src/main/resources/org/apache/tika/mime/tika-mimetypes.xml

3.3.6 Coverage of file formats

Apache Tika™ implements many different detection algorithms, and can easily be extended to incorporate new ones.

- Mime type from magic numbers
- Mime type from filename extension
- Already known mime type (e.g. when downloading a file from a webserver)
- Content aware detection (including container formats, e.g. OLE2 documents)
- Language detection

3.3.7 Extendibility

Apache Tika™ can be extended in two ways. Firstly, for file identification, one can extend the configuration files used by the mime-type detection, in particular, Apache Tika™ provide instructions on how to add your own mime-types (http://tika.apache.org/1.0/parser_guide.html#Add_your_MIME-Type). This requires knowledge of magic bytes in the file formats, but you can add these formats quite easily to Apache Tika™.

The second way to extend Apache Tika™ is to write your own parser by implementing a simple Java interface. This is how the Content aware detection from the list above has been implemented.

3.3.8 Output format

Apache Tika™ can output information in a number of formats.

- XHTML content (default)
- HTML content
- JSON content
- Plain text content
- Plain text content (main content only)
- Only metadata
- Only language

The documentation provides no further specification about the output format.

Unfortunately, Apache Tika™ also seems unable to combine XML and "metadata only" on the command line. For example, if you choose the –x option (output in XML format), you get a nice XML output with all the metadata; however if you pass it a DOC file, you do not only get the metadata in the XML output, but you also get the content of the document too. This can lead to long processing times and some very large output files. You can use the –m option which triggers Apache Tika™ to only include metadata and discard the content, but you cannot combine it with the –x option to get it in XML format. I.e. it is not possible to get metadata without content in an XML format.

Output format restrictions can be overcome by using the provided API. Using the API adds some extra effort on writing a wrapper tool around it, but it is the most flexible way to benefit from Apache Tika™'s abilities and extract only the metadata you need for your approach. There are many

examples on the web about using the Apache Tika™ API. One of these examples is the evaluation tool used for the results in section 4. The code for this is located at https://github.com/blekinge/Tika-identification-Wrapper.

3.3.9 Unique output identifiers

Mime types are used as format identifiers.

3.3.10 Granularity of output

Apache Tika™ is primarily a metadata extraction tool. It will extract a lot of relevant metadata from the files, but the format is only specified as a mime-type. At the time of writing, these mime-types do not contain version information. This shortcoming has been explored during this work and is described in section 5.1.1

3.3.11 Deployment and usage

Apache Tika™ is provided as a JAR file and can be downloaded from the Apache website: http://tika.apache.org/download.html

After downloading the file, place the Apache Tika™ JAR (tika-app-1.0.jar) in a tool folder of your choice. On a windows machine that could be, for example, c:\programs\tika1.0\.

The deployment of the tool is more or less done with this step.

After that, you have 3 choices on how you would like to use Apache Tika™. You can use its API, use it in GUI mode or as a command line tool.

3.3.11.1 GUI Mode

GUI mode is probably not the best choice in most cases, but it can provide a good overview on what Apache Tika™ returns in command line mode.

Open a command line window and enter the following command:

```
java -jar c:\programs\tika1.0\tika-app-1.0.jar -g
```

On the provided GUI, use FILE for selecting a file you would like to analyse and VIEW for selecting the desired output format. On Apache Tika™'s output screen, you will find the metadata extracted from the selected file, e.g. “Content-Type”, “Content-Encoding”, etc.

The options you find on the GUI reflect the options you have if you are using it as a command line tool.

3.3.11.2 Command Line Interface

Command line mode is a good choice if you would like to run a short test on a particular file or if you would like to run it in a batch or shell script. If you pass it a file path, Apache Tika™ will analyse this file and output the result to standard out.

For example, you could issue the following command on the command line or in a script to analyse a sample file and then output the result in an XML formatted text file:

```
    java -jar c:\programs\tika1.0\tika-app-1.0.jar -x c:\docs\myExample.doc >
c:\result.txt
```

The parameter behind the jar path ("-x") defines the output structure. Further parameters are available that can be used to obtain different output formats, however these cannot be used in combination. This shortcoming is not present when using Apache Tika™ though its API. See next subsection

3.3.11.3 API

Apache Tika™ API usage is your choice if you would like to have the most control over:

- What you would like to analyse (files, folders, a set of folders...)
- Which information you would like to extract (which metadata)
- In which format you would like the output
- How you want to deal with error handling (various Exceptions that might occur)

This usage scenario requires some java programming skills and the Apache Tika™ JAR as a dependency of your JAVA project. There are many examples available on the web that demonstrates how to use this API. For an example of such a usage, see section 5.1.5

A good starting point for this approach would be the API description on the Apache Tika™ website: http://tika.apache.org/1.0/api/

A possible project goal for a small program using the Apache Tika™ API could be a program analysing all files in a folder structure, collecting all occurring MIME-TYPES, counting them and returning a list of all found MIME-TYPES with the number of occurrences. A program of that kind could be used as a standalone component, as well as in bigger workflows, for mass processing of large folder structures, e.g. web archive content.

4 Evaluation of Selected Tools

Before we started to improve the chosen tools, we wanted to evaluate them. So this section presents such an evaluation. It also presents a evaluation framework for easy and mostly automatic evaluation. A framework that, apart from what is presented here, will be put to extensive use during the remainder of the SCAPE project.

4.1 The Scape Characterisation Tool Testing Suite

In order to fairly compare DROID, Fido and Apache Tika™ we have created a testing framework based on the Govdocs1 digital Corpora[15]. This framework uses the characterisation results from Forensic Innovations, Inc.[16] as ground truths.

The framework is available from https://github.com/openplanets/Scape-Tool-Tester

All the tested tools use a different identification scheme for the formats of the identified files. As a common denominator, we have decided to use mime types. Still, mime types are not detailed enough to describe all the relevant information about a file format but all the tested tools are capable of reducing their more complete results to mime types. Therefore, by using MIME types, we ensure a level playing field for the evaluation.

4.1.1 The ground truth and the corpus

The Govdocs1 corpus is a freely available set of about 1 million files (984307 to be exact). Forensic Innovations, Inc. have kindly provided the ground truth for this testing framework, as a ZIP file, accessible at: http://digitalcorpora.org/corp/files/govdocs1/groundtruth-fitools.zip. Unfortunately, they do not list the MIME type for the files, but rather a vendor specific numeric ID. They do provide a mapping between these IDs and the MIME types:http://www.forensicinnovations.com/formats-mime.html. This list is not complete, as they have not provided MIME types for certain formats for which they claimno MIME type have been defined. For the testing suite, we have chosen to disregard files that Forensic Innovations, Inc. do not provide MIME types for, as they make up a very small part of the collection (6422 files, ie. 0.65%). The remaining files number 977885. It should be noted that these files are only disregarded in the subsequent data processing and not in the actual identification process. The identification tool still tries to identify all the files in the corpus.

This leaves us with 87 different formats for the coverage evaluation. The distribution of formats over the corpus is rather uneven as some of the formats are only represented by a single file, while other formats make up close to 25% of the corpus. Figure 1 shows the breakdown of the corpus by file format, focusing on the 20 most common file formats in the corpus. The remaining 67 formats are grouped together in what we call "the long tail". These 67 formats only make up 0.56% of the total number of files in the corpus but depitching them all in the graph would make the graph unreadable. In a subsequent graph we will examine this long tail in more detail.

[15] http://digitalcorpora.org/corpora/files

[16] http://www.forensicinnovations.com/

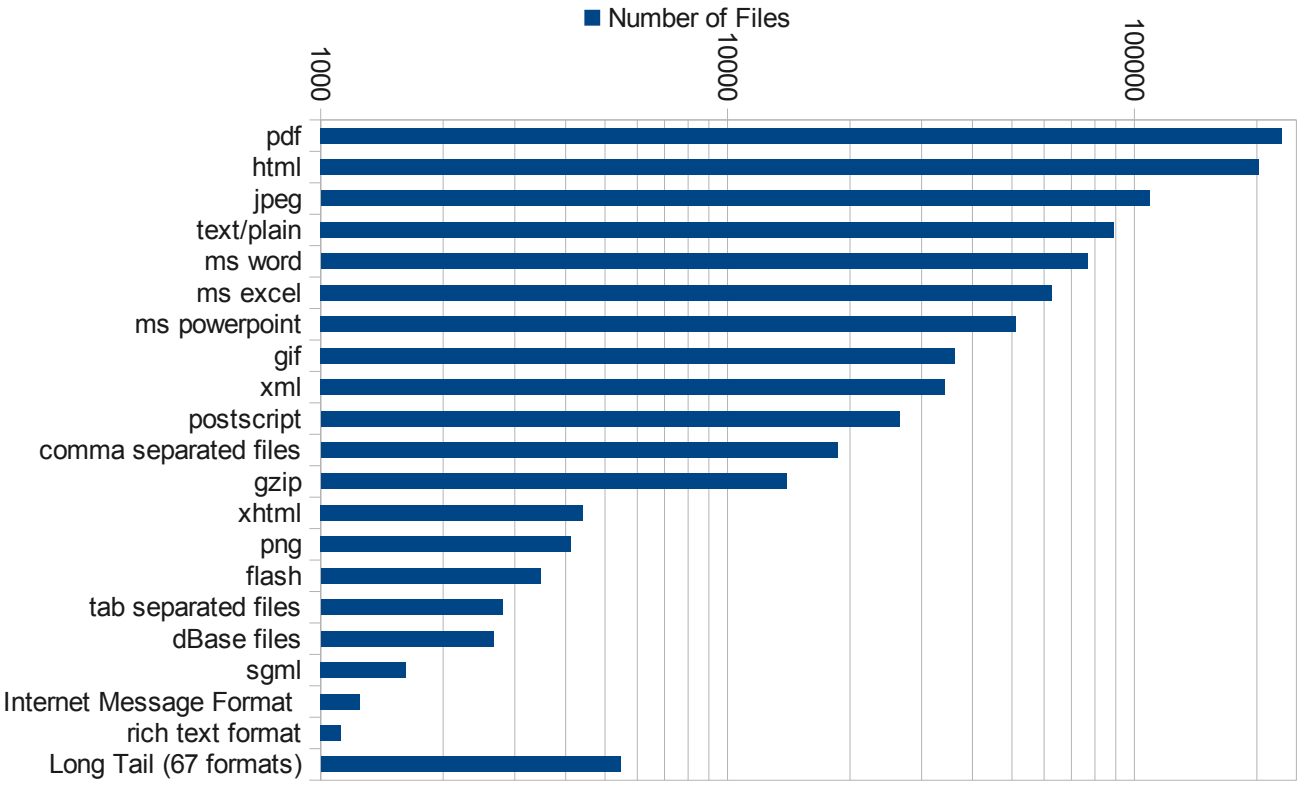

Figure 1: Number of Files by Format

One interesting characteristic of the ID-to-mime table from Forensic Innovations, Inc. is that each format only has one MIME type which is patently untrue. Many formats have several MIME types, the best known example probably being text/xml and application/xml. To solve this problem, we have introduced the MIME type equivalent list, which amends the ground truths with additional MIME types for certain formats. It should be noted that this list has been constructed by hand, simply by looking at the result of the characterisation tools. Any results that do not match the ground truth is recorded as an error, but inspection of the logs later have allowed us to pick up the results that should not have been errors, but rather alias results.

4.2 The test iterator

We have endeavoured to use the tools in a production-like way for benchmarking purposes. This means that we have attempted to use the tools' own built-in recursion features to avoid redundant program start up times, which is mainly relevant for the Java based tools. Likewise, we have, where possible, disabled those parts of the tools that are not needed for format identification (most relevant for Apache Tika™ - see 4.2.1). We have also hidden the filenames from the tools by simple renaming the data files in order to test their format identification capabilities, without resorting to file extension analysis.

4.2.1 Apache Tika™ – a special note

For this test, Apache Tika™ has been used as a Java library wrapped in a specialised Java program (https://github.com/openplanets/Tika-identification-Wrapper). This way, we can ensure that only

the relevant parts of Apache Tika™ are being invoked (i.e. identification) and not the considerably slower metadata extraction parts. By letting Java, rather than the test framework, handle the iteration over the files in the archive, we have also been able to measure the performance in a real mass processing situation, rather than the large overhead in starting the JVM for each file.

4.3 Results

We tested how precisely the tools are able to produce results that match the ground truths. As stated, we focused on the 20 most common formats in the corpus, bundling the remainder into the long tail group of formats.

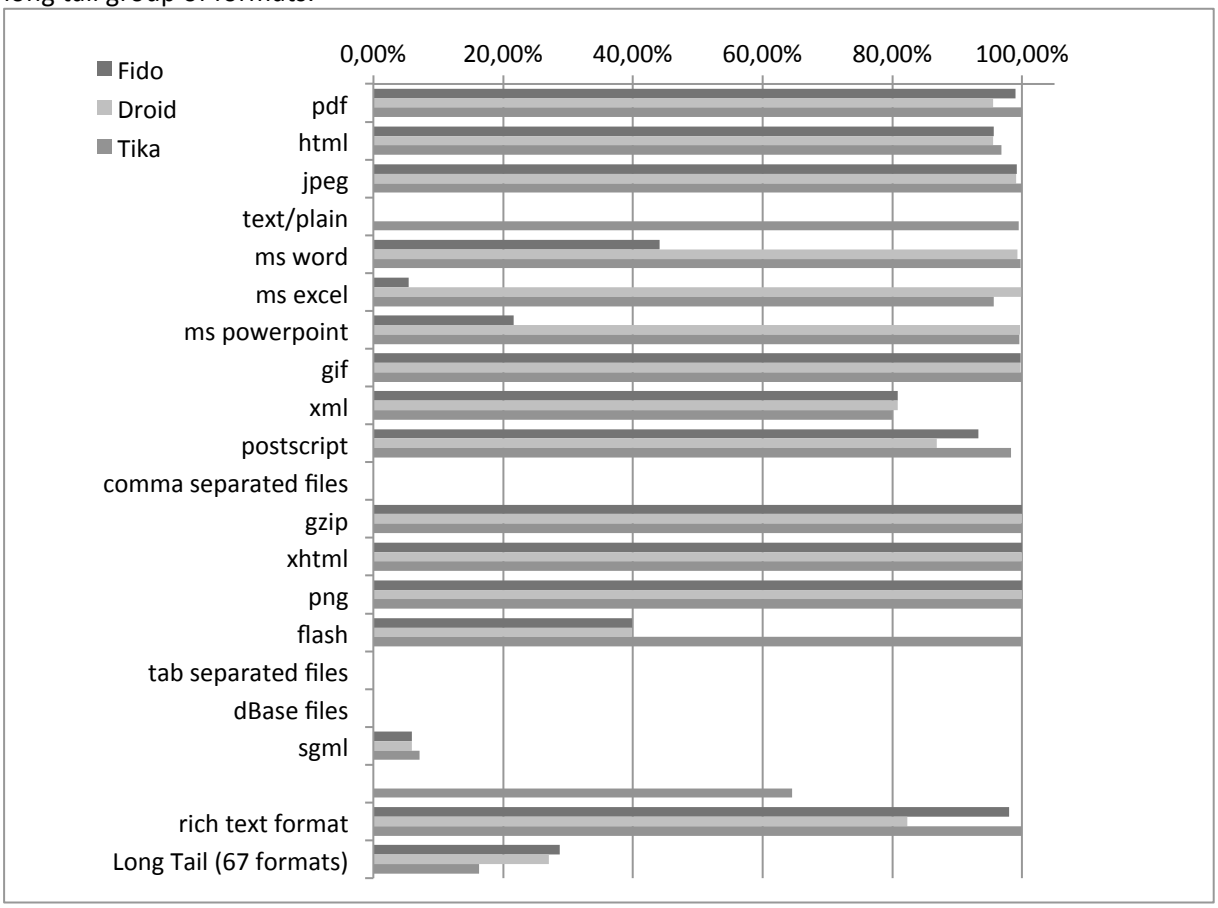

Figure 2: Percentage of Correctly Identified File Types by Tool

As can be seen from Figure 2, Apache Tika™ generally performs best for all of the 20 most common formats, in particular for text files (text/plain), as it is the only tested tool that correctly identifies all these files. For Microsoft Office files, especially Microsoft Excel and Microsoft PowerPoint, DROID seems to be more precise; Apache Tika™ is almost as precise, but Fido looses greatly here. Given that Fido is based on the DROID signatures, it is surprising that, when compared to DROID, Fido underperforms on these formats and yet outperforms it on other formats, e.g. PDF, postscript, and rich text format. No formal analysis of this behaviour has been performed, and so the reason will not be speculated upon.

Comma and tab separated data files are fairly common in the corpus. These formats can not be detected by Apache Tika™. Instead Apache Tika™ simply categorises them as text/plain files. Fido and DROID fail completely to identify both comma/tab separated data files and text/plain files

The dBase files, a somewhat abundant format in the corpus, are not detected by any of the tools.

Apache Tika™ have a 65% hit rate on RFC2822 formatted files while both Fido and DROID fails completely.

All three tools have the same low hit rate on SGML files.

4.3.1 The Long Tail

As mentioned above we have 67 very scarce formats in the corpus. Of these formates only 19 are detected by any of the three tools. Interestingly, DROID and Fido seem to work much better than Apache Tika™ on the long tail of formats.

Table 1 shows the coverage of each tool with reference to the total number in the Ground Truths file (Formats that none of the tools managed to identify have been removed). This table serves to

Table 1: File Format Coverage of each Tool

	Tika	Droid	Fido	Ground Truth
fits image	0	1040	1040	1057
microsoft openxml document	420	0	0	422
application xml	240	263	263	290
application pdf	0	0	80	166
application/rdf+xml	84	0	0	86
bmp image	72	72	72	72
autoCAD dxf	0	32	35	39
tiff image	31	31	31	31
zip archive	14	14	14	14
Virtual Reality Markup Language	0	12	12	12
lotus 123 document	10	0	10	10
ms windows media	7	7	7	7
gnu tar archive	5	0	0	5
oasis document	2	0	0	2
photoshop image	2	0	0	2
autoCAD dwg	2	2	2	2
excel spreadsheet	0	1	1	1
bzip archive	1	1	1	1
unix shell script	0	0	1	1

highlight that it is not just different levels of precision that matter, but which formats are supported by which tools.

DROID and Fido support the Fits image format. Apache Tika™ does not. Apache Tika™ however, supports the openxml document format, which Fido and DROID do not—and so on.

The two formats application/pdf and application/xml are some rather odd files in this table. The ground truth have chosen, for reasons unknown, to separate these two files formats from the great masses of pdf and xml files, which we examined in the previous section. Their apperance in this table

could be the symptom of some anomaly in either the Govdocs1, the detection results from Forensic Innovations, Inc or our evaluation framework. Due to time constraints and the minor number of files involved we have not examined this further.

It is clear that while the overall precision in the long tail is almost equivalent for the three tools, the coverage differs greatly. If Apache Tika™, for example, gained support for the fits image format, it would outperform DROID and Fido on the long tail. Droid and Fido, however, would score much higher, if they gained Apache Tika™'s support for Microsoft Office openxml documents. After the work for this report was completed it came to our knowledge that DROID 6 actually supports the Microsoft Office openxml format. Why our version fails to detect it will be examined in work following this report.

4.3.2 The speed of the tools

For production use of these tools, not just the precision, but also the performance of the tools is critical. For each tool, we timed the execution of the evaluation, to give us a measure of the absolute time in which the tool is able to parse the archive. Of course, getting precise numbers is difficult, as keeping an execution totally free of delays is almost impossible on modern computer systems.

We ran each of the tools on a Dell PowerEdge m160 Blade server, with two Intel(R) Xeon(R) CPU X5670 @ 2.93GHz. The server had 70 GB RAM, in the form of 1333MHz Dual Ranked LV RDIMMs.

The corpus was mounted on a file server accessed through a mounted Network File System via Gigabit network interface.

Each of the tools were allowed to run as the only significant process on the given machine, but we could not ensure that no delays were caused by the network, as this machine was shared with other processes in the organisation.

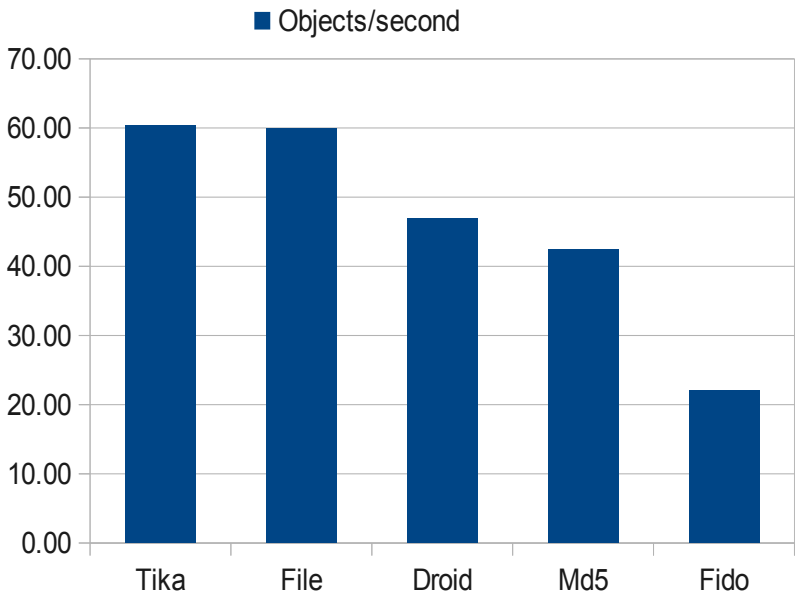

Figure 3:Files Processed per Second for each Tool

To give a sense of the numbers in figure 3 we have run two well known tools on the files of the corpus. None of the tools are to be considered part of the work for their finctionality. Instead they help in giving and judging the speed results.

The Unix File tool checks the file headers against a database of signatures. If a given tool is significantly faster than the File tool, it indicates that the tool was able to identify the file without reading the contents. To do so, it would probably have to rely on filenames. Apache Tika™ seems to be faster, but with a small margin covered by the uncertainties in the system.

Md5 is a tool for calculating check sums of files. To perform such a calculation the tool needs to read the entire file. For the system in question the actual checksum calculation is negligible, so the Md5 tool gives a sense of what it takes to read the entire contents of the archive.

As can be seen, Apache Tika™ is the fastest of the tools, and Fido is the slowest. That the showdown was to be between Tika and Droid was expected. Python does not have the highly optimised Hotspot JVM and will not be able to compete with JVM programs for processes such as these. That Fido was even slower than Md5 came as a surprise, but again, Md5 is written in very optimised C while Fido is running using Python.

While just one process was started for each of the tools, some of the tools have the capability to run multithreaded, while others do not. Fido in particular will only use a single thread, which could account for it's much slower speed. As such, deeper examination will be required, as the speed of the tools is probably proportional to their CPU usage, and some tools will use the CPU much more extensively than others.

5 Work on the selected tools

This section discusses work that has been done to extend each tool or supporting work to enable adoption by the SCAPE community (e.g. services for Taverna workflows).

The work described here were carried out in parallel to the evaluation described in the previous section. Therefore none of the improvements have been evaluated in regard to coverage or speed of the tools.

Text surrounded by <> indicates an XML element from a context-related XML file, e.g. <mime-type> refers to the mime-type element in the Apache Tika™ signature file.

5.1 Apache Tika™

5.1.1 Fine-grained Mime-type Support

5.1.1.1 Problem Addressed

Apache Tika™ currently does not provide version specific mime-type information for each file it parses, e.g., it responds with "PDF" rather than a specific PDF version such as "PDF-1.4". The reason for this is because the Apache Tika™ signature file currently only contains <mime-type> fragments for top level signatures rather than version specific instances; adding version information into the signature file provides a simple approach to improving the identification capabilities of Apache Tika™.

5.1.1.2 Work Done

The Apache Tika™ signature file was updated with eight additional mime-type XML fragments specifying the PDF version (version 1.0 - 1.7) in the <mime-type> and the associated magic <match> value, i.e.,

```
<mime-type type="application/pdf; version=1.0">
  <acronym>PDF 1.0</acronym>
  <_comment>Portable Document Format - Version 1.0</_comment>
  <sub-class-of type="application/pdf"/>
  <magic priority="50">
    <match value="%PDF-1.0" type="string" offset="0"/>
  </magic>
</mime-type>
```

Updates were made to existing unit tests to cope with version-specific responses.

5.1.1.3 Status and Notes

All the unit tests of Apache Tika™ pass indicating PDF versions are correctly identified from new XML fragments in the extended signature file.

However, only PDF versioning has been added to the signature file, so further work is needed to add in new XML fragments containing version information; ideally this could be done by utilising signature files from other tools, but some method of mapping signatures from one signature file to

another is required (i.e. what provides a common key between signature files, e.g. mime-type + version + file extension?)

The above work led to a minor study to determine what affects the processing order of Apache Tika™'s signature file mime-type fragments. The following conclusions were noted:

- Ordering of XML mime-type fragments in the signature file has no effect
- Higher priority fragments are matched first
- Equal priority fragments match against the longer <match> value first (e.g. against "%PDF-1.4" match value first rather than "%PDF-")
- Equal priority, equal <match> value length fragments match against the lexicographically *greater* MIME type first (e.g. "application/rtf" before "application/pdf")

Code is accessible from the GitHub repository: https://github.com/openplanets/tika

If we decide to actually add this functionality to Apache Tika™ the improvements should be added to the Apache Tika™ code base allowing this to live beyond the SCAPE project. As the Apache Tika™ community has proven very open regarding our improvements (see section 5.1.3) we are confident that this is feasible.

5.1.2 Regular Expression Support

5.1.2.1 Problem Addressed

Apache Tika™ uses string/byte matching to ascertain which specific <match> value matches the bytes in the file being parsed. Currently there is no support for regular expression syntax to express the magic pattern, as used by other signature files, such as FIDO. Providing this functionality would enhance Apache Tika™'s signature file and enable cross-tool signature file reuse (e.g. FIDO could be modified to use Apache Tika™'s enhanced signature file).

5.1.2.2 Work Done

The <match> element for PDF-1.4 was modified to use the regular expression from FIDO's signature file (with adjustments to escape '\'s):

```
<match value="(?s)\\A.{0,144}%PDF-1\\.0" type="regex" offset="0"/>
```

A new pattern type ("regex") was also introduced, expanding on the Freedesktop MIME-info format used by the Apache Tika™ signature file (http://standards.freedesktop.org/shared-mime-info-spec/shared-mime-info-spec-latest.html). Apache Tika™ was adapted to handle this new pattern type and process the file using the associated regular expression. A patch has been submitted to include this in the official Apache Tika™ codebase (https://issues.apache.org/jira/browse/TIKA-847).

5.1.2.3 Status and Notes

Parsing using regular expressions has been demonstrated to work. All existing unit tests pass, as do the unit tests created for the regular expression patch.

Further work is needed to add regular expressions to the signature file; ideally, this should be automated to enable future updates from the PRONOM signature file (and potentially others),

although this approach, as for the fine-grained mime-type support, will also require a mapping between <mime-type> fragments in the signature files.

Some notable points:

- Regular expressions require backslash ('\') escaping, e.g. '\\.' translates to the regular expression '\.' which matches a period (full stop); without the backslashes, it would translate to the '.' meta character, matching any single character.
- Many FIDO regular expressions use the '*' meta-character to signify matching "zero or more" characters. Apache Tika™ matching currently works by reading *n* bytes from the file where *n* equals the length of the <match> value. Any non-deterministic length regular expression (i.e. one including a '*' or '+') may potentially not read in the necessary number of bytes from the file to make a correct identification.
- Only Beginning of File (BOF) regular expressions are supported (although offset within the 8KB buffer is allowed).

Code is accessible from the GitHub repository: https://github.com/openplanets/tika

If we decide to actually add this functionality to Apache Tika™ the improvements should be added to the Apache Tika™ code base allowing this to live beyond the SCAPE project. As the Apache Tika™ community has proven very open regarding our improvements (see section 5.1.3) we are confident that this is feasible.

5.1.3 Identifying and Parsing Apple's iBooks Format

5.1.3.1 Problem Addressed:

As this work was being undertaken, Apple released a new eBook-authoring tool called iBooks Author (v.1.0). This piece of software also defines at least two new file formats: the .iba format (used when editing documents, analogous to Microsoft Office .doc), and the .ibooks formats (for publication, analogous to PDF or, more accurately, ePub). These formats, particularly the latter publication format, are likely to start appearing in institutional collections during the lifetime of the SCAPE project. Therefore, it is desirable to at least be able to identify these formats, and ideally to be able to perform some metadata and content extraction (at least for indexing purposes).

5.1.3.2 Work Done:

The .ibooks format is structurally very similar to the ePub format, and therefore it was possible to modify the existing Apache Tika™ ePub support and extend it to cover .ibooks. For identification purposes, this means spotting the presence of an 'application/x-ibooks-zip' MIME Type in the relevant ZIP file entry instead of 'application/epub+zip'.

For parsing, the ePubParser was altered to accept application/x-ibooks+zip format data streams and an iBooks unit test and example file were added. This allowed the general metadata to be extracted from the eBook, but the content could not be extracted trivially due to the way Apple have modified the ePub format.

5.1.3.3 Status and Notes:

Despite the current limitations, the basic identification and parsing logic have been accepted into the Apache Tika™ codebase and will be included in the official 1.1 release[17].

In the future, it would be desirable to add a dedicated iBooks parser that is capable of compensating for the current problems in extracting the textual content of the items. In addition, although they are less likely to turn up in general collections, it would be desirable to add at least support for identifying .iba documents to Apache Tika™.

5.1.4 Fine-grained Language Detection Support

5.1.4.1 Problem Addressed:

Language Detection is used to identify what natural language (English, German…) a given document is in. Currently, Apache Tika's Language Detection is based on 3-grams using Pearson's chi-square test, which has problems with short runs of text, as well as noisy text with errors in spelling and grammar. The issue is relevant as such noisy text can be frequently found on HTML pages or as result of OCR errors. The corresponding JIRA issue is https://issues.apache.org/jira/browse/TIKA-369, and is classified in the Apache Tika™ community as "major".

5.1.4.2 Work Done:

After contacting the Tika community, we implemented and experimented with the log-likelihood ratio (LLR) based procedure discussed in (Dunning kein Datum) and suggested in the JIRA issue. We found improvements in Language Detection on short runs of text.

5.1.4.3 Status and Notes:

After encouraging results in our experiments, we will proceed to integrate the Language Detection support into Tika.

5.1.5 Apache Tika™ RESTful Web Service for Taverna Workflows

5.1.5.1 Problem Addressed:

Whilst the Toolwrapper can be used for generating an Apache Tika™ WSDL service that can be incorporated into a Taverna workflow, it requires a user to install Apache Tika™ to their local machine. Instead, a RESTful web service could be hosted externally (to their local machine, e.g. on the web) to provide an accessible and scalable service for use in Taverna workflows.

5.1.5.2 Work Done:

The SCAPE Apache Tika™ implementation was wrapped in a RESTful web service, built using Jersey (http://jersey.java.net/), and the resulting WAR file deployed to a local Apache Tomcat 7 web container.

---

[17] https://issues.apache.org/jira/browse/TIKA-849?page=com.atlassian.jira.plugin.system.issuetabpanels:all-tabpanel#issue-tabs

5.1.5.3 Status and Notes:

The current RESTful API exposes two resources, one to retrieve the mime-type of a specified file, the other to completely parse the specified file using Apache Tika™'s Parser API (http://tika.apache.org/1.0/parser.html) and return a String representation of the file's metadata:

```
GET tika/rest/mime?file=<path_to_local_file>
GET tika/rest/parse?file=<path_to_local_file>
```

5.1.6 Tika Identification Wrapper

The command line interface of Apache Tika™ places some heavy restrictions on the actions you can take with Tika. This is a shame, as the tool lends itself well to identification actions, when used as a Java library. When used directly through java, it is possible to control which operations should be performed on the files, which can provide significant savings in execution time.

To alleviate these issues, we have made the Tika Identification Wrapper (https://github.com/openplanets/Tika-identification-Wrapper). This tool wraps the identification parts of Tika, but dispenses with the metadata extraction parts.

5.2 DROID

DROID is a very important tool for digital preservation, as it is widely known and used, and closely linked with the PRONOM format registry. In order to make optimal use of it on the SCAPE platform, it was necessary to investigate the DROID 6 codebase to determine how best to exploit it.

5.2.1 Fork of DROID to Github

The DROID source code is officially released via SourceForge (http://sourceforge.net/projects/droid/files/). Unfortunately, this source package is incomplete and could not be built. Therefore, and in agreement with TNA, we set up a public SCAPE fork of the DROID code where modifications could be tracked (https://github.com/openplanets/droid/). This repository has a 'releases' branch where TNA official DROID releases can be checked in, using the branching mechanism to merge their changes in with ours (in the master branch).

On GitHub, we fixed the build, which mainly involved slight modifications to the Maven POM files, but also required a full third-party library (byteseek v1.1) to be pulled in from a separate project (http://code.google.com/p/byteseek/). The remaining differences appeared to arise from the build being targeted at a particular version of the NetBeans IDE. It was not possible to fix all such build problems – for example, we were not able to fix the configuration for the CheckStyle plugin that was being used to validate the coding style.

While this allowed the code to build, there were many problems with the implemented units tests in the project, again due to implicit assumptions in the codebase. In particular, a number of tests came with variants that were designed to work in the presence of a web proxy. This was problematic because these tests contained hard-coded references to a particular TNA web proxy, and because it is not clear what to do with any of these tests when running where there is no proxy at all. Currently, the solution has been to disable these tests, but it would be preferable to spin-up a Java-based pseudo-proxy during the testing phase.

To aid deployment, the necessary files for building a DROID Debian Package were added (https://github.com/openplanets/droid/tree/master/droid-packaging-tools), including a simplified DROID launch script.

5.2.2 Experimental DROID on HADOOP

In an attempt to run DROID over a web archive as a straightforward HADOOP job a small experimental project was created based on the above mentioned mentioned fork of DRIOD. As DROID runs native Java, this experiment revealed some dependency issues (the DROID codebase references multiple versions of the Spring Framework[18]), and a serious architectural scaling problem within the DROID code base. A particular class (CachedByteArrays.java) was aggressively opening RandomAccessFiles before they were strictly required (i.e. upon construction, rather than when the accessor getRaf was called). When running at scale, this meant that the DROID code opened a large number of files unnecessarily, causing the overall execution process to hit the operating system limit for the maximum number of open files, crashing the execution. This was remedied by moving the creation of the RandomAccessFile to the getRaf method, and also explicitly setting the internal reference to null when the file was closed, to ensure the class was garbage-collected and the file handle dropped.

The HADOOP job was based on a reduced version of DROID, which can be found here: https://github.com/openplanets/scape/tree/master/pc-cc-nanite. It comes with a simpler command-line interface as well as clearer dependencies, but is not fully tested and does not integrate with the PRONOM Service efficiently at present.

5.2.3 Tool descriptor for DROID

Due to the complexities involved in using the DROID source code distribution, we also experimented with using the binary distribution directly. A tool descriptor has been created that, using a shell script, makes it possible to run DROID through a SOAP interface. This can be found here: https://github.com/openplanets/scape/tree/master/xa-toolwrapper.

5.3 FIDO

Fido has been wrapped to run in the Scape evaluation framework. This entailed no actual changes to the Fido code.

The Fido tool turned out to have coverage comparable, if worse, than Droid and Apache Tika™. It ran slower than these tools as well. Investigating possible fixes to these issues should be done, if we want to work further with Fido.

[18] http://www.springsource.org/

6 Context Based Characterization of Web Content

In addition to the above described work this work package also deals with complex objects and their contexts. During the first year of the SCAPE project we have worked towards a better understanding of this task and are finally able to produce an overview. The task of characterising complex objects is split in two main strands. Firstly it deals with embedded objects and their tecnological dependencies and secondly it deals with text mining technology. The former fits nicely with the work begun in the last twelve months. The latter strand has allocated one third of the total amount of work in this work package and, as monetioned,proved hard to define. We will therefore elaborate upon that strand in the following.

In the course of the first iteration, a road map and concrete goals for context based characterization within digital preservation have been defined for the next two years. The overall goal is to support the growing use of web content for analytical purposes by allowing analysis of large scale collections of web pages from multiple points of views. Characterization of HTML pages will take place on both a syntactic and a content level and all results will be closely tied in with the efforts in the Watch component subproject.

In the following two subsections, an overview of the planned effort on both levels of characterization will be given, along with the stated goals and relevance for the Watch component. In the subsequent subsection, the priorities for year two of the project as well as the second deliverable are defined.

6.1 Syntactic Characterization

The goal of syntax-level characterization is to achieve fine-grained analysis of syntactic elements of script languages in text files in order to build usage statistics of certain tags or methods. These statistics are to be computed over large collections of HTML pages, each of which represents the point in time in which the collection was crawled. By computing statistics over snapshots of the Web over different years, we will establish a framework for measuring emerging trends in Web technologies, as well as technologies that are slowly being phased out. The characterization will be fine-grained, so that – for example - statistics over specific tags can be computed. Accordingly, the overall effort has been internally dubbed ‘measuring tag decay’. In order to analyse entire collections, the characterization effort will need to scale, which will be achieved through coordination with WP6. The generated statistics are to be fed back to the Watch component.

6.2 Web Content Characterization and Trends

The goal of content characterization is to observe specific types of content within HTML and other Web content such as Twitter feeds that are relevant to questions and general trends in the digital preservation community. Common examples for such questions are statistics over the usage of preservation or migration tools and notifications for the emergence of new versions of existing tools, as well as entirely new preservation tools. Other examples consider the usage and emergence of file formats as well as detection of file format obsolescence. An important difference to the efforts described in 6.1 is that this level of characterization directly analyses trends by means of information extraction technologies, e.g. measuring what technologies and tools are currently discussed in the community, for example in forums, Twitter feeds and Web portals. This information is to enable search for relevant indicators of risk assessment issues such as format obsolescence and is to be tied

in with the Watch component subproject. The overall vision is to give users of the Watch component a means to define custom notifications over topics of interest by ways of a controlled vocabulary of operators.

6.3 First Objective for the Second Year of the SCAPE Project

In the second year of the SCAPE project, we will focus our efforts on content characterization. Together with our partners in the digital preservation community we will define a set of source data types (such as forums, blogs, micro blogs) that are especially important to the community and develop a set of classification and extraction methods. The approach is bottom-up and requirement-driven, e.g. together with our partners we will select specific questions of interest to the community and develop methods to address these questions. From there, we will abstract to more general operators in order to facilitate the specification of generic notifications. In the second deliverable, we will present the content characterization system and its specific use cases.

In order to minimize risks in the dependence on a parallel execution environment, the large scale classification of syntactic features as described in 6.1 will not be the focus of year two. We believe that the parallel infrastructure will be more advanced in year three, facilitating the development of methods to measure statistics such as tag decay.

7 Road map for next iteration

As mentioned in chapter 1, the first iteration of this work package has focused on the evaluation and improvement of tools mainly suited for identification of digital objects. We selected three tools: DROID, FIDO and Apache Tika™.

In this upcoming year, we will move from simple identification of digital objects to complex characterisation and expand upon the set of selected tools. We will consider both more complex tools such as FITS and JHOVE2, more simplistic tools such as UNIX file, specialised tools such as ffProbe (use for characterisation of multimedia files) etc.

As we have limited resources, it is paramount that we focus on what will create the most value for the implicated partners, SCAPE, and the Digital Preservation Community. With this in mind, we will examine if one of DROID, Apache Tika™ and FIDO with a minimum of effort can be made into a tool that covers the functionality and performance of the other two. This decision will be based on the evaluation described in this report in conjunction with a discussion of less quantifiable properties such as:

- Developer and user community,
- Ease of adding functionality to the core of the tools—both technical and community-vice,
- Adaptation of the tools both within and outside of the digital preservation community,
- Requirements of the content holders,

In addition, we will expand pattern matching methods for complex characterization for Web sources, following the road map we have worked out as described in chapter 6 in order to tie in characterization efforts and the developed tools with the efforts in the Watch component. The requirements on the tools from the content holders will be extracted from the “Scenarios, Issues, Dataset and Solutions” wiki and in correspondence with the content holders through emails and at face-to-face meetings.

According to the “Description of Work”, the PC.WP1 work package is also obligated to do characterisation on complex files, i.e. files with content under DRM, component files, container files, etc. This objective will be met by considering the tools JHOVE2 and FITS, embedding, extending or learning from them as appropriate.

The actual selection of tools and focus areas for the work will be completed as per the check point CP067 in M15. This also means that any conclusion drawn from the work described in this report will be presented at that check point.

8 Citation references

Askov Blekinge, Asger. “Tika evaluation.” draft, Preservation Components, Characterisation Components, SCAPE, 2011.

Dunning, Ted. “Accurate Methods for the Statistics of Surprise and Coincidence.” New Mexico State University.

Raditsch, Markus. “Experimentel 'real world ARC' extraction report.” draft, Testbed, Web Content Testbed, SCAPE, 2011.

The National Archives. “Documentation - droid.” *SourceForge.net.* http://sourceforge.net/apps/mediawiki/droid/index.php?title=Documentation (accessed 02 14, 2012).

—. “Licensing: droid.” *SourceForge.net.* http://sourceforge.net/apps/mediawiki/droid/index.php?title=Licensing: (accessed 02 14, 2012).

—. “Sourcecode.” *SourceForge.net.* http://sourceforge.net/apps/mediawiki/droid/index.php?title=Sourcecode (accessed 02 14, 2012).

van der Knijff, Johan, and Carl Wilson. “Evaluation of Characterisation Tools.” check point report, Preservation Components, Characterisation Components, SCAPE, 2011.

van der Knijff, Johan, Asger Askov Blekinge, and Sven Schlarb. “Characterisation Evaluation Framework.” Check Point report, Preservation Components, Characterisation Components, SCAPE, 2011.

van der Knijff, Johan, Asger Askov Blekinge, and Sven Schlarb. “WP 9: target characterisation tools.” draft, Preservation Components, Characterisation Components, SCAPE, 2011.