Web content executable workflows for experimental execution

Authors
Markus Raditsch, Sven Schlarb (Austrian National Library), Per Møldrup-Dalum (The State & University Library Aarhus), Leila Medjkoune (Internet Memory Foundation)

April 2012

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).

Executive Summary

The objective of the Web Content Testbed work package TB.WP.1 is to develop workflows for preserving the content of web archives. The specific challenge of web archives is the heterogeneity of its content, especially the huge variety of digital objects of different file formats. A web archive contains text content, loosely following a variety HTML format specifications, it also contains images, audio, and video content where only a minor part strictly follows the corresponding file format specifications. In order to preserve this content and to ensure accessibility to the content in the future, the goal of work package TB.WP1 is to develop workflows that make use of tools and services that have been evaluated and/or developed in the PC sub-project.

In the first experimental phase of work package TB.WP1, the focus was to provide implementations of workflows as executable solutions for concrete scenarios, describing issues with the data sets that institutions are facing with their web archive content.

The work done was organised in two strands, each with a different focus: On one hand, work package TB.WP1 considers identification of web content as the first necessary step for any subsequent preservation measure, and applies different identification tools on their web archive content. On the other hand, it makes use of image comparison tools developed in work package PC.WP.3 for web archive related quality assurance workflows.

Table of Contents

1 Web Content Testbed scenarios

Each web content Testbed scenario consists of a description of the data set, the preservation issues that have been identified and a proposed solution which solves the connected issue. These solutions are presented as experimental workflows. The data sets originate from real life institutional repositories, namely, from the web archive of the Austrian National Library[1] (ONB) for the first scenario, and the Internet Memory Foundation[2] for the second scenario.

Scenario 1: *WCT4 - Web archive MIME type detection at the Austrian National Library*[3]

Scenario 2: *WCT1 - Comparison of web archived pages*[4]

The two scenarios correspond to two strands of work package TB.WP1 with different focus:

1. Identification[5] of web content as the first necessary step for any subsequent preservation measure.
2. Image comparison for doing quality assurance in web content preservation workflows.

The first strand focuses on developing identification workflows and mainly makes use of the tools and services that are the outcome of work package PC.WP.1[6]. While work package PC.WP.1 performs an assessment of the tools regarding performance, reliability, and coverage of heterogeneous file formats in general, the objective of the Web Content Testbed work package TB.WP.1 is to assess the applicability of the workflow using a representative test data set taken from the institutional data repository. The second strand of work package TB.WP.1 is focused on image comparison tools and services that are the outcome of work package PC.WP.3[7] (Task 3). That work package is focused on developing tools for using reference snapshots of websites which will then be used to automate quality assurance processes by identifying missing elements, comparing image instances, and identifying similar and dissimilar images, for instance to detect rendering errors.

[1] http://www.onb.ac.at/ev/index.php
[2] http://internetmemory.org/en/
[3] http://wiki.opf-labs.org/display/SP/WCT4+Web+Archive+Mime-Type+detection+at+Austrian+National+Library
[4] http://wiki.opf-labs.org/display/SP/WCT1+Comparison+of+Web+Archive+pages
[5] In this scenario, identification refers only to the process of determining the MIME type of a digital object.
[6] Deliverable D9.1 of work package PC.WP.1
[7] Deliverable D11.1 of work package PC.WP.3

1.1 Web archive MIME type detection at the Austrian National Library

The scenario, including data set, issue, and solution description, is available on the publicly available wiki under the identifier *WCT4 - Web archive MIME type detection at the Austrian National Library*[8].

This scenario is being developed at the Austrian National Library for identification of web archive content in the form of MIME type detection of file formats. The identification of web archives at such a granular level is being considered the first necessary building block towards preservation of the web archive as a whole. Without knowing what exactly the web archive contains, it is in many cases not possible to take appropriate actions like choosing the right migration tool or emulation environment or to elaborate a preservation plan that is supposed to be able to deal with the individual content items.

While work package PC.WP.1[9] did an analysis of existing identification tools and finally chose

- DROID[10] 6.0
- FIDO[11] 0.9.3
- Apache Tika™[12] 1.0

as the candidates for further analysis and improvement, the evaluation has been done in an abstract manner to be applicable for different workflow implementation scenarios in the TB.WP.1 and TB.WP.3 work packages. In the *WCT4 - Web archive MIME type detection at the Austrian National Library* scenario, these identification tools are applied to a data set from the Austrian National Archive's web archive which will be described in the next section, and the focus will be on how on site integration of a workflow could be reached.

Furthermore, this document lines out how initial results regarding performance are evaluated using Taverna Workbench[13] workflows.

1.1.1 Data set - Web archive of the Austrian National Library

The data set for this Web Content Testbed scenario is a test data set taken from the web archive of the Austrian National Library[14]. The data set originates from different types of crawls. It contains data from two types of web harvests:

Selective crawls related to specific events where web sites are frequently harvested during the event, e.g. the 2009's EU election or Olympia 2010 events

Domain crawls from 2009 of about 1 million domains.

---

[8] http://wiki.opf-labs.org/display/SP/WCT4+Web+Archive+Mime-Type+detection+at+Austrian+National+Library
[9] Deliverable D9.1 of work package PC.WP.1
[10] http://sourceforge.net/apps/mediawiki/droid
[11] http://www.openplanetsfoundation.org/software/fido
[12] http://tika.apache.org
[13] http://www.taverna.org.uk
[14] http://wiki.opf-labs.org/display/SP/Austrian+National+Library+-+Web+Archive

Technically, the web archive data store consists of files around 100 Megabyte each with the extension *.ARC.GZ in the GZIP format[15] (extension “gz”). The *.ARC.GZ files themselves concatenate multiple archive files in the ARC file format[16] (extension “arc”). The format is used by The Internet Archive[17].

1.1.2 Issue - Web content characterization

The main issue with the above mentioned data set is the fact that web archive data is very heterogeneous. Depending on the policy of the institution, data contains text documents, html content loosely following different HTML specifications, audio and video files that were encoded with a variety of codecs[18], all sorts of imaginable binary files, etc. First of all, in order to make any decisions in preserving such archives, it is very important to have detailed information about the content, especially those pieces of information that preservation tools depend on.

In many cases, the file format migration of digital objects, for example, depends on the information, like the MIME type, the file format or even on specific properties of the object. It is therefore necessary to identify the single objects contained in an ARC/WARC file, and to identify container formats, like packaged files or any other container formats. Video files, for example, are often available as so called wrapper formats, like AVI, where the audio and video stream can be encoded by different codecs. Down to this level the content stream must be identified if the institutional policy would foresee to preserve all video and audio content contained in a web archive.

Secondly, the issue has two different aspects; one is the challenge to identify content that is already known. In this sense, the main goal of identification is to identify the content correctly. Work package PC.WP.1 is dealing with this aspect by using an annotated corpus (Govdocs1 digital Corpora[19]) that can be used as gold standard in order to verify the reliability of identification results.[20] It must be mentioned, that this corpus is text document centric and does not constitute a representative test data set of web archive content, it is therefore considered as a general indicator of the reliability in this context.

The second aspect is unknown content in the web archive which is measured by the coverage of identification tools, where coverage indicates the part of the content that can be identified. Coverage depends on reliability in the sense that a bad reliability can hide a bad coverage in case that many objects are incorrectly identified, but are actually unknown.

From a practical point of view, the challenge starts with the ARC/WARC file format that ONB and SB[21] as the main stakeholders of this issue are using in their web archive. The Heritrix web crawler[22] produces these files as a result of the web crawls. The business logic and implementation is

[15] http://www.gzip.org/zlib/rfc-gzip.html
[16] http://web.archive.org/web/20021002080721/pages.alexa.com/company/arcformat.html
[17] http://www.archive.org/web/researcher/ArcFileFormat.php
[18] A computer program for encoding / decoding a data stream.
[19] http://digitalcorpora.org/corpora/files
[20] Deliverable D9.1 of work package PC.WP.1
[21] http://en.statsbiblioteket.dk/
[22] https://webarchive.jira.com/wiki/display/Heritrix/Heritrix

accessible[23], but it has been integrated in the web crawler, not in web content preservation workflows. This leads to the subordinate issue of dealing with ARC/WARC files as the basis of web content preservation workflows.

The last aspect of this issue is the fact that several tools are known to generally address these kinds of challenges, still integration of the tools provided by work package PC.WP.1 must be ensured by integrating them into real life workflows.

The description of this issue is available on the publicly available wiki under the identifier **IS25**[24] **Web Content Characterisation**.

1.1.3 Solution - Web Archive MIME type detection

As a solution to this issue an experimental workflow using the Taverna Workflow Workbench, will be presented in detail in section 2 Solution SO17: Web archive MIME type detection. The workflow is exclusively using tools that run locally using the command line interface. Due to the large amount of local data to be processed, the Taverna "tool" service is used as integration mechanism instead of web services. Note that for the preservation catalogue[25] it is still possible to use the tool wrapper[26] in order to provide single components as a web service (REST and SOAP web service interfaces).

In summary, the workflow takes a text file that lists file paths to the web archive container files (*.arc.gz). The workflow then unpackages and analyses these files in parallel and as a result creates a summary report about MIME-Type distribution of the web archive content.

The description of this solution is available on the public wiki under the identifier **SO17 Web Archive MIME-Type detection workflow based on Droid and Apache Tika™**[27].

1.2 Automated QA for Web Archives

This scenario is closely related to the SCAPE work package:

- PC.WP3: Quality Assurance (build and test the automated QA tool)

The scenario described here is looking into applying visual and structural comparison to Web pages to automate quality assurance for Web Archives. Visual and structural comparison tools are developed as part of work package PC.WP3, to improve Web Archives quality by solving crawling or access (rendering) issues.

Part of the tools developed and statistics produced by comparing snapshots should then be used in work package PW.WP1 to feed the Web Watch adaptor and improve format obsolescence detection (WCT7[28]).

---

[23] Heritrix is available as a collaborative code project at Github: https://github.com/internetarchive/heritrix3
[24] http://wiki.opf-labs.org/display/SP/IS25+Web+Content+Characterisation
[25] http://catalogue.scape-project.eu/
[26] https://github.com/openplanets/scape/tree/master/xa-toolwrapper
[27] http://wiki.opf-labs.org/display/SP/SO17+Web+Archive+Mime-Type+detection+workflow+based+on+Droid+and+Apache+Tika
[28] http://wiki.opf-labs.org/display/SP/WCT7+Format+obsolescence+detection

The scenario, including data set, issue, and solution description, is available on the publicly available wiki under the identifier *WCT1 - Comparison of web archived pages* [29].

1.2.1 Data set - Internet Memory Foundation Web Archive

To develop and train the visual and structural tool, Internet Memory Foundation worked closely with the UPMC[30] team by providing pairs of annotated Web Pages[31] from its Web Archive[32].

To complete the first test our Quality Assurance team provided around 1000 pairs of annotated web pages.

We defined an assessment plan as follows:

- Pages: Several levels of pages (homepage, page at deeper level that might be updated less frequency)

- Frequency: Close in time (1 month is the largest interval)

- **Method:** The QA team will compare both the structure and the content of pages to create pairs. Page will be checked entirely.

Evaluation criteria: Would we ask for a re-crawl in this case?

- **Number of data:** At least 100 annotated pairs each week and as much as possible during the test period.

- **Tag**: Annotated pairs should be tagged as: similar, dissimilar and ambiguous.

- Pairs annotated as similar should present small differences (but not big enough to require a new crawling).
- Identical pairs must not be considered for annotation.
- Pairs annotated as dissimilar must have strong enough differences.
- Pages from different web sites must not be considered for annotation.

- **Delivery:** Three text files per week containing a list of URLs of the web pages per pair for UPMC to access content online. One text file per tag (similar, dissimilar, ambiguous).

The Internet Memory Foundation QA team delivered around 1000 annotated pairs of pages, from which around 200 pairs had significant differences. These 200 were used to improve the tool training.

1.2.2 Issue - Inconsistency of web archive data

Web archives use crawlers, such as Heritrix[33], for collecting resources from the Web. Crawls can be selective (domain, sub domain or even page or resource level) or large (.uk or .eu domain, for

[29] http://wiki.opf-labs.org/display/SP/WCT1+Comparison+of+Web+Archive+pages
[30] http://www.upmc.fr/
[31] http://wiki.opf-labs.org/display/SP/Internet+Memory+Web+Archive
[32] http://collections.europarchive.org
[33] http://crawler.archive.org/index.html

example). Although improvements are constantly made in the domain of crawling strategies and tools (execution based crawlers or tools, access tools, etc.), web content remains extremely complex to capture, it is a challenge to enable access, and crawls can never be proven exhaustive. Because of this complexity and the steadily increasing size of web archives, developing automated, scalable quality assurance solutions is crucial to the preservation of web content.

The description of this issue is available on the publicly available wiki under the identifier **IS7**[34] : Incompleteness and/or inconsistency of web archive data.

1.2.3 Solution - Comparing web page versions

The solution developed by the UPMC team as part of the SCAPE work package PC.WP3 is available on the wiki under the S018 reference: “Comparing two web page versions for web archiving”[35].

The solution is provided in form of a tool called “MarcAlizer” which allows a structural and a visual comparison of web pages.

The solution is based on:

- a combination of structural and visual comparison methods embedded in a statistical discriminative model
- a visual similarity measure designed for Web pages that improves change detection
- a supervised feature selection method adapted to Web archiving

A related Taverna workflow is available on my experiment and will be explained in section 3 Solution SO18: Comparing two web page versions for web archiving.

2 Solution SO17: Web archive MIME type detection

The following section describes the solution SO17[36]: "Web archive MIME type detection at the Austrian National Library" in detail and gives additional background information on the used tools and workflow design.

2.1 Introduction to the experimental work flow

The focus of work package TB.WP1 is to implement experimental Taverna workflows to identify the critical steps, the right tools and the most efficient workflow design for the given scenario[37]. These workflows analyze each file contained in a list of ARC.GZ files and extract the MIME types. Each ARC.GZ consists of thousands of files. These workflows can be fed with a list of ARC.GZ containers and are able to process hundreds of thousands of content files in a single run.

[34] http://wiki.opf-labs.org/display/SP/IS7+Incompleteness+and+and+inconsistency+of+web+archive+data
[35] http://wiki.opf-labs.org/display/SP/SO18+Comparing+two+web+page+versions+for+web+archiving
[36] http://wiki.opf-labs.org/display/SP/SO17+Web+Archive+Mime-Type+detection+workflow+based+on+Droid+and+Apache+Tika
[37] http://wiki.opf-labs.org/display/SP/WCT4+Web+Archive+Mime-Type+detection+at+Austrian+National+Library

After running over the entire content stored in the ARC.GZ container, the result is a list of MIME types with a counter holding the occurrences for each type. The results will be explained in detail in the following sections and can be used to better estimate the amount of storage needed during the processing phase of the extracted web content, the influence of the involved tools on performance, error handling for robust operation and the impact of workflow design (parallelization) on the performance of the workflows.

Based on these more advanced work flows created for technical workflow optimization, further experiments using two selected identification tools have been made. The two tools were the DROID file profiling tool and Apache Tika™.

2.2 Scope of the experiments

Doing file type identification on a single file or only a few files on a local file system is similar to a simple batch processing. But if the web harvests of an entire country domain are to be processed, the requirements change:

- Tools and work flows have to be able to operate fully unattended
- All Components need to run stably while processing hundreds of Terabytes and more
- The process has to be fast
- The process has to be reliable in terms of error handling
- A high coverage[38] of the different digital objects and precise results are desired
- The solution must be easy to set up and can be integrated in different types of preservation infrastructures

At the current stage of the SCAPE project, the TB.WP.1 work package is focused on functionality and experimentation. Scalability issues will be the focus later in the project.

Figure 1 shows the conceptual diagram of the components.

[38] Coverage indicates the part of the content that can be identified correctly.

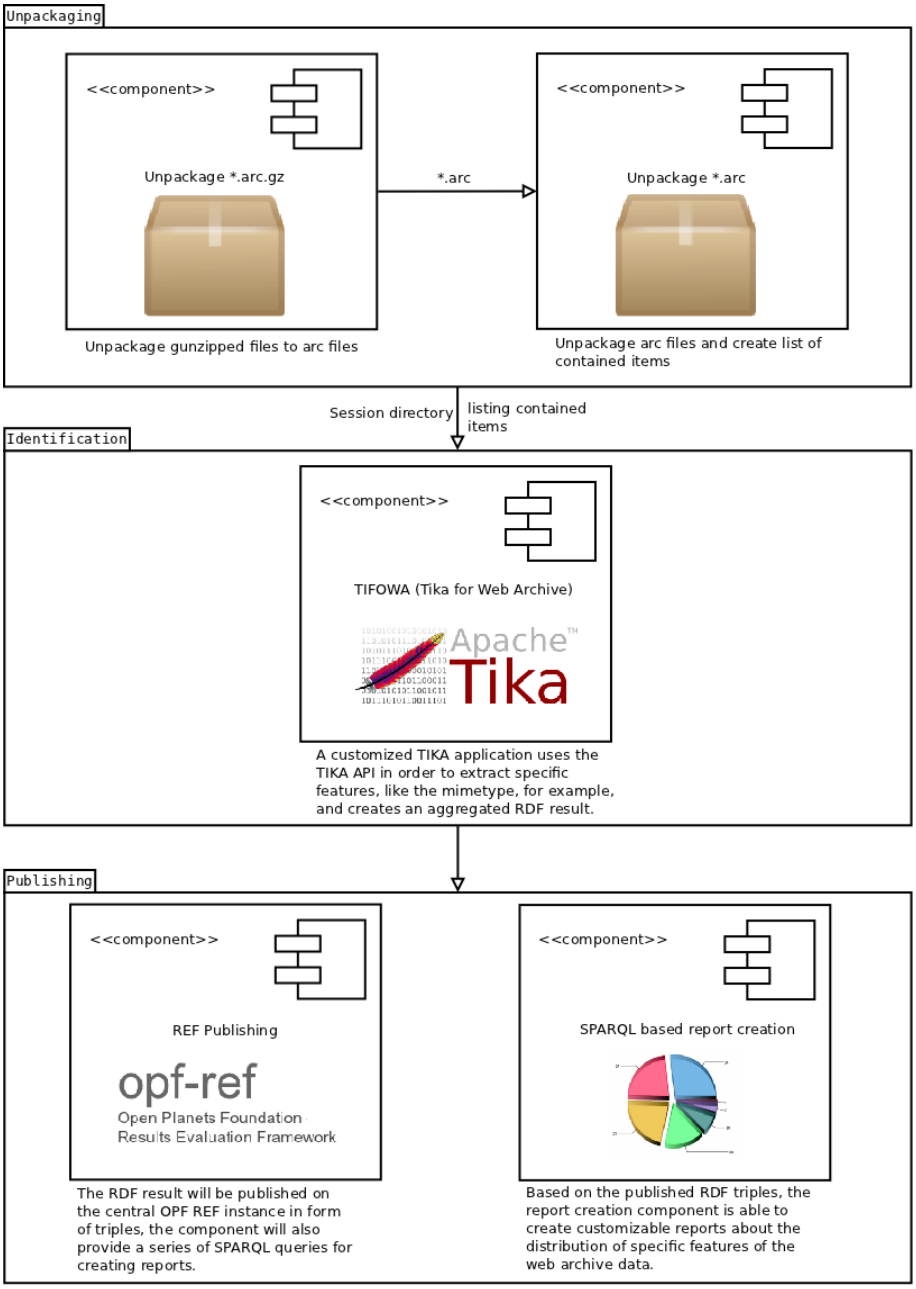

Figure 1: Conceptual workflow diagram

2.3 Prerequisites for the experiments

Before starting with the actual experiments, some prerequisites have to be met. To run the experiments it is required to:

- Outline what the goals of the experiment is
- Have a platform to run the experiments on
- Define a set of tools used for the implementation of the workflow
- Create the implementation of the workflow

On this basis, the following steps have been carried out for planning the implementation of the workflow:

- Identify which file identification tools to use
- Share the results with the community by publishing the tools as web services
- Identify tools for reliable disassembling of ARC.GZ containers
- Design and implement the experimental work flow(s)
- Run the experimental workflow(s)
- Collect the results

2.4 Identification and evaluation of identification tools

One of the main components needed for identifying the objects inside the web archive containers, is the software used to analyze the content files. The quality and reliability of this component is essential for the quality of the entire workflow and the entire series of experiments.

Besides coverage[39] and reliability, processing speed and error handling are also very important criteria. If an error occurs during the file identification process, a transparent error report is required.

For our experiments we chose:

- Apache Tika™ 1.0
- DROID 6.01

The following two sections, describing Apache Tika™ and DROID, are based on input from Section 3, [D9.1][40] which used information from:

- 'Real world ARC' extraction report (Raditsch, M. (2011). Experimentel 'real world ARC' extraction report. draft, SCAPE, Testbed, Web Content Testbed)
- Evaluation of Characterisation Tools (Van der Knijff, J., & Wilson, C. (2011). Evaluation of Characterisation Tools. check point report, SCAPE, Preservation Components, Characterisation Components)
- Target characterisation tools (Van der Knijff, J., Askov Blekinge, A., & Schlarb, S. (2011). WP 9: target characterisation tools. draft, SCAPE, Preservation Components, Characterisation Components)

2.4.1 Apache Tika™

The Apache Tika™ toolkit detects and extracts metadata and structured text content from various documents using existing parser libraries.

It 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).

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

---

[39] Coverage indicates the part of the analyzed content that can be identified correctly.
[40] Deliverable D9.1 of work package PC.WP.1

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

Unfortunately, it is not possible to combine the XML and "metadata only" option on the command line. Example: Choosing the `-x` option (output in XML format), the XML output with all the metadata is returned. Providing a DOC file as input, it returns not only the metadata in the XML output, but the content of the document too. This can lead to long processing times and very large output files. It is possible to use the `-m` option which indicates Apache Tika™ to only include metadata (and discard the content) – but, using the command line interface, it is not possible to combine it with the `-x` option to get it in 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 provides most flexibility[41] to benefit from Apache Tika™'s abilities and extract only the data needed for the here presented approach.

Apache Tika™ is provided as a JAR file and can be downloaded from the Apache website[42].

The **GUI mode** is not the best choice for processing large data sets, but it can still provide some useful insight on what Apache Tika™ is capable of in command line mode.

The **command line mode** is for running batch jobs. Apache Tika™ offers a file path as input parameter which is then analyzed, and it returns the result in the standard output (stdOut). The command line program is not able to do recursive analysis of content in sub folders. Furthermore, it starts a new JVM instance for each call. This becomes problematic regarding performance when processing hundreds of thousands of files or more.

The following command is an example that analyzes a sample file and outputs the result as XML:

```
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 format. The documentation[43] provides information on all parameters.

The **Apache Tika™ API** enables to control:

- The content to be analyzed - including recursive folder structures
- The type and amount of meta data to be extracted
- The output format

[41] Flexibility here means: Analyze recursive folder structures instead of a single file. Border the type and amount of meta data to be extracted. Define the output format. Do more advanced exception handling.
[42] http://tika.apache.org/download.html
[43] http://tika.apache.org/1.0/gettingstarted.html

- Advanced exception handling

To use the API it is necessary to develop an application that makes use of the API.

The requirements for a basic program that uses the Apache Tika™ API are:

- Make the tool accept a parameter holding a folder path (for unattended operation)
- Read the folder recursively
- Detect the MIME types
- Care about exceptions
- Normalize the types (remove whitespaces, convert to lowercase, ...)
- Store each occurring type
- Count the number of occurrences
- Format the output
- Output the result

2.4.2 DROID

DROID (Digital Record Object IDentification) is a tool that attempts to identify digital objects using PRONOM[44] 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. For our experiments, we used the most recent version of DROID , which is DROID 6.01 (released March 2011).

The identification results are stored in a database format that is not human readable. 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 Values) format. By default, each row in the exported CSV file represents one file object.

Additionally, DROID 6 can generate XML and PDF reports, but this report functionality is mainly useful for generating aggregate statistics of the identification results (e.g. for each format the corresponding number of files). None of these output formats are particularly suitable for use in automatic workflow systems

Usage:

Getting started with the DROID tool requires some minor preparation steps. The following will briefly describe how to deploy and use the DROID tool to create a simple report with DROID.

The DROID tool package can be downloaded from The National Archives website[45].

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

[44] http://www.nationalarchives.gov.uk/PRONOM
[45] http://www.nationalarchives.gov.uk/

When analysing a set of files, DROID follows a two step approach. First it is required to add a set of files or folders to a "profile", and run an analysis step afterwards using this profile ending up with the 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
```

In the second step DROID analyzes the content of the 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 easily be combined in a script or, for example, invoked by a Taverna external tool service if it is supposed to run as a Taverna workflow.

2.5 Apache Tika™ and DROID as a web service

Working in an international project requires sharing results and enabling a good understanding of the concrete interfaces and implementations of tools. For that reason a tool called ToolWrapper[46] has been created on the basis of the tool wrapper that had been developed in the IMPACT project.[47] ToolWrapper is a java project to generate web services based on tool descriptor files.

This makes it easy to describe the technical environment needed for a particular tool in an XML file and exposing it as a web service.

This approach enables partners to share their work and allows them to test a tool without having to install the tool and to find out how it is operated. This applies for command line options as well as for the input and output parameters.

In the SCAPE project the Taverna Workbench is used for creating workflows. With this graphical tool, it is straightforward to create workflows based on existing web services and publish them using the myExperiment[48] platform.

For instance: The basic Apache Tika™ workflow below (Figure 2) takes a single input file, analyzes the file and returns the metadata collection result as a XML output file.

Tool used: Apache Tika™ 1.0 (command line triggered through the ToolWrapper)

[46] https://github.com/openplanets/scape/tree/master/xa-toolwrapper
[47] https://github.com/impactcentre/interoperability-framework/tree/master/toolwrapper
[48] http://www.myexperiment.org/

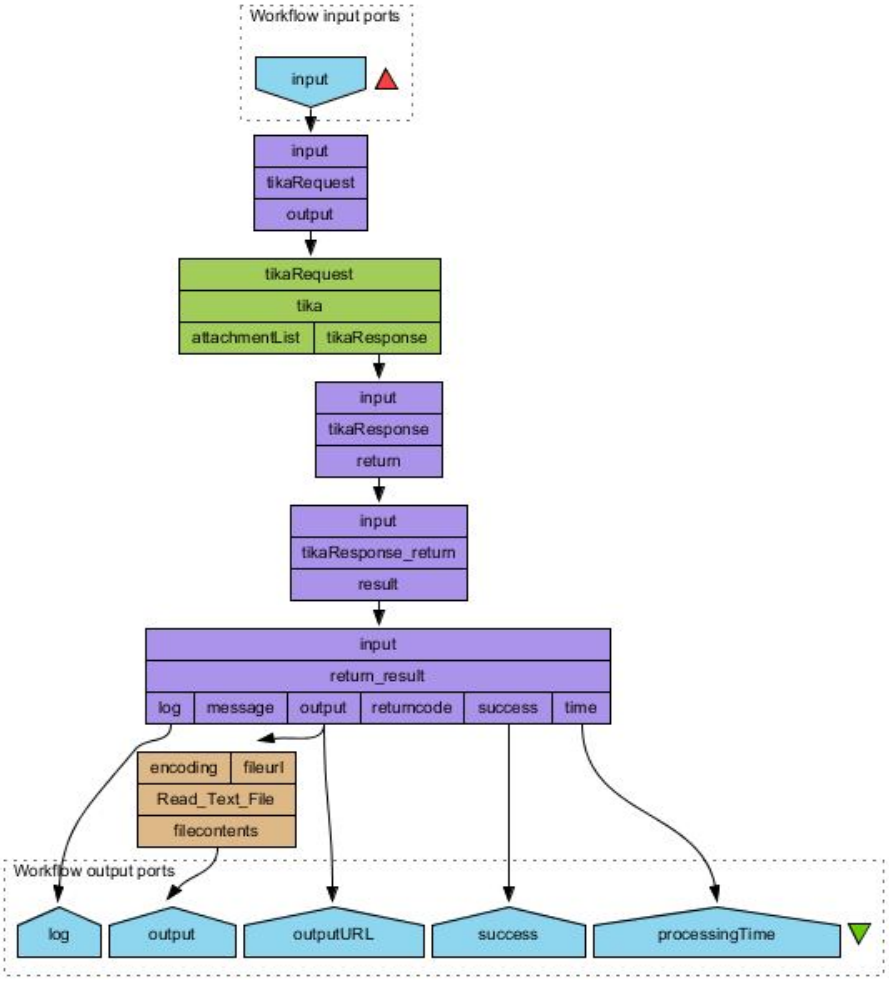

Figure 2: Basic Apache Tika™ workflow

In the Taverna Workbench, the green box named “tika” represents a web service using Apache Tika™ to extract metadata from a file. At the “outputURL” port the workflow returns a URL which can be use to access the results file. Alternatively, the result can accessed directly at the “output” port too (e.g. if you like to use the result as the input for another workflow).

Input port:

- File URL of the single file to be analyzed. Represented as [input] in Figure 2: Basic Apache Tika™ workflow.

Output ports:

- File URL of the result output XML file. [outputURL]
- The file content of the XML result output file. [output]

- Processing time in milliseconds (measured by the “ToolWrapper” code). [processingTime]
- The console output. [log]
- Boolean (true/false) success message. [success]

2.6 Choosing the right ARC unpacker

An important component for our experiments with ARC.GZ containers is the software for extracting the content of these containers into single files, which then can be analyzed.

The requirements for such an extraction tool are that it should be:

- able to run fully unattended (programmatic control)
- optimised in terms of performance
- reliable in its task of making all the content items available
- reliable in its error handling capability
- easy to setup (for later packaging and deployment)

Three tools have been evaluated against these requirements:

- fuse-j-2.4: https://github.com/blekinge/fuse-j-2.4-prerelease
- arc_extractor: https://wiki.lib.umn.edu/wupl/DI2.HowToCrawl/arc_extractor.txt
- ARC unpacker: https://github.com/openplanets/Arc-unpacker

2.6.1 fuse-j-2.4

This tool has a lot of software pre-requisites, depending even on specific versions of other software components. Additionally, mounting and un-mounting the archives turned out to be unreliable in terms of automation.

The fact that it is difficult to setup and that it is more or less impossible to integrate it into an unattended workflow in a reliable manner, led us to the conclusion that it is not the right tool for our approach.

2.6.2 arc_extractor

This Perl based tool is easy to setup. The tool worked well in preliminary testing, but when trying to integrate it into some basic workflows, the extraction process failed on some of the archived files due to long path names.

2.6.3 ARC unpacker

This project is based on libraries from Internet Archive's Heritrix web crawler and has been created by our SCAPE partners at the SB[49]. It is easy to setup and to deploy.

During testing we did not find any problems while unpacking container files with it and decided to compare it to the “arc_extractor”.

[49] http://en.statsbiblioteket.dk/

2.6.4 “arc_extractor” vs. “ARC unpacker”

In order to compare these two tools, some tests have been run with a set of real world ARC.GZ containers.

Test scenario 1:

- The “arc_extractor” is used to unpackaging the ARC files to the temporary file system.
- The main workflow components are connected through Taverna’s “run after” connections to run them in the correct order.

Test scenario 2:

- The “ARC unpacker” is used to unpackaging the ARC files to the temporary file system.
- The main workflow components are connected through Taverna’s “run after” connections to run them in the correct order.

Scenario1 and scenario2 are comparing the performance of the two tools - independent to the design of the workflow.

The conclusion is that “ARC unpacker” has a performance advantage of around 30% compared to “arc_extractor”.

In a third scenario, the design of the Taverna workflow was optimised.

Test scenario 3:

- The “ARC unpacker” is used to unpackaging the ARC files to the temporary filesystem.
- The main workflow components are connected through “standard” connection arrows to run them in the correct order. To achieve this, we need some additional “dummyIn” ports.

No data is passed through these ports. These ports are only used to define the correct workflow order.

This setup adds a performance gain of about 25% compared to scenario 2. This workflow design concept was therefore generally adopted for the experiments.

For a more detailed description about the implementation of scenario 3, please refer to the section 2.7.2.11 “Details about the component connections” in this document.

Because of the fact that the “ARC unpacker” proved to be good in performance, did not produce any errors during testing and is easy to setup and deploy, the tool was chosen for further experiments.

2.7 The experiments

The purpose of the experiments performed in this work package is to evaluate tools and work flow designs for web content analysis. A diagram showing the logical work flow can be found in section 2.2 Scope of the experiments, Figure 1: Conceptual workflow diagram.

The logical workflow is divided into three sections:

- Unpackaging
- Identification
- Publishing

Unpackaging:

This is the first step in the workflow. At this stage we need to extract the individual files of the ARC files and make them accessible for later identification. The files are extracted to a temporary storage system.

Identification:

As soon as the individual files from the ARC archives are available on the file system, the identification tool starts to run the recursive analysis over each object in the temporary folder structure. The tools involved at this stage are extracting metadata from each file or use other techniques to identify its file format (depending on the tool used). Detailed analysis on how each tool works has been done in work package PC.WP.1[50].

During this stage a series of reports are created; one per archive container. Each report contains the MIME type list for that container.

Publishing:

After the identification phase has finished, the results have to be published in an appropriate format. In the current stage of the project, publishing is done through summary reports in the CSV format. The summary report is an aggregation of all the single reports created in the identification stage. It contains a list of all MIME types found in all of the analysed web archive containers.

2.7.1 Workflow sequence overview

In the following workflow overview we briefly describe the steps performed during the runtime of the work flow illustrated in Figure 3: Workflow for unpacking analyzing ARC files.

Input of the workflow:

- A flat text file. Listing the ARC.GZ files to be analyzed.

Output of the workflow:

- One report file per ARC.GZ file, containing all MIME types plus a count on them in CSV format.
- One report file over all processed ARC.GZ files containing a normalized MIME type distribution list in CSV format.

The workflow exists in two versions:

- One using the TIFOWA tool utilizing the Apache Tika™ 1.0 API.
- One using DROID 6.01 in command line mode.

Rough workflow walkthrough:

- Read input list of ARC.GZ files
- Unpack each GZ to ARC

[50] Deliverable D9.1 of work package PC.WP.1.

- Unpack each ARC to a temporary folder (flagged with a "taskID[51]")
- Run the identification tool (Apache Tika™ or DROID)
- Cleanup the temporary files (per iteration)
- Wait for all iterations to be completed and generate a summary report over all partial reports

The summary output makes it easy to compare the results produced by the two different identification tools – if running on the same test data set.

All steps are running in parallel. Identification of one file is running while another file is being unpackaged and yet another is deleted from the temporary storage. Only the creation of the summary report is a last sequential step of the workflow.

Figure 3 illustrates the above described workflow. It has been published to the SCAPE group on the myExperiment platform for sharing with the rest of the SCAPE project.

It is available on the following URL: http://www.myexperiment.org/workflows/2690.html

[51] We need to separate the unpacked web content from container A from the unpacked content of container B. Therefore we need to create something like session based temporary directories. One for each ARC.GZ in the input file list.

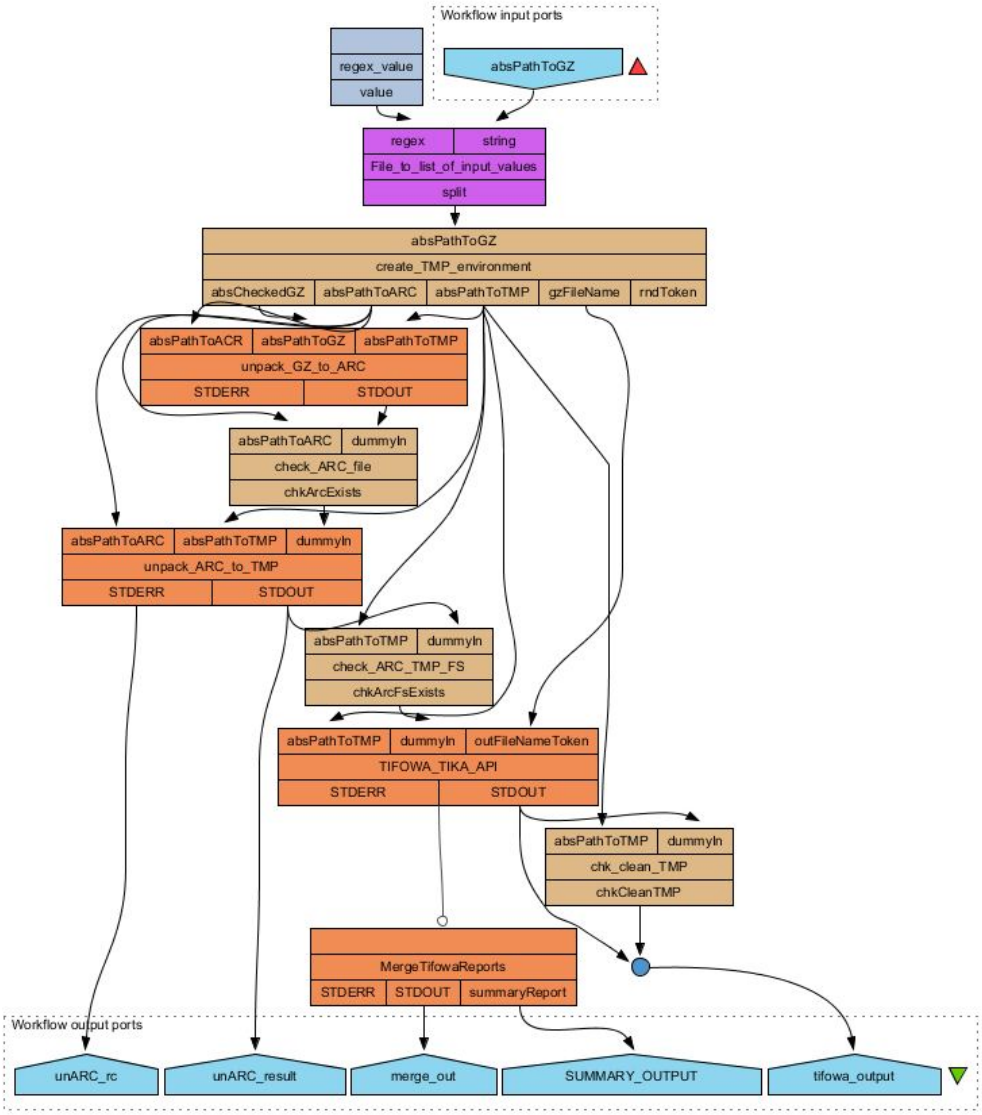

Figure 3: Workflow for unpacking analyzing ARC files

A detailed description how to setup the workflow can be found in section 2.7.5 Workflow setup guide.

2.7.2 Implementation details

Due to the large amount of local data to be processed, the Taverna “tool service” is used as integration mechanism instead of web services.

In the following section we will run through the entire workflow in detail and describe what is going on while processing the different sections of the workflow.

2.7.2.1 Input

Component name (in Taverna Workbench represented as workflow “boxes”):

absPathToGZ

regex_value

File_to_list_of_input_values

In this phase of the workflow, we prepare a list of ARC.GZ files to be processed by the workflow. The list is a flat text file containing the full absolute path to one container file per line.

While working with the experimental implementations, it has turned out that it is very practical to have different “compilations” of these lists. For example to have a list of containers consisting of very large content files, a list of containers which have turned out to be problematic in terms of unpacking for some reasons, a list of small containers for demonstration, and so forth.

We decided to go in the direction of compiling the different sets in separate text input files and read the files on demand for the particular experiment. This allows very quick switching between different sets.

To transform the list of files into a Taverna workflow data structure, we used the Taverna local service “Split_string_into_string_list_by_regular_expression” with the regular expression “\n” (read the file line by line). In the workflow this is the box named “File_to_list_of_input_values”.

The Taverna input port “absPathToGZ” is, as described above, expecting a flat text file.

COMMENT: In Taverna Workbench add your input file by clicking "Add file location..." (NOT "Add value").

2.7.2.2 Prepare the workflow run time environment

Component name: **create_TMP_environment**

Before the final summary report can be created, we need the single reports of the content in each of the ARG.GZ containers.

To achieve this, we need to be able to pass a large list of container files to the workflow and process it fully unattended. This requires running the workflow components in parallel. Which means that the later file-identification stage of the workflow is running in parallel to the unpacking process.

Due to this requirement we need to separate the unpacked web content from container A from the unpacked content of container B. So we need to create something like session based temporary directories. One for each ARC.GZ in the input file list.

To ensure unique temporary file paths, we created the “create_TMP_environment” component. It has been implemented as a Taverna beanshell. This component prepares a set of variables for later use in the workflow:

- rndToken … a random token value (per iteration)
- absPathToGZ … absolute path to the original ARC.GZ file
- absPathToARC … absolute path to the unpacked ARC file
- absPathToTMP … absolute path to the users temporary directory including the random token part “rndToken”

- gzFileName … the original file name of the ARC.GZ file

2.7.2.3 Unpack ARC.GZ

Component name: **unpack_GZ_to_ARC**

This component, which has been implemented as a Taverna "tool" service, has two tasks to do.

Firstly, it uses the *absPathToTMP* output from the previous step to create a temporary directory beneath the users TMP folder; one per workflow iteration (per processed ARC.GZ file).

Taverna itself provides a similar functionality for temporary directories. But we decided to use our custom directory to have full control over cleanup. This will, for example, make it easier to join the workflow later with other work flows.

Secondly, it unpackages the ARC.GZ file stored at *absPathToARC* to an ARC file stored at *absPathToARC*.

2.7.2.4 Condition checking - the ARC file

Component name: **check_ARC_file**

The workflow uses three components for checking the condition of workflow:

- check_ARC_file
- check_ARC_TMP_FS
- chk_clean_TMP

These components have been implemented as Taverna beanshell scripts and are used to check prerequisites for the subsequent components.

The component named *check_ARC_file* checks for the existence of the unpacked ARC file at *absPathToARC*. If the file is there, unpacking has been finished successfully. This implementation has been chosen intentionally because it offers an advantage compared to simply checking the error level directly in the previous tool service and exiting with error code 1. The advantage of doing it in a beanshell script is that we can throw a specific exception with a meaningful description.

The check_ARC_TMP_FS and chk_clean_TMP components will be described in later sections.

2.7.2.5 Unpack ARC

Component name: **unpack_ARC_to_TMP**

As mentioned previously we use the arc-unpacker for extracting the files from the ARC files.

ARC unpacker … https://github.com/openplanets/Arc-unpacker

Triggered from a Taverna tool service, the tool unpacks the ARC file (absPathToARC) to its dedicated, temporary session folder at absPathToTMP/content/.

2.7.2.6 Condition checking - the content folder

Component name: **check_ARC_TMP_FS**

This component is part of our previously described exception handling. It follows the disassembling of the ARC file and checks for the existence of the temporary session folder. If it is not there, it throws an appropriate exception.

2.7.2.7 Run identification tool and create single reports

Component name: **TIFOWA_TIKA_API** (Apache Tika™ version of the workflow)

This step is the primary step of the workflow. This is where the actual identification takes place.

For this purpose the SCAPE tool named "TIFOWA" has been integrated into the workflow.

TIFOWA is a tool wrapping the Apache Tika™ API for file identification and is also caring about running through a particular folder structure recursively and creating the output. A detailed description of the tool is a available in section 2.7.3 Tools.

TIFOWA is started from a Taverna tool service as a java jar file. The working directory is passed through the absPathToTMP variable and the output will go to the directory:

~/scanARC/outputCSV/

The output file contains a section storing all the mime types found in the archive container, plus a count on the occurrences of each found MIME type. This is what this document often calls "the single results". It is the identification result of a single ARC.GZ container.

In an additional step, the temporary session folder containing the content for this iteration will be removed from the temporary session directory. This step could be optionally skipped by un-commenting the appropriate command in this component to preserve the unpacked files for further processing in a joined or extended workflow.

Component name: **DROID_CMD** (only available in the DROID version of the workflow)

The workflow is available in two flavours, which are pretty much identical - except exactly this step of the workflow.

Compared to TIFOWA, the tool needs to be called twice to create the result for the identification step. The first tool run adds a set of files or folders to a "DROID profile". The second run starts analyzing the files and creates the DROID report file.

The output format is naturally completely different compared to the output from TIFOWA. But to keep the other parts of the workflow identical to the TIFOWA version of the workflow, an additional step needs to be performed.

A java tool has been created that can be called as a java jar file from a Taverna tool service. The tool called csv2Tifowa (2.7.3.2 csv2Tifowa) is accepting a DROID CSV file as input and is returning the results in the same format as TIFOWA does.

After converting the output format, the temporary session folder containing the content for this run time iteration will be removed from the temporary directory to free disk space.

2.7.2.8 Condition checking - cleanup

Component name: **chk_clean_TMP**

This component is part of the exception handling aspect of the workflow.

At the end of the previous step, the temporary session folder for each iteration is getting deleted. This beanshell component checks the success of the previous cleanup step and throws an appropriate exception on failure.

2.7.2.9 Create combined overall report

Component name: **MergeTifowaReports**

Since we are caring about the output formats returned by the two different versions of the workflow in the component **DROID_CMD**, the merging process for both versions can be identical.

The main task of the Taverna tool service MergeTifowaReports is to collect all the single reports from the previous step and merge the results into one overall report file. It contains a list of MIME types found in all processed ARC.GZ containers. The component uses the mergeTifowaReports tool which is described in detail in the section 2.7.3 Tools.

Additional steps done here:

- We want to publish the content of the results file on a Taverna output port. Because Taverna "File outputs" expect files in Tavernas default task directory, we need to copy the results file from "~/scanARC/outputCSV/fullTIKAReport.csv" to "/tmp/usecaseXYZ/".
- Use mergeTifowaReports again to create a report in a format required for the graphical representation output e.g. at ~/scanARC/outputGraphics/graphicsTIKA/.
- Move single report files to the ~/scanARC/outputCSV/oldSingleReports/ folder to free the space for the next workflow run.

2.7.2.10 Output

After the workflow has finished processing all iterations, the produced results can be picked up in several formats.

- Taverna output Ports
- Single CSV files (one per processed web archive container)
- Overall CSV file containing a summary report over all processed ARC files
- Graphical representation of the overall result

A more detailed description of the different output formats follows.

Taverna output Ports

- unARC_rc … mainly for trouble shooting (interim tool result)
- unARC_result … mainly for trouble shooting (interim tool result)
- merge_out … mainly for trouble shooting (interim tool result)
- tifowa_output … mainly for trouble shooting (interim tool result)
- SUMMARY_OUTPUT … The content from the overall report file made available on a Taverna output port.

The SUMMARY_OUTPUT output port holds the content of the final overall report file. It could for example be used to connect the workflow with subsequent workflows or just for quickly viewing the result without going to the file system.

Single CSV files

~/scanARC/outputCSV/oldSingleReports/

Contains all single report files for later reference. This file can easily be imported into a spreadsheet program. When opening the CSV file in a spreadsheed application, the “space” character must be used as the separator.

Overall CSV

~/scanARC/outputCSV/

Contains the overall result files.

TYPE	COUNT
image/jpeg	3531
text/html	1815
application/xhtml+xml	1806
image/gif	1025
text/plain	424
image/png	296
application/rss+xml	79
application/x-shockwave-flash	36
application/xml	35
application/pdf	31
image/x-icon	15
audio/mpeg	4
audio/x-ms-wma	2
audio/x-wav	1
application/vnd.ms-excel	1
video/x-ms-wmv	1
audio/midi	1
application/x-rar-compressed	1
TOTAL	**9104**

Figure 4: Sample workflow output

Graphical representation

To get a better impression about the MIME type distribution, the workflow is also producing a graphical representation of the overall content result report.

~/scanARC/outputGraphics

This folder contains a sub folder for each workflow version. Within that folder resides a static HTML file and a data.js file created by the workflow.

Example:

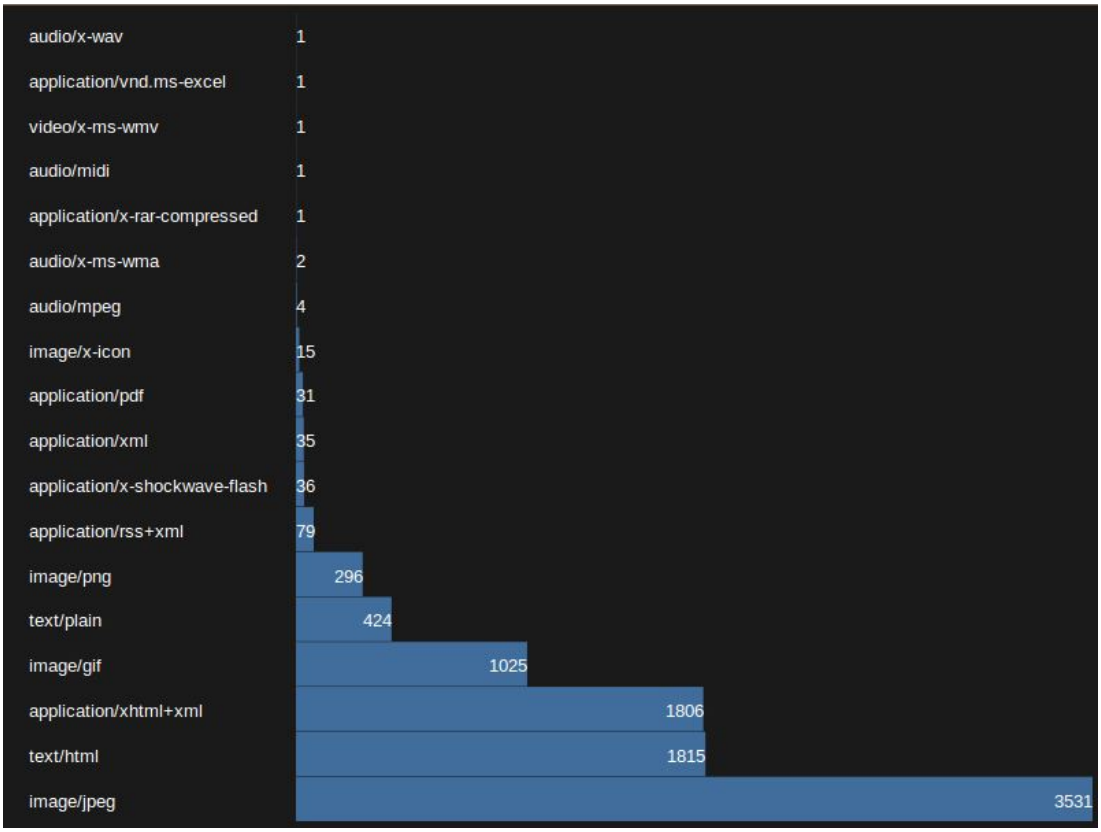

Figure 5: Graphical representation of a sample workflow output

2.7.2.11 Details about the component connections

While implementing the designed workflows, we made some general observations.

The design of the workflow depends on the type of the underlying component, the error handling that is planned, and the way the components should be connected. The involved components have already been described and gave some insight into the implementation of error handling in section 2.7.2.4 Condition checking - the ARC file.

The remaining part of how to connect the components in an effective way has a very strong impact on both, the overall workflow design and the performance of the workflow. It might also influence the required infrastructure in terms of storage space.

If the workflow is completely linear in the sense that it seems appropriate to process one step after the other, Taverna would also be able to run multiple iterations in parallel.

An obvious approach would be to make use of Taverna's "run after"connectors. This assumption might be fostered by components which do not need to pass data from the output port from one component to the input port of the other component - but instead are based on writing files to the file system. Like in our case in which we are dealing with components unpacking files to the file system and creating report files on the file system. For that reason it seems to be the best way to run a component one after the other. But only at the first look.

If - in the described scenario - two workflow components are connected using a "run after" connector, the second component will wait for the first component to fully complete all of its

iterations (one iteration per ARC.GZ to be processed). In our workflow this means, that the remaining part of the workflow waits for the "unpack_GZ_to_ARC" to complete. The following component will start to unARC the ARC files to the temporary file system as soon as the complete list of GZ files has been unpacked.

The same is true for all the other steps like deleting the temporary files. This might be a time consuming task on a huge amount of small files.

As an alternative, it is possible to use standard connectors in combination with some "dummy input ports". The following components can then start as soon as the previous components finish the current iteration. "dummy input ports" in this context means that no data is passed through these ports. Keep in mind that these ports are only used to define the correct workflow order. Practically this has the effect, that all of the main components are running simultaneously during a workflow run. e.g.: The content of ARC10 is being identified while ARC30 is being unpacked and ARC01 has already been deleted from the temporary file system.

An additional advantage is that it is not necessary to allocate the complete disk space required for processing x ARC.GZ files during one workflow run because older (already finished) iterations have already de-allocated disk space while new iterations are just starting.

During our experiments it turned out that this alternative implementation method adds a performance gain of about 25% compared to a strict structural implementation using the more obvious “run after” connectors. Therefore, we took that in consideration when we designed our experiments and used the "dummy input ports" approach.

2.7.3 Tools

The following tools have been developed by the SCAPE project to fulfil some tasks required for the described workflow experiments.

2.7.3.1 tifowa

TIFOWA is currently (March 2012) using the Apache Tika™ 1.0 API to extract metadata from the files contained in a particular folder structure. It creates a list of all identified MIME types with the total number of occurrences and presents the output in a format which can be easily imported as CSV data for further processing in a spreadsheet program. Embedded in the described Taverna workflow, the result is a bunch of files containing the MIME type distribution list for each ARC file.

By letting the tifowa tool handle the iteration over the files and folders recursively (extracted from the archive containers), we have been able to avoid the large overhead of starting the JVM for each file - compared to the Apache Tika™ command line usage mode in a script.

Usage:

```
java -jar tifowa.jar -d [DirectoryToBeScanned] > [OutputFile]
```

Source code:

https://github.com/openplanets/scape/tree/master/tb-wc-tifowa

2.7.3.2 csv2Tifowa

The csv2Tifowa tool is only used in the DROID version of the workflow.

The output of DROID is completely different compared to the output of TIFOWA. To keep the other parts of the workflow identical to the TIFOWA version of the workflow, an additional step needs to be performed to convert the output format.

The csv2Tifowa tool converts DROID CSV reports into an output format similar to the output produced by tifowa.

Usage:

```
java -jar csv2Tifowa.jar [DroidCSVFile] > [TifowaCompatibleOutputFile]
```

Source code:

https://github.com/openplanets/scape/tree/master/tb-wc-csv2tifowa

2.7.3.3 mergeTifowaReports

MergeTifowaReports has been created to merge the single reports produced by tifowa or csv2Tifowa (in the DROID version of the workflow) into an overall summary report file.

During the run time of the Taverna component “MergeTifowaReports ” the tool is called twice. First to create the actual overall report in CSV format and secondly to create a report in Javascript/JSON format which is needed as input for the graphical representation (see: "Workflow Setup - more details") of the report.

Usage:

```
java -jar mergeTifowaReports.jar [ReportDirectory] [OutputFile] [FormatOption]
ReportDirectory     …   the directory containing the single tifowa report files
OutputFile          …   the output file
FormatOption        …   xls | csv | js
xls     …   currently not used by the workflow
csv     …   this is the output the workflow uses for the full report
js      …   this format is used as input for the graphical representation
```

Source code:

https://github.com/openplanets/scape/tree/master/tb-wc-merge-tifowa-reports

2.7.4 Some additional observations

The main focus of this work package in the current stage is to design executable experimental preservation workflows. The key points in regards of workflow design and the reliable, unattended execution for our experiments have already been described in the previous sections.

As a requirement for a more detailed file analysis, we need to know the content we will have to deal with at a later stage of this work package. A possible outcome might be that we need more detailed PDF identification capabilities. For instance the PDF version or type distribution in a particular web archive might be interesting. For that reason we have performed a series of experiments with

Apache Tika™ and DROID to roughly compare the results in terms of coverage, performance and stability based on a test data set of real world web archive containers[52].

2.7.4.1 Typical output

Figure 6 shows a typical output produced by the described workflow using the Apache Tika™ API, the different colours highlighting different content types:

TYPE	COUNT	PERCENTAGE
text/html	2655	45,06%
image/jpeg	1638	27,80%
image/gif	702	11,91%
text/plain	427	7,25%
image/png	200	3,39%
application/xhtml+xml	141	2,39%
application/rss+xml	43	0,73%
application/pdf	36	0,61%
application/xml	15	0,25%
application/x-shockwave-flash	15	0,25%
application/octet-stream	8	0,14%
image/x-icon	3	0,05%
audio/mpeg	2	0,03%
video/quicktime	1	0,02%
image/bmp	1	0,02%
audio/x-wav	1	0,02%
application/x-tika-ooxml	1	0,02%
application/x-gzip	1	0,02%
application/vnd.ms-powerpoint	1	0,02%
application/vnd.ms-excel	1	0,02%
TOTAL	**5892**	

Figure 6: Sample workflow output with grouping

An interesting observation was that each analyzed archive (containing a set of completely different sites within the AT domain) follows roughly the following (%) distribution on the file type count:

[52] http://wiki.opf-labs.org/display/SP/Austrian+National+Library+-+Web+Archive

	PERCENTAGE
Text	52,31%
Image	43,18%
Application	4,45%
Video	0,02%
Audio	0,05%

Figure 7: Colouring of content type groups. Values in percent.

2.7.4.2 Output differences between Apache Tika™ and DROID

On our test data sample we did see some important differences between the output of the two different workflow versions using Apache Tika™ and DROID:

TIFOWA (Apache Tika™ 1.0 API):

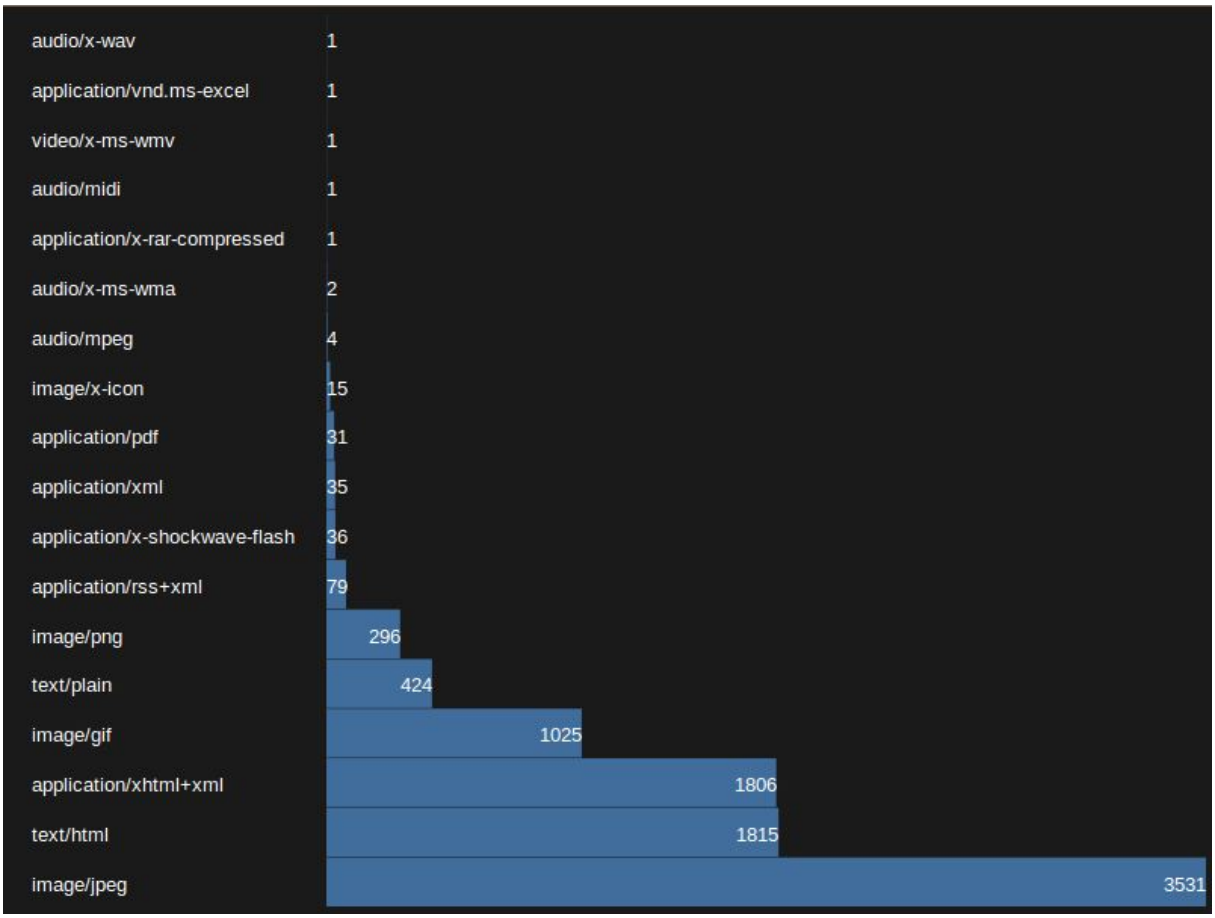

Figure 8: Apache Tika™ 1.0 API sample output

DROID 6.01:

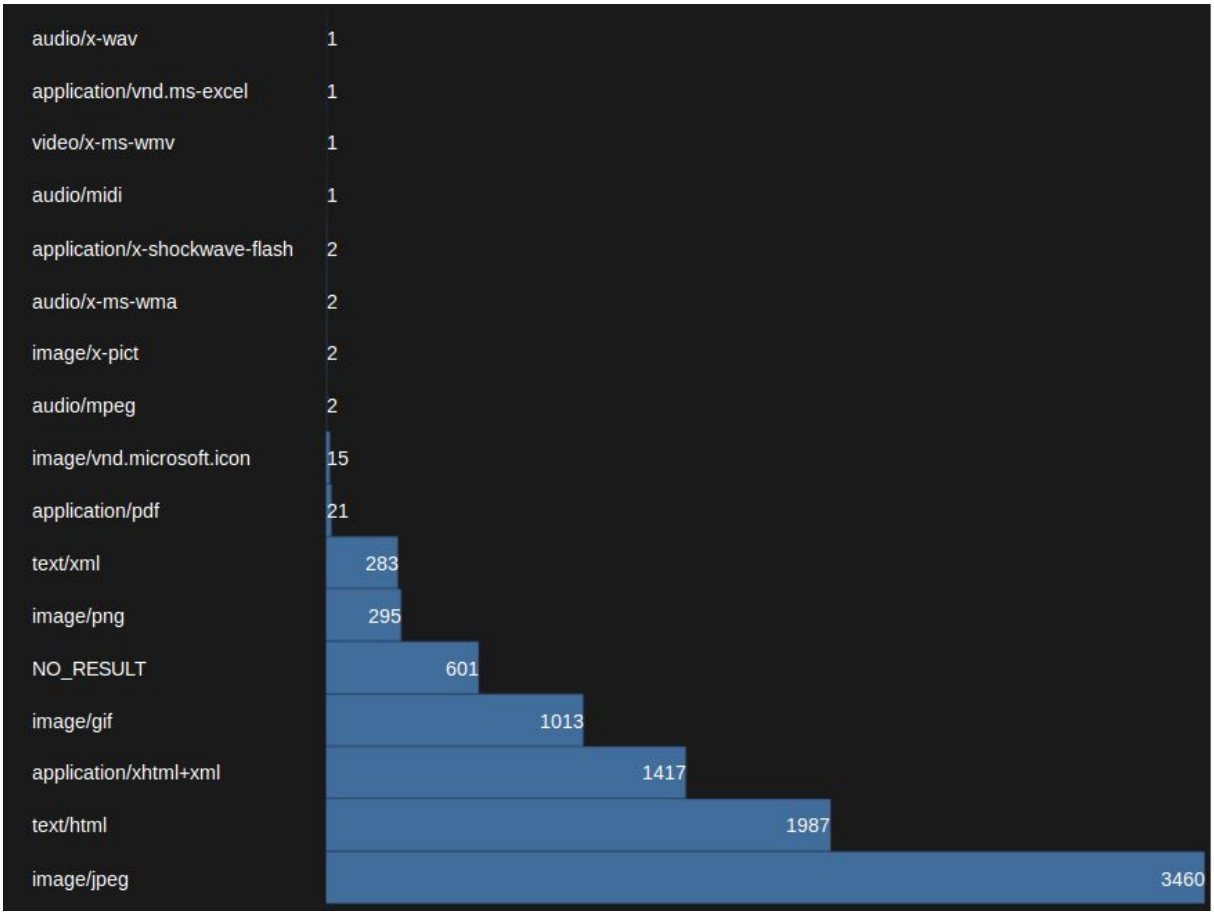

Figure 9: DROID 6.01 sample output

As shown in Figure 9, DROID marks a lot of files as unknown (NO_RESULT). On the small test data sample described here (9104 files in total), DROID marks 601 files as unknown - while Apache Tika™ detects all of them.

It is important to note here that the data set is not an annotated corpus, and therefore we are not able to say if all the formats detected by Apache Tika™ (and also DROID) are correct or not.

But the assumptions regarding the file format coverage of both tools have been confirmed by the detailed results of work package PC.WP.1[53] - based on an annotated corpus (Govdocs1 digital Corpora[54]).

In regards of the MIME-type naming, Figure 8 and Figure 9 show that there are slight differences between the tools (e.g.: image/x-pict, image/x-image). To exactly compare characterization tools in an automated manner, PC.WP.1 has developed “The SCAPE Characterisation Tool Testing Suite” which uses a MIME- type equivalent list (a mapping list). Please refer to [D9.1] for further details.

[53] Deliverable D9.1 of work package PC.WP.1
[54] http://digitalcorpora.org/corpora/files

2.7.4.3 Rough performance estimation

It is not the main objective of this work package to do precise performance measurement on identification tools. This topic has also been covered in detail by work package PC.WP.1[55].

However, since we were experimenting with workflows utilizing identification tools, we did observe some performance tendencies.

Processing of the same set of input ARC.GZ files (10 ARC GZ files, 100MB each), takes:

12.4 minutes using the TIFOWA version

15.0 minutes using the DROID version

These results are very rough values provided by the Taverna workbench - but they do give a sense of what throughput to expect in a particular test environment.

TIFOWA (Apache Tika™ API):

Name	Status	Queued itera...	Iterations done	Iterations w/e...	Average time...	First iteration...	Last iteration...
Workflow3	Finished	-	-	-	12.4 m	12:59:12	13:11:38
check_ARC_file	Finished	0	10	0	16 ms	12:59:21	13:01:06
check_ARC_TMP_FS	Finished	0	10	0	21 ms	12:59:30	13:01:07
chk_clean_TMP	Finished	0	10	0	18 ms	13:02:28	13:11:37
create_TMP_environment	Finished	0	10	0	26 ms	12:59:16	12:59:18
TIKA	Finished	0	10	0	1.2 m	12:59:31	13:11:37
File_to_list_of_input_values	Finished	0	1	0	80 ms	12:59:15	12:59:15
MergeTifowaReports	Finished	0	1	0	320 ms	13:11:38	13:11:38
regex_value - \n	Finished	0	1	0	21 ms	12:59:15	12:59:15
unpack_ARC_to_TMP	Finished	0	10	0	6.7 s	12:59:21	13:01:07
unpack_GZ_to_ARC	Finished	0	10	0	10.3 s	12:59:17	13:01:06

Figure 10: TIFOWA - rough performance

Note: average time for the Apache Tika™ (API used in a wrapping java project called “tifowa”) step per iteration: 1,2 Minutes

DROID 6.01:

Name	Status	Queued itera...	Iterations do...	Iterations w/e...	Average time/...	First iteration...	Last iteration ...
Workflow3	Finished	-	-	-	15.0 m	12:11:04	12:26:02
check_ARC_file	Finished	0	10	0	13 ms	12:11:12	12:13:49
check_ARC_TMP_FS	Finished	0	10	0	35 ms	12:11:23	12:13:52
chk_clean_TMP	Finished	0	10	0	17 ms	12:15:30	12:26:02
create_TMP_environment	Finished	0	10	0	24 ms	12:11:08	12:11:09
File_to_list_of_input_values	Finished	0	1	0	53 ms	12:11:08	12:11:08
MergeDROIDReports	Finished	0	1	0	336 ms	12:26:02	12:26:02
regex_value - \n	Finished	0	1	0	35 ms	12:11:07	12:11:07
run_DROID	Finished	0	10	0	1.5 m	12:11:24	12:26:02
unpack_ARC_to_TMP	Finished	0	10	0	6.8 s	12:11:13	12:13:52
unpack_GZ_to_ARC	Finished	0	10	0	15.3 s	12:11:08	12:13:48

Figure 11: DROID - rough performance

Note: average time for the DROID (3 step approach = 1) create droid profile 2) analyze files 3) prepare output) step per iteration: 1.5 Minutes

Our rough observations on the performance tendencies are support by the detailed results from work package PC.WP.1.

[55] Deliverable D9.1 of work package PC.WP.1

2.7.5 Workflow setup guide

This section describes how to setup the experimental web content identification workflow for demonstration purposes within the SCAPE project.

The workflow and the setup guide have been tested with:

- Ubuntu Linux 11.10 (32bit)
- JAVA version jdk1.6.0_29

This section only describes the Taverna workflow using tifowa - but the DROID version of the workflow is also included in the package.

SCAPE project members can download the required workflow package from:

http://fue.onb.ac.at/scape/dist/scanARC_0.2.tar.gz

The workflow exists in two versions:

- TIFOWA tool utilizing the Apache Tika™ 1.0 API.
- DROID 6.0.1 in command line mode

2.7.5.1 Preconditions

Ensure that your machine has a suitable JDK installed.

Verify that these variables are set correctly using a terminal window:

echo JDK_HOME

echo $JAVA_HOME$

Make sure you have added the appropriate Java bin path to your environment variables:

"java -version" from the command line should give you back the installed version.

Make sure the java environment variables are available for Taverna too - not only for your terminal window. Under Ubuntu, this can be achieved by setting the environment in **/etc/environment**.

example /etc/environment:

PATH=**"/path2JAVA/jdk1.6.0_29/bin:**/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"

JAVA_HOME="**/path2JAVA /jdk1.6.0_29**"

JDK_HOME="**/path2JAVA /jdk1.6.0_29**"

Setup Taverna Workbench 2.3.0 according to the setup instructions[56].

Remember to install "Graphviz". This is a prerequisite for Taverna Workbench on Linux (sudo apt-get install graphviz).

2.7.5.2 Quick start

- Make sure you meet the preconditions above

[56] http://www.taverna.org.uk/download/workbench/2-3/

- Unpack the scanARC.tar.gz archive to your home folder (scanARC in the root of your home)
- Copy your ARC.GZ files to ~/scanARC/input/ARCs/
- Prepare an input file e.g. ~/scanARC/input/myInput.txt (list of paths to ARC.GZ)
- Start Taverna Workbench
- Open the Apache Tika™ workflow: ~/scanARC/workflow/scanARC_TIKA.t2flow
- Click PLAY in Taverna Workbench
- Add your input file by clicking "**Add file location...**" (NOT "Add value")
- Run the workflow
- Wait until the workflow has finished
- Pick up your full result at ~/scanARC/outputCSV/fullTIKAReport.csv (Remember that the field seperator is the space character.)
- or get the full result on the workflow output port "SUMMARY_OUTPUT "
- Open ~/scanARC/outputGraphics/graphicsTIKA/tika-identification.html with your browser

If you would like to have more details on the setup or technical details, switch to the next section.

If your workflow fails, check:

- Does "java -version" in a terminal window give you the correct version info?
- Did you set JAVA_HOME **and** JDK_HOME correctly?
- Does Taverna pick up your environment variables (see "preconditions" above)?
- Does your input file contain full absolute paths (e.g.: /home/user/scanARC/input/ARCs/test1.arc.gz)?

2.7.5.3 Workflow Setup - more details

Unpack the scanARC.tar.gz archive to your home folder. After unpacking a folder named "scanARC " is sitting in the root of your home directory.

Description of the sub folders:

input sub folder

This sub folder contains the list of ARC.GZ files to be processed by the workflow.

Copy your ARC.GZ container files to the ~/ scanARC/input/ARCs/ folder.

After that, create a text file containing a list of ARC.GZ files you want to be processed (including the absolute path) in the folder ~/ scanARC/input/.

Please have a look at the example file: nonWorkingSampleList.txt

In Taverna Workbench you will later pick up that file on the RUN dialog by clicking "Add file location".

The workflow will then read the input file line by line and process all the contained files.

outputCSV sub folder

This is the place where your combined CSV result will be copied to. The combined full result will be named fullTIKAReport.csv. It is the accumulated result from all of the singe results (one for each ARC).

During the runtime of the workflow, the single results will be stored there too. As soon as processing has finished, the single results will be copied from ~ /scanARC/outputCSV/ to ~/scanARC/outputCSV/oldSingleReports/ to clean up for the next run and to keep the old single file results.

If you are interested in the single file results, you will find them in the folder ~/scanARC/outputCSV/oldSingleReports/.

outputGraphics sub folder

The graphical presentation is prepared here.

During the merge of all the single file results into the combined full result file, a Javascript/JSON file called data.js will also be generated.

After processing, you can find the Javascript/JSON file at ~/scanARC/outputGraphics/graphicsTIKA/data.js.

To view the graphical presentation, simply open the tika-identification.html file with your browser of choice.

tools sub folder

The tools folder contains sub folders for local tools needed for the workflow.

The **tifowa, tifowaMerge, unARC** folders are pre populated with tools written by the SCAPE project for the SCAPE project. All the tools are ready to run. Details about these tools can be found in the section 2.7.3 Tools.

The **droid-6.01** folder is empty and a placeholder if you like to run the optional scanARC_DROID.t2flow workflow.

In order to work with the DROID based workflow, you need to get DROID 6.01 from the official web site[57] and install the package to: ~/scanARC/tools/droid-6.01/

HINT: The path to **droid-command-line-6.0.jar** should then be ~/scanARC/tools/droid-6.01/droid-command-line-6.0.jar (but you still need the entire package).

workflow sub folder

This folder contains the two actual workflows that both opens in Taverna Workbench.

scanARC_TIKA.t2flow is the primary Taverna workflow.

scanARC_DROID.t2flow is the secondary Taverna workflow.

If you like to run this workflow, please install the DROID tool first at ~/scanARC/tools/droid-6.01/.

[57] http://www.nationalarchives.gov.uk/

3 Solution SO18: Comparing two web page versions for web archiving

This solution SO18[58] section, describing Quality Assurance Components, is primarily based on input from [D11.1][59].

3.1 Introduction

As outlined in the first section of the report, this second scenario aims at building automated tools and methodologies to improve and maintain quality of Web Archives over time.

As shown in the characterization scenario, Web Archives are “black boxes” containing all sorts of resources that can currently not be characterized or catalogued in the librarian and archivist traditional perspective. Not only the characterization of Web Content at an obvious large scale is a complex and crucial issue but the way we gather and provide access to this content rely on tools that can never ensure completeness. Indeed, parsing crawlers used my most Web Archives to capture Web Content encounter technical and size issues that cannot yet be overcome.

We will describe in detail the current existing workflows and web archiving tools and then expose the solutions and tests made in the first part of this project. We will then outline our planned future tasks to tackle the scalability issue for this specific scenario.

3.2 Current status

3.2.1 Context

Monitoring Web Archives quality is a complex task and the quality of Web Archives themselves depends on several complex processes and tools.

In the case of the Internet Memory Foundation, the Web Archive contains around 200TB following the latest statistics and is growing rapidly. The Web Archive consists of two types of crawls, the large or domain crawls and the selective crawls.

Quality for large crawls such as for an .fr domain or a .eu domain crawl can only be assessed through the use of metrics. The idea is to compare crawl after crawl a number of key metrics such as the size of the crawl itself, for example.

For selective crawls, we still can use metrics comparison to outline a “big” possible issue after a crawl, such metrics are the size, the number of URLs crawled or even the number of PDFs or images found. Obviously, in the latter example, we rely mainly on Web Servers answers stored during the crawl. Used to compare recurrent crawls, this method provides good results, as spotting a failed or problematic crawl is quite straightforward. It is also possible to get information such as the number of videos or images from the live site or the webmaster and then compare it with crawl results.

Another way of checking the quality of selective crawls (site level) is to make a visual comparison at the page level. We basically select a number of pages to test visually and compare them to the live version. This control can be made following various methodologies. In the case of IM, we provide a quality assurance depending on the contract and institutions needs. The control follows a

[58] http://wiki.opf-labs.org/display/SP/SO18+Comparing+two+web+page+versions+for+web+archiving
[59] Deliverable D11.1 of work package PC.WP.3

methodology we defined internally, which is a mix between systematic checking up to a defined number of level down from the homepage but also a detection of technological areas that might not have been captured correctly by the crawler. This methodology obviously requires a good knowledge of both the crawler and the access tool used, as quality issues can be related to a lack of capture but also to a rendering issue due to our tool.

The "manual" process has several disadvantages:

- It is time consuming and requires to train a QA team
- It is limited and can only apply to small amount of content
- It requires to be made as soon as possible after the crawl ends when content is identified as missing and needs recapturing.
- It can hardly be applied to a regular monitoring of the archive for long-term preservation purposes.

For all these reasons, developing automated QA is crucial.

Some experiments are made using tools like Selenium[60] adapted to mimic a QA assessor's clicks on Web pages to identify missing content, but this solution does not cover the rendering issues we also proposed to tackle here.

3.2.2 Tools and Workflow

The solution developed by UPMC[61] to answer the QA issue described, includes a visual and a structural comparison of Web pages.

Both comparisons are made through the use of a screenshot of a web page, which will be the input of the workflow. The current workflow relies on the use of an adapted version of VIPS (Vision-based Page Segmentation Algorithm)[62], a non-open source tool, which will soon be replaced by an open source solution entirely developed by UPMC's team.

In addition to the fact that it is a proprietary solution, VIPS[63] has the inconvenience of being compatible with Internet Explorer only. This is obviously problematic in our case, as the preservation scenario includes a test on several browsers and browsers' versions to detect rendering issues. Our near future work will therefore include testing the open source solution and choosing and adapting a screenshot tool that will fit our needs. Still, the workflow should remain very close to the one described below. It consists of three main steps:

- Conversion
- Feature extraction
- Similarity calculation

[60] http://seleniumhq.org/
[61] http://www.upmc.fr/
[62] http://research.microsoft.com/apps/pubs/default.aspx?id=70027
[63] D. Cai. S. Yu. J.R. Wen. W.Y. Ma. VIPS: a Vision-based Page Segmentation Algorithm. Nov. 1, 2003. Technical Report. MSR-TR-2003-79

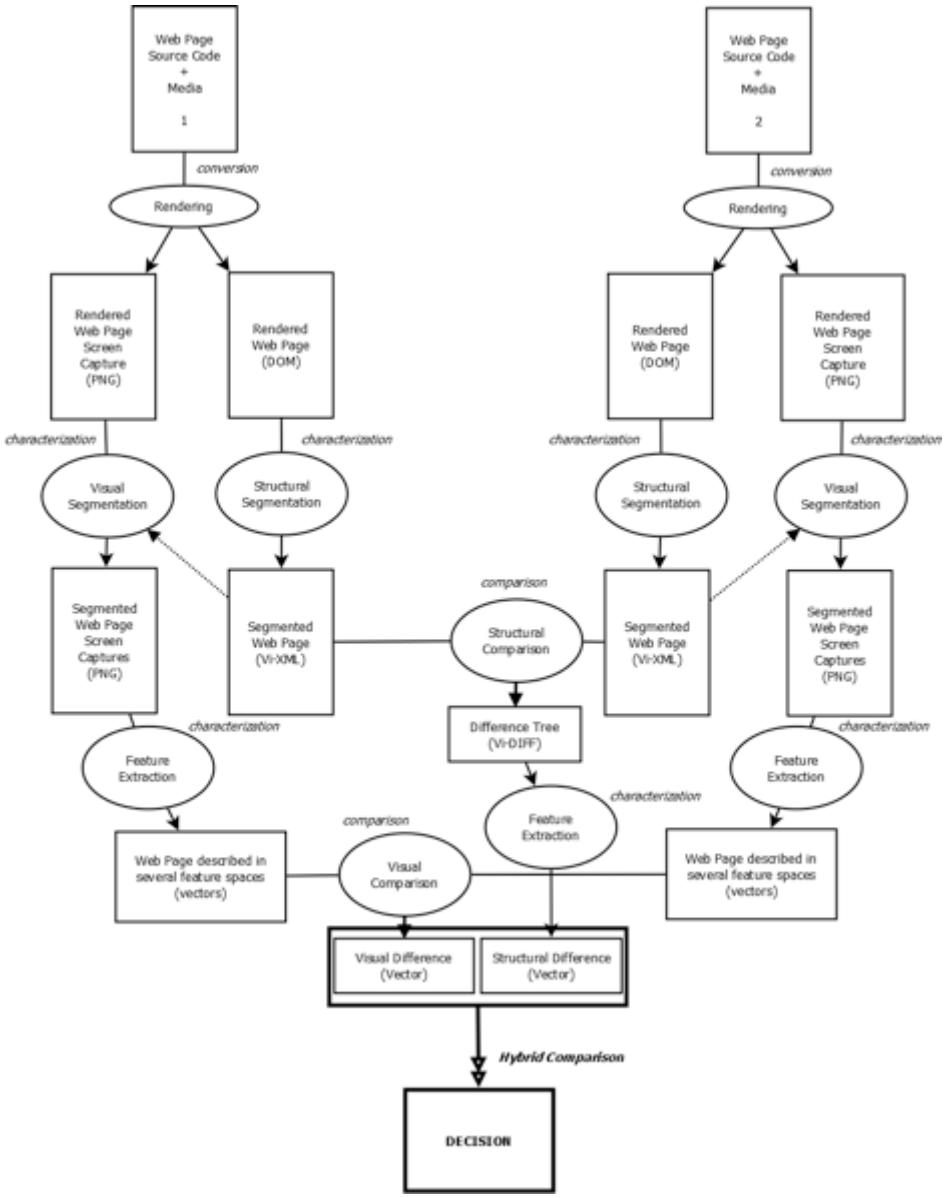

Figure 12: Visual and structural comparison workflow diagram

3.2.2.1 Conversion step

In our current experiment, this process is handled by VIPS. As explained above, VIPS tolerates only the use of Internet Explorer. VIPS segments the web page and produces an XML tree of the selected page. As outlined above, the UPMC team uses an adapted version of VIPS, developed by Myriam Ben Saad, (2009)[64], which produces XML trees named Vi-XML in our schema.

[64] M. Ben Saad, S. Gancarski, and Z. Pehlivan. « A novel web, archiving approach based on visual pages analysis ». In IWAW 2009.

The segmentation process is based on heuristics relying on the HTML tags of a Web page and the visual homogeneity between the segmentation blocks[65], called Degree of Coherence.

As for the image screenshot, UPMC team used IECapt (Höhrmann, 2003-2008)[66] as only the DLL of VIPS is freely available. IECapt relies on Internet Explorer's rendering of Web pages and produces images as outputs in BMP, JPEG or PNG. In our experiment, we produced PNG image files of the same version of Internet Explorer as VIPS and used them as an input.

The coordinates of the segmented blocks of the XML tree obtained with VIPS were then used to get the different sub-images corresponding to the visual blocks of the Webpage.

The UPMC team was given access to IM Web Archive and took screenshots of two versions of the same URL at a different time. The pair of images produces were then used for the experiment following the workflow described above in Figure 12: Visual and structural comparison workflow diagram. In parallel, IM QA team compared visually the same set of data trying to assess if a re-crawl was required.

3.2.2.2 Feature extraction step

Basic visual components are selected in the image: pixels, patches, regions, points of interest, edges, etc. To describe these visual components, shape-based, colour, texture features are commonly used. In the case of colour, which is the most often used feature for image indexing, the building of colour features is based on the choice of a colour space, which can include properties such as invariance with illumination. Many systems simply use the RGB colour space; however, transformed colour spaces such as HSV or CIELab[67], differentiating hue from intensity, are also widely used.

Once the features are extracted, building a visual codebook[68] is an effective means of extracting the relevant visual content of an image database; such a process is used by most retrieval systems.

The codebook can be determined *statically* (J.R. Smith, November 2006). The elements (words) of the codebook are then *a priori* defined. A second approach is to perform a *dynamic* or *adaptive* clustering, using a standard clustering algorithm such as k-means[69]. In this case, the visual codebook is adapted to the image database. Once colour, texture, shape-based codebooks have been created, image signatures are computed. For each pixel, the closest codeword is detected and a pixel distribution is generated for each image over the visual codebook.

In the visual codebooks computed from local feature extraction and characterization as Scale-Invariant Feature Transform (SIFT[70]) descriptors, the resulting codebook dimension is usually much larger, around 1000. There is no consensus about the best way to handle this problem of dimension, and there are a lot of attempts to deal with every high dimension giving promising results.

---

[65] Segmentation of images and text into blocks is an important step in document analysis.

[66] http://iecapt.sourceforge.net/

[67] http://en.wikipedia.org/wiki/Color_space

[68] http://en.wikipedia.org/wiki/Bag_of_words_model_in_computer_vision#Codebook_Generation

[69] Hartigan, J. A.; Wong, M. A. (1979). "Algorithm AS 136: A K-Means Clustering Algorithm". Journal of the Royal Statistical Society, Series C (Applied Statistics) 28 (1): 100–108

[70] http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

3.2.2.3 Similarity step

Once signatures are computed, a metric or similarity function has to be defined to compare images. The Euclidian distance[71] is used to compute the similarity (or dissimilarity) between histograms, or more generally a Minkowski distance[72]. Similarity functions can also be learnt from a training set, for instance with kernel methods.

UPMC's solution merges the structural and visual differences to obtain a similarity vector combining the DOM structure of the web page's code and its visual aspect, as shown in Figure 13.

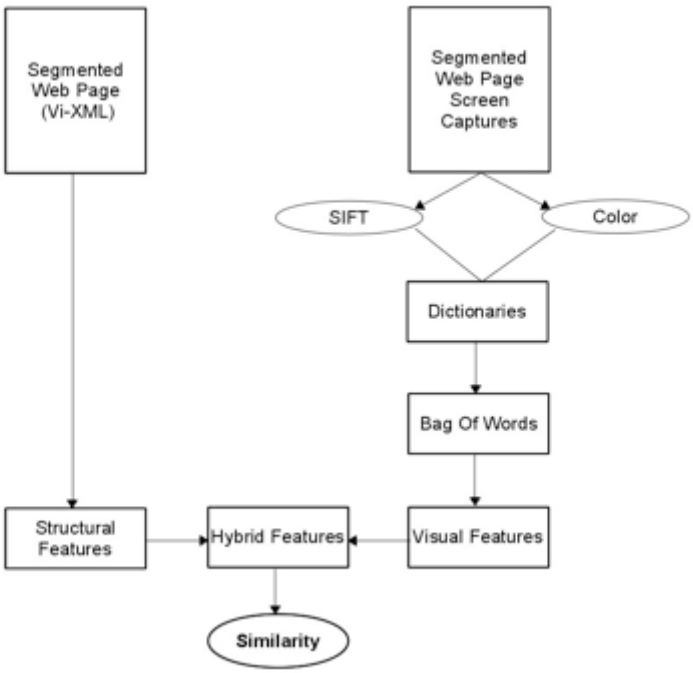

Figure 13: Functional similarity

3.2.3 Taverna Workflow

The solution and workflows developed by UPMC and described above were wrapped up into a Taverna workflow available on my experiment.[73]

Figure 14 represents this decision workflow, which groups steps described above.

[71] http://en.wikipedia.org/wiki/Euclidean_distance
[72] http://en.wikipedia.org/wiki/Minkowski_distance
[73] http://www.myexperiment.org/workflows/2810.html

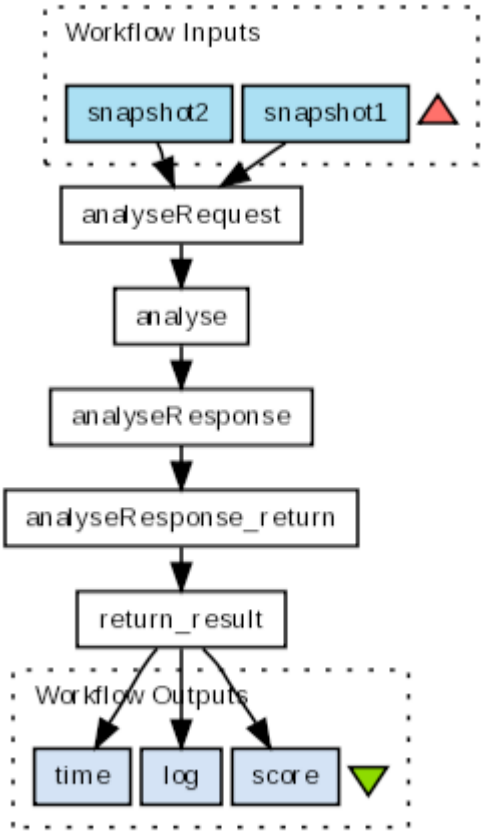

Figure 14: Visual and structural comparison workflow

Screenshots of web pages are given as input, then the analyse web service starts (conversion, feature extraction), and finally, the similarity scores allow the decision: re-crawl / no action required.

The MarcAlizer [74] is the solution developed and wrapped by UMPC and aims at providing decisions scores to define if a web page should be re-crawled or not. As explained in the previous sections, the comparison is made by analysing both the structure and the rendering of a web page.

The class needs as input: VIPS files and web screenshots. Finally, a score that represents the classifier decision and the computational time required are printed. A negative classification score means that the two web pages are dissimilar and thus both need to be archived, a positive score means the two web pages are similar and do not require any action.

To evaluate its system, UPMC used a single 3.47GHz PC to compute the average running times of comparison on the Internet Memory Foundation dataset. The captured images of Web pages are about 929 (±251) width and 1272 (±723) height pixels.

The brute force based algorithm time of execution of the whole process is close to 20 seconds with the SIFT computation as the bottleneck.

[74] http://wiki.opf-labs.org/display/TR/MarcAlizer

3.2.4 Next steps

The solution was evaluated by comparing its results to manual assessment made by the IM QA team and showed a good accuracy. Further tests will be made on larger sets of data to evaluate how to use this solution on whole Web Archives.

Following the performance evaluation outlined in the previous section, UPMC states that some of the parameters can be optimized to speed up the process without losing the accuracy of the results. With these optimizations the time of computation to process each couple of images in less than two seconds with an accuracy of 88%.

One of the big issues will therefore be to get a good enough performance for the whole workflow.

Another issue outlined in this report is the dependency to the Microsoft segmentation tool VIPS. Although the open source solution is under development and should be available very soon, we will need to start new tests based on this new version of the tool to reach the scalable system we are aiming at building within this scenario.

Finally the testing phase itself showed some issues related to the subjectivity of the manual QA annotation. For future tests, assessment guidelines should be reviewed to reduce this aspect if possible and get a higher number of annotated pages.

4 Scalable workflow development (Hadoop)

The Testbeds sub-project (TB) depends directly on the Platform sub-project (PT), as the platform will provide the means to perform large scale workflow execution using a cluster and in a parallelised and distributed manner (1st platform release will be available in January 2013). The experimental phase of the Testbeds work packages TB.WP.1 to TB.WP.3 ends in April 2012, and the subsequent phase of large scale workflow development based on the experimental workflows, shown in the previous sections, begins.

However, in our opinion, the planning of the experimental workflows should not be completely decoupled from the large scale workflow development. Although the details about the SCAPE platform will become available in March 2013 (see [D4.1][75]), it is already known that the technology will be based on Apache Hadoop™[76]:

> Yahoo makes use of Hadoop, an open-source map/reduce engine with a distributed file system, which may serve as a good base for SCAPE. HBase4 (used by Microsoft) and Cassandra5 (used by Facebook) provide high performance and scalable distributed data storage on top of Hadoop's Distributed File System, thus they give us a basis for storing intermediate results and metadata, but do not solve the problem of executing optimised preservation workflows.[77]

In our opinion, the question how preservation workflows can be optimised for the SCAPE platform cannot be answered without taking the concrete scenarios into consideration. It is definitively not only a question of designing an appropriate software architecture that works for preservation workflows in general, but each scenario requires to think about how the "hadoop way" of providing a solution for an issue would be, or, in other words, to think about how the solution of a scenario can be implemented by dividing the solution following the map-reduce paradigm.

Hadoop™ provides an implementation for MapReduce which is a programming model for processing and generating large datasets. The execution of a program is parallelised on multiple clusters of machines. The MapReduce programming model defines a Map and a Reduce function that allows the distribution of computational tasks on various cluster nodes. Roughly speaking, the Map function is taking input data and produces a list of intermediate results in the form of key value pairs. The Reduce function then processes all intermediate results with the same key and produces a list of end results.[78] It is important to note, that the Reduce step is not simply aggregating or collecting the Map results in a centralised way, but runs itself in parallel on multiple nodes.

---

[75] Deliverable D4.1 of work package PT.WP.1

[76] http://wiki.apache.org/hadoop/

[77] SCAPE Description of Work (Annex I of the Grant Agreement with the European Commission), p. 10.

[78] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis: „Evaluating MapReduce for Multi-core and Multiprocessor Systems.", in: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture (HPCA '07), p. 2.

First of all, in order to make use of the Hadoop™ framework, it is required to think about some practical questions:

1. In which form is data stored within the given IT environment? For example, would it make sense to store the entire data in the cluster using the distributed file system or will the large data sets primarily be stored on a NAS (network-attached storage)?
2. Does the cluster have enough storage space to replicate the data completely in HDFS or would it only be possible to store bunches of data of a certain size in the cluster?
3. How would the input data generally be processed by Hadoop? Is the input data splitable and can it therefore be easily divided into equally sized data blocks that are distributable on the Hadoop Distributed File System (HDFS)[79]?
4. Is the MapReduce programming model applicable? More concretely, can the business logic be divided into a Map and a Reduce function in a sensible way?

To some extent, the Platform subproject is dealing with those questions, therefore it is not the objective to answer them all in the context of this work package. The Platform subproject will provide the means to execute workflows on a cluster using the Taverna workflow and there will be general recommendations regarding the questions of how to deal with non-splitable binary data objects that have to be processed as indivisible entities or how to process data that can only be accessed on network attached storage (NAS). However, the last question of those listed above is very specific to a particular scenario.

4.1 Planning for clustered Web Content Testbed experiments

For hardware / software sizing and configuration reason, it has been decided to think about the prerequisites to perform the experiments that are part of the Web Content Testbed scenarios (see section: 1) on a cluster using Apache Hadoop.

The web archive content used in these scenarios (1.1.1 Data set - Web archive of the Austrian National Library) is available as compressed files in the ARC.GZ format produced by the Heritrix crawler.[80] As described in section "Choosing the right ARC unpacker", the first challenge consisted in upackaging the packaged web archive files in order to be able to access the individual items that they contain. While one option could be to load single items directly into the Hadoop Distributed File System, web content definitively contains many small files which are a major barrier to Hadoop, also reported as “The Small Files Problem”:

> “A small file is one which is significantly smaller than the HDFS block size (default 64MB). If you’re storing small files, then you probably have lots of them (otherwise you wouldn’t turn to Hadoop), and the problem is that HDFS can’t handle lots of files.
>
> Every file, directory and block in HDFS is represented as an object in the namenode’s memory, each of which occupies 150 bytes, as a rule of thumb. So 10 million files, each using a block, would use about 3 gigabytes of memory. Scaling up much beyond this level is a problem with current hardware. Certainly a billion files is not feasible.

[79] http://hadoop.apache.org/common/docs/current/hdfs_user_guide.html
[80] https://webarchive.jira.com/wiki/display/Heritrix/Heritrix

> Furthermore, HDFS is not geared up to efficiently accessing small files: it is primarily designed for streaming access of large files. Reading through small files normally causes lots of seeks and lots of hopping from datanode to datanode to retrieve each small file, all of which is an inefficient data access pattern."[81] ("The Small Files Problem" byTom White February 02, 2009. http://www.cloudera.com/blog/2009/02/the-small-files-problem/)"[82]

The web archive of the Austrian National Library contained in December 2012 about 800 Million files where a reasonable part of that collection is small files. While this number does not present a major obstacle with regards to the total storage they would occupy, the main disadvantage comes from storing a multitude of small files in the Hadoop Distributed File System and the startup costs of each individual map task may be too high given that each task would only process little input.[83]

But even the packaged files in the ARC.GZ format produced by the Heritrix crawler which are of the size of 100 Megabytes each are still small given the fact that the default block size in the Hadoop Distributed File System is 64 Megabytes.

Hadoop is primarily designed to process very large or huge amounts of text files, but also provides a suitable format for binary data types with its `SequenceFile`[84]. Especially, the `SequenceFile` class can be used to create containers for smaller files which are aggregated into one large file that make the processing with Hadoop more efficient.

In relation to the scenarios described above, we are planning to distinguish two separate phases: In phase one, the analysis of all the single web archive content items takes place and generates large text files containing the results of the analysis. The analysis can be any kind of metadata extraction or identification of text and/or binary content, the MIME-type extraction in the scenario above being only one of many possible concrete application scenarios in this sense. In phase two the text files are processed further in order to create a report that makes use of the information contained in the text files. With the web archive of the Austrian National Library, which in December 2011 contained about 800 Million files, these text files can become several gigabytes large and the processing of those text files implies a scalability challenge in itself.

In the light of these general remarks, there are three options for integrating the data of the web archive at the Austrian National Library and planning the large scale execution of workflows:

For example, taking the concrete IT-environment of the web archive at the Austrian National Library into account, there are several options for phase 1, the metadata extraction or identification of text and/or binary content:

1. Process web archive data as a whole, replicate all the web archive data as they are (*.ARC.GZ) in the HDFS.
2. Split the web archive data in blocks
   a) Store blocks of the web archive data as they are (10 Terabytes each) in HDFS and process the blocks one after the other.

---

[81] "The Small Files Problem", http://www.cloudera.com/blog/2009/02/the-small-files-problem/
[82] "The Small Files Problem" by Tom White; February 02, 2009:
[83] "The Small Files Problem", http://www.cloudera.com/blog/2009/02/the-small-files-problem/
[84] "Class SequenceFile", http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/io/SequenceFile.html

b) Convert blocks of the web archive data into the SequenceFile format (10 Terabytes each), store them in HDFS and process the blocks one after the other.
c) Create a customised *.ARC.GZ record reader.

3. Data is not loaded into HDFS at all, but all nodes get read access to NAS storage and Hadoop organises the processing.

Of the above mentioned, the first option is probably ruled out because of the high storage costs this would imply. Given that storing the web archive data in HDFS (or any combination of HDFS and HBASE[85]) only is currently not an option. In contrary, options two and three are considered as viable options that will be further evaluated.

Regarding phase two, the question of which are the possible scenarios is easier to answer, because the expected total size of the analysis results in text format can be fully loaded into HDFS and processing large amounts of text files is what Hadoop is designed for, so that it is expected to fully realise the advantages of the framework in this phase.

5 Citation references

- *"The Small Files Problem" byTom White February 02, 2009. http://www.cloudera.com/blog/2009/02/the-small-files-problem/.*
- *SCAPE Description of Work (Annex I of the Grant Agreement with the European Commission), p. 10.*
- *Deliverable D11.1, WP11, PC.WP.3 Quality Assurance Components.*
- *Deliverable D9.1, WP9, PC.WP.1 Characterisation Components.*
- *Raditsch, M. (2011). Experimentel 'real world ARC' extraction report. draft, SCAPE, Testbed, Web Content Testbed.*
- *Van der Knijff, J., & Wilson, C. (2011). Evaluation of Characterisation Tools. check point report, SCAPE, Preservation Components, Characterisation Components.*
- *Van der Knijff, J., Askov Blekinge, A., & Schlarb, S. (2011). WP 9: target characterisation tools. draft, SCAPE, Preservation Components, Characterisation Components.*

[85] http://hbase.apache.org