SCAPE

Scalable Preservation Environments

Web content executable workflows for large-scale execution

Authors
Sven Schlarb (Austrian National Library), Leila Medjkoune (Internet Memory Foundation), William Palmer (British Library)

May 2014

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

In the SCAPE project, the role of the Testbeds is to employ SCAPE platform technology as well as preservation components to develop workflows that can be used to process very large data sets.
The Web Content Testbed develops workflows for processing web content in the context of preservation scenarios related to identification and characterisation of web content as fundamental digital preservation tasks and quality assurance to support web archives in improving web crawls and preserving stored content.
In relation to the previous deliverable, D15.1, The Web Content Testbed has refined preservation scenarios and requirements to provide input for the development of new tools and the adaption of existing software. By using large data sets from a real-world production context, the Testbeds were able to provide feedback, bug reports and to define further requirements for component developers.
In this deliverable, the large-scale web content processing workflows are described in the context of the different preservation scenarios with a focus on the technical solutions. Data sets and details about the data is only mentioned if this is required in order to explain certain technical decisions that have influenced the workflows design.

Table of Contents

1 Introduction

The Testbeds are divided into application areas, and this deliverable is about the development of large-scale executable workflows for processing web content using real-world data from different SCAPE partner institutions doing web archiving.
Web archives usually consist of large data collections of multi-terabyte size, the largest archive being the Internet Archive[1], which according to its own statements stores about 364 billion pages that occupy around 10 petabytes of storage.[2]
All of the memory institutions participating in the *Web Content Testbeds* have data sets of multi-Terabyte size, and the goals of the Testbeds include making use of the *SCAPE Execution Platform* and *Preservation Components* and developing workflows that can be used to process large web content data sets in different institutional environments.
The beginning of 2012 was an important turning point for the Web Content Testbed in several respects. First, during this period, there was a transition from the "preservation scenarios" collected at the beginning of the project to a consolidated list of so called "user stories" describing an application scenario from a user perspective and allowing for requirements to be easily derived for solution and component development. Second, it was at this time that the SCAPE Execution Platform became available and the large-scale workflow development tasks started. This meant that the workflow design, components and technical solutions described in the previous SCAPE deliverable D15.1 had to be checked against the new requirements for large-scale data processing using the SCAPE Execution Platform. Third, from this time on, the Web Content Testbed was using large data sets consisting of archived web content in the *ARC*[3] and *WARC*[4] format from Terabyte to multi-Terabyte size.
In this deliverable, the large-scale web content processing workflows are described in the context of the different user stories with a focus on the technical solutions. Evaluation results will be made available in deliverable D18.2 and will therefore only be mentioned if this is required to explain certain technical decisions that have influenced the workflows development. The user stories are the main structuring principle of this deliverable and are described in section 2.
Section 3 describes the solutions that have been developed in form of workflows that can be used to address the requirements outlined for the various user stories presented in section 2. The solutions are described in a generic manner, while details that only apply in an institutional context are explained in corresponding sub-sections.
The conclusion gives a summary highlighting the most important outcomes of this deliverable.

2 Refined Web Content Testbed User Stories

With the user stories presented in this deliverable, the Web Content Testbed presents large-scale executable workflows for all of the preservation component types, namely migration, characterisation, and quality assurance preservation components. The development of these components is represented by the corresponding work packages *Action Services* (PC.WP.1),

[1] http://archive.org
[2] http://archive.org/web/petabox.php
[3] http://archive.org/web/researcher/ArcFileFormat.php
[4] http://bibnum.bnf.fr/WARC

Characterisation Services (PC.WP.2), and Quality Assurance Services (PC.WP.3) of the *Preservation Components* sub-project.
In this sense, the ARC to WARC Migration *user story* in section 2.1 is focused on the development of a file format migration workflow with an action service. The *Comparison of Web Snapshots user story* in section 2.2 presents the approach to integrate a *Quality Assurance Component* in an existing workflow for web site crawling, and the *File Format Identification and Characterisation user story* in section 2.3 is about the development of deep characterisation workflows identifying and/or characterising the payload content of web archive container files.
All stories have in common that the data they are using is available in form of container files in the ARC format as proposed by the *Internet Archive* or in the successor format WARC, which is published as an ISO Standard.[5] Therefore, finding appropriate ways to deal with these kinds of container files was essential to all of the user stories presented here.
Another requirement that relates to all Web Content Testbed user stories has to do with the fact that the content can in principle be anything harvested from the web. This means that components used to process these files must be able to handle nearly anything in a stable manner and that a process showing unexpected behaviour must not interrupt the main process. This is a necessary requirement that SCAPE Preservation Component developers had to take into consideration, because only by addressing this requirement is it possible to set-up experiments for evaluating workflow runs using large data sets.

2.1 ARC to WARC Migration User Story

The ARC to WARC migration user story deals with the question of how web archive content should actually be stored for the long term. Originally, content was stored in the ARC format, a format developed by the Internet Archive[6] in connection with the *Heritrix Web Crawler*[7] software which produced these files as the default persistent storage file format for crawled web sites. The format was designed to hold multiple web resources aggregated in a single – optionally compressed – container file. But this format was not supposed to be an ideal format for storing content for the long term; for example, it was lacking features that allow adding contextual information in a standardised way. For this reason, the new WARC format provides additional features, especially the ability to hold harvested content as well as any meta-data related to it in a self-contained manner.
A particularity of web archiving is the fact that, while content is continuously changing on one side, it remains static on the other. In order to preserve the changes, web pages are harvested with a certain frequency of crawl jobs. Storing the same content at each visit would create content redundantly and not make efficient use of storage.
For this reason, the *Netarchive Suite*[8] – originally developed by The Royal Library[9] and The State and University Library[10] and used by other libraries as well – provides a mechanism called "deduplication" that detects if content has already been retrieved and therefore references the existing payload content. The information about where the referenced content is actually stored is available in the crawl log files. This means that if the crawl log file is missing, there is actually no knowledge of any referenced content. In order to display a single web page with various images, for example, the

[5] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=44717
[6] http://archive.org/web/researcher/ArcFileFormat.php
[7] https://webarchive.jira.com/wiki/display/Heritrix/Heritrix
[8] https://sbforge.org/display/NAS/NetarchiveSuite
[9] https://sbforge.org/display/NAS/KB
[10] https://sbforge.org/display/NAS/SB

Wayback machine needs to know where to find content that may be scattered over various ARC container files. An index file, e.g. an index in the *CDX file format*[11], contains the required information. To build this index, it is necessary to involve ARC files and crawl log files in the index building process. From a long-term-preservation perspective, this is a problematic dependency. The ARC container files are not self-describing; they depend on operative data (log files generated by the crawl software) in a non-standardised manner. Web archive operators and developers know where to get the information, and the dependency might be well documented. But there is the risk of losing information that is essential for displaying and accessing the content.
This risk is one of the reasons why the option to migrate from ARC to the new WARC format is being considered by many institutions. But, as often happens, what looks like a simple format transformation at first glance rapidly turns into a project with complex requirements that are not easy to fulfil.

In the SCAPE project, there are several aspects that, in our opinion, deserve closer attention:

1. The migration from ARC to WARC is typical done with large data sets; therefore, a solution must provide an efficient, reliable and scalable transformation process. There must be the ability to scale-out, which means that it should be possible to increase processing power by using an appropriately-sized computing cluster to enable organisations to complete the migration in a given time frame.
2. Reading and writing the large data sets comes with a cost. Sometimes, data must be even shifted to a (remote) cluster first. It should therefore be possible to easily hook in other processes that are used to extract additional meta-data from the content.
3. The migration from one format to another carries the risk of information loss. Measures of quality assurance like calculating the payload hash and comparing content between corresponding ARC and WARC instances or doing rendering tests in the Wayback machine on subsets of migrated content are possible approaches in this regard.
4. Resolving dependencies of the ARC container files on any external information entities is a necessary requirement. A solution should therefore not only look into a one-to-one mapping between ARC and WARC, but it should involve contextual information in the migration process.

The ARC to WARC migration user story represents a typical file format migration scenario in that an instance of one file format will be converted into an instance of another format.
But, it is important to note that, because of the dependency on contextual information, this is not a pure one-to-one mapping, as with most image file format conversions, for example. ARC and WARC are both container formats and the reason for aggregating content in these containers is only partly a logical aggregation, where the content belongs to the same harvest definition or the same crawl job. There is also a technical separation between ARC files, where content is dispersed over various container files because of a technical size limit of the ARC files. This means that even use cases about re-arranging the ordering principle of content could be considered, even though this is not in the scope of this deliverable.

2.2 Comparison of Web Snapshots User Story

[11] http://archive.org/web/researcher/cdx_file_format.php

Web Archiving means capturing web content, which is *per se* heterogeneous, complex and highly ephemeral. When captured, resources are stored into a standard archiving format, like the mentioned ARC and WARC file formats, and viewable online thanks to access tools recreating the website look and feel. Each of these steps contains challenges on its own that can have an impact on web archive quality.

First of all, capturing of web content might seem like a simple processing step at the first, glance, but taking a closer look, it turns out to be a complex procedure: Specific tools, so called *Web Crawlers* were developed to capture resources from the web following predefined parameters and a scope by parsing or executing web pages. Most of the web archives use open source parsing crawlers such as Heritrix, a crawler developed by the Internet Archive[12]. Crawls can be selective (domain, sub domain or even page or resource level) or large (.uk or .eu domain, for example). When crawling in a selective manner, web archives are faced with issues such as capturing complex or hidden content. In addition to these technical limitations, crawling at large scale also means pushing the limits when trying to achieve completeness or to avoid temporal incoherencies. Accessing crawled resources online by rebuilding websites' look and feel is yet another challenge. Here again, access tools were developed to allow navigating within web content but limitations exists such as replaying complex content (e.g. non HTTP protocol videos or Flash animations). Therefore, crawled and long term preserved content might not always be accessible to users.

As briefly outlined above, 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 and access; and these known limitations have an impact on the quality of a web archive. For this reason, web archives developed methods and tools to perform crawl quality assurance. Controlling the quality of crawls can be done in different ways. It can consist in producing crawl statistics that are used to check, for instance, that a website crawled regularly is always more or less of the same size and contains more or less the same MIME type distribution. It can also be done by checking visually a sample of website crawled using an access tool. Tools were also developed or adapted to perform automated or semi-automated quality assurance on crawled websites, for instance by listing "404 Not Found" HTTP responses and triggering an immediate re-crawl, if necessary.

Although the solutions applied so far are valuable and allow for the improvement of the quality of web archives, they are not easy to implement on a large scale and can be very expensive to operate. Furthermore, none of the solutions allow an automated detection of rendering issues with the same quality as a human assessor can do. Indeed, the existing tools that allow for the comparison of crawl figures or the spotting of missing resources on more or less – a large scale do not allow for checking the rendering quality.

For this reason, within this work package, we have tried to leverage the visual quality assurance performed by a human by using an image comparison tool and by integrating it to a standard web archiving production workflow. As web archives preserve web content for long-term access and end users, it is crucial to spot rendering issues.

2.3 File Format Identification and Characterisation User Story

Memory institutions doing web archiving usually have an implicit or explicit policy that determines which type of material is collected. Therefore, data may be text documents in all kinds of text

[12] http://sourceforge.net/projects/archive-crawler/

encoding, HTML content loosely following different HTML specifications, audio and video files that were encoded with a variety of codecs, etc.
In order to make any decisions in digital preservation about mitigating the risk of losing information contained in an archive, it is indispensable to have detailed information about the content in the web archive, especially those pieces of information that preservation tools depend on. This information allows the prioritisation of which type of content and which properties of content need special attention and the planning of concrete actions to fulfil the mandate of preserving digital collections.
It is essential for all of these institutions to know precisely what this content is before any preservation planning is undertaken. For example, web archives may contain lots of PDF documents. If a specific version of these PDF documents is known to have renderability issues in modern PDF rendering software, it is necessary to identify exactly the instances of PDF documents that have these properties. If the archive contains video material, the availability of codecs for playing these videos is essential.
In any of these cases it is necessary to know which type of content is to be handled on a coarse granular level, like *MIME-Type* information, or on a fine-granular level, like *DROID*[13] *PUIDs*[14], for example. It is not possible to perform a data migration without knowing exactly what kind of digital object is encountered in the collection and what the logical and technical dependencies of the object are.
The main issue that we are dealing with in this deliverable is how these actions can be achieved using workflows that process large amounts of web archive content at scale. This means that coverage and precision of identification tools as was presented in D9.1[15] are not in the scope of this deliverable; instead, the focus lies in achieving good stability and runtime performance of the workflows.

3 Large-scale Web Content Testbed Solutions

In relation to the user stories presented in the previous section, this section presents the solutions to address the requirements of the user stories in form of a data-flow oriented workflow for processing web archive content.
As already mentioned in the introduction, compared to the previous version of this deliverable (D15.1) the focus has changed, especially regarding the technologies and the size of the data sets being used.
The gap analysis report[16] listed the requirements and tools to be developed; this list was sorted by level of priority assigned to the requirements gathered by the Testbeds.
The solutions presented below address the priorities related to these requirements: Section 3.1 covers the requirement to develop a workflow that is “capable of migrating ARC to WARC” (WCT2.1) including a suggested approach for “checking that the content of the migrated WARC is the same as the original ARC”[17]. Section 3.2 is about the development and integration of “a tool capable of comparing two versions of the same web page”[18]. The priorities of developing "a tool capable of unwrapping and copying the contents of ARC/WARC files into HBase" and of "validating that the files migrated to HBASE are according to the original" have not been addressed because in the end it

[13] http://digital-preservation.github.io/droid/
[14] Pronom Unique Identifiers, see glossary.
[15] SCAPE Deliverable D9.1
[16] SCAPE Deliverable D10.2
[17] SCAPE Deliverable D10.2, p. 28 (Ref. WCT2.1,WCT2.2)
[18] SCAPE Deliverable D10.2, p. 28 (Ref. WCT6.1, WCT6.2)

made more sense to store the ARC files in HDFS instead of unpacking them and storing raw content in HDFS/HBase. This will be explained in more detail. Section 3.3 presents workflows which address the requirement "of doing deep characterization of ARC and WARC files", while the optional extension to use ffprobe[19] in such a workflow to do "deep characterization of video wrapper formats (e.g. AVI)" was not addressed.[20]

3.1 ARC to WARC Migration Solution

In the SCAPE project, the *Apache Hadoop* framework[21] is an essential element of the SCAPE Execution Platform. *Hadoop* is the core which carries the responsibility of efficiently distributing processing tasks to the available workers in a computing cluster.
There were different options to implement a solution (to what) which takes advantage of software development outcomes from the SCAPE project. The first option was using a module of the SCAPE Execution Platform called ToMaR[22], a *Map/Reduce* java application that allows for the easy distribution of command line application processing on a computing cluster (in the following: ARC2WARC-TOMAR). The second option was using a Map/Reduce application with customised reader for the ARC format and customised writer for the WARC format so that the Hadoop framework is able to handle these web archive file formats directly (in the following: ARC2WARC-HDP).
Parts of the following experiment description will also be included in deliverable D18.2, which contains the reports on the final evaluation results, especially those parts that have influenced the workflow design for the sake of completeness.
The experiment was set up to gain evidence in order to decide between two different approaches to developing the large-scale workflow. The main question was whether the native Map/Reduce job implementation had a significant performance advantage compared to using ToMaR with an underlying command line tool execution.
The reason why it was considered that the advantage might be "significant", is that the ARC2WARC-HDP option has an important limitation: In order to achieve the transformation based on a native Map/Reduce implementation, the use of a Hadoop representation of a *Web Archive Record* is required. This is the intermediate representation that is created between reading the records from the ARC files and writing the records to WARC files. As the intermediate representation uses a byte array field to store web archive record payload content, it is theoretically limited to around 2 GB due to the Integer length of the byte array which would be a value near Integer.MAXVALUE.[23] In reality, the practical limitation regarding the manageable payload content size might be much lower depending on hardware setup and configuration of the cluster.
This limitation would create a need for an alternative solution for records with large payload content. And, such a separation between "small" and "large" records would possibly increase the complexity of the application, especially when it is required to involve contextual information across different container files in the migration process.
The implementations used to do the migration are proof-of-concept tools and therefore they are not intended to be used to run a production migration at this stage. This means that there are the following limitations:

[19] http://ffmpeg.org/ffprobe.html
[20] SCAPE Deliverable D10.2, p. 29 (Ref. WCT3.1, WCT3.2)
[21] http://hadoop.apache.org/
[22] https://github.com/openplanets/tomar
[23] http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#MAX_VALUE

1. As already mentioned, ARC2WARC-HDP comes with a file-size limit regarding the in-memory representation of a web archive record; the largest ARC file in the data sets used as evaluation data sets is around 300MB, therefore record-payload content can be easily stored as byte array fields.
2. Exceptions are caught and logged, but there is no gathering of processing errors or any other analytic results.
3. The job implementation neither includes quality assurance nor does it involve contextual information which, as was previously mentioned, both are important aspects of the ARC to WARC migration.

The basis of the implementations for reading web archive ARC container files and for iterating over the records is the Java Web Archive Toolkit (JWAT)[24].

As an example for a process that is used while we are reading the data, the implementations include *Apache Tika*[25] to identify the payload content as an optional feature. All Hadoop job executions can therefore be used with and without payload content identification enabled.

As already mentioned, the ARC2WARC-HDP[26] application was implemented as a Map/Reduce application which is started with the following command line:

```
hadoop jar arc2warc-migration-hdp-1.0-jar-with-dependencies.jar \
-i hdfs:///user/input/directory -o hdfs:///user/output/directory
```

A wrapper Taverna workflow for this Map/Reduce application does this invocation and allows varying the parameters on the workflow input level. See Figure 1.

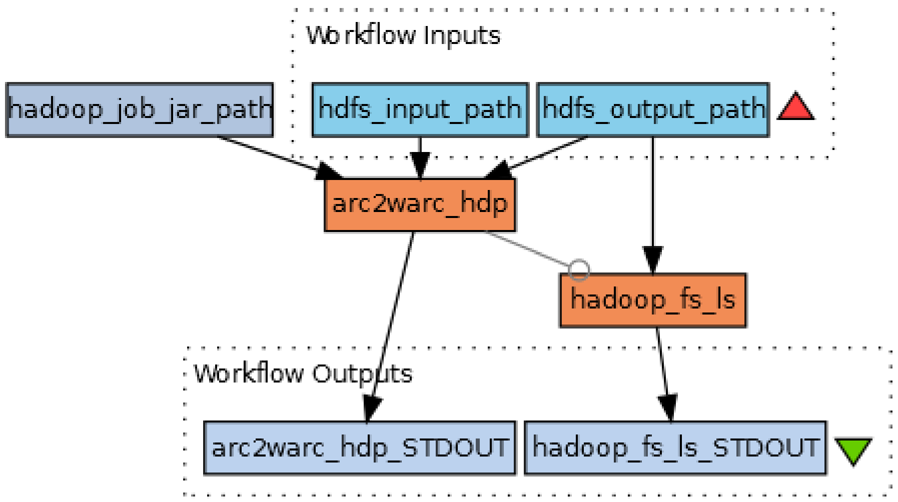


Figure 1: Taverna Workflow for ARC2WARC-HDP execution, http://www.myexperiment.org/workflows/4154

The workflow illustrated here allows configuring the file system location of a jar file implementing the Hadoop job using the "hadoop_job_jar_path" constant value. The "hdfs_input_path" input port indicates the path to the directory that contains the ARC files, and the "hdfs_output_path" property defines the HDFS directory where the migrated WARC files and the job output will be stored.

[24] https://sbforge.org/display/JWAT/JWAT
[25] https://tika.apache.org/
[26] https://github.com/openplanets/hawarp/tree/master/arc2warc-migration-hdp

And the ARC2WARC-TOMAR workflow is using a command line Java-Implementation[27] and executed using ToMaR. One bash script[28] was used to prepare the input needed by ToMaR and another bash script[29] to execute the ToMaR Hadoop job, a combined representation of the workflow is available as a Taverna workflow illustrated in Figure 2.

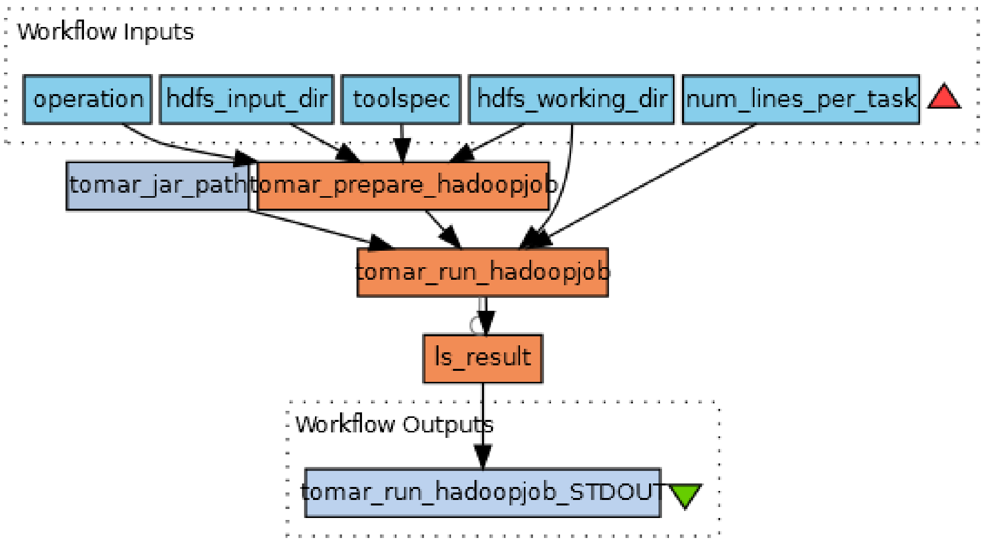


Figure 2: Taverna Workflow for ARC2WARC-TOMAR execution, http://www.myexperiment.org/workflows/4144

The workflow allows processing an HDFS input directory using ToMaR. The "hdfs_working_dir" input port is the HDFS input directory which contains the data to be processed by ToMaR. The "toolspec" input port contains the *toolspec* XML describing operations that can be used (see "operation" input port). The "operation" input port defines the operation to be used in the current ToMaR job execution (see "toolspec" input port, an operation port used here must be defined in the tool specification). The "hdfs_working_dir" input port defines the directory where the outputs will be stored in a date/time-subdirectory.

The following is an example for a workflow output in HDFS:

```
tomarworkingdir/20140304130007/dataout
tomarworkingdir/20140304130007/joboutput
tomarworkingdir/20140304130007/tomar-controlfile.txt
tomarworkingdir/20140304130007/toolspec
```

The "dataout" directory contains the output data of the ToMaR process. Depending on the operation used, this can be the result of a file format identification process or a data migration process. The "joboutput" directory contains the Hadoop job output of the ToMaR Hadoop job. The "tomar-controlfile.txt" file is the input file for the ToMaR Hadoop job execution. The "toolspec" directory contains the tool specification file given by the "toolspec" input port.
A so called “tool specification” is needed to start an action in a ToMaR Hadoop which specified inputs and outputs and the java command to be executed:

[27] https://github.com/openplanets/hawarp/tree/master/arc2warc-migration-cli
[28] https://github.com/openplanets/hawarp/blob/master/tomar-prepare-inputdata/scripts/prepare-input.sh
[29] https://github.com/openplanets/hawarp/blob/master/tomar-prepare-inputdata/scripts/tomar-run-hadoopjob.sh

```
<?xml version="1.0" encoding="utf-8" ?>
<tool xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://scape-project.eu/tool tool-
1.0_draft.xsd"
    xmlns="http://scape-project.eu/tool"
    xmlns:xlink="http://www.w3.org/1999/xlink" schemaVersion="1.0"
    name="bash">
  <operations>
    <operation name="migrate">
      <description>ARC to WARC migration using
          arc2warc-migration-cli</description>
      <command>
        java -jar arc2warc-migration-cli-1.0-jar-with-
dependencies.jar
             -i ${input} -o ${output}
      </command>
      <inputs>
        <input name="input" required="true">
          <description>Reference to input file</description>
        </input>
      </inputs>
      <outputs>
        <output name="output" required="true">
          <description>Reference to output file</description>
        </output>
      </outputs>
    </operation>
  </operations>
</tool>
```

Listing 1: ToMaR Tool specification file for the arc2warc-migration executable jar

All commands, such as the one displayed as text node of the command element shown Listing 1, allow the use of a “-p” flag to enable Apache Tika to identify payload content.

Additionally to scale out the migration of ARC container files to the new WARC format, there is also the need to find a way to make sure that the instances in the target format are valid new representations of the original. This means that the WARC container files can be used instead of the ARC container files to display archived web pages using the Wayback machine.
In order to render web pages based on the ARC or WARC container files, the Wayback machine needs an index to find required archived items. One such index type is the *CDX index*[30] which consists of individual lines of text, each of which represents a single web document.
There is the basic assumption that except for container-file related fields, the CDX index must be the same for both types of container formats for displaying web content to assure that the content can be rendered correctly using the Wayback machine. In this regard, the workflow in Figure 3 represents an executable non-scalable workflow that is used to verify if the migration works correctly on small sample data sets.

[30] http://archive.org/web/researcher/cdx_file_format.php

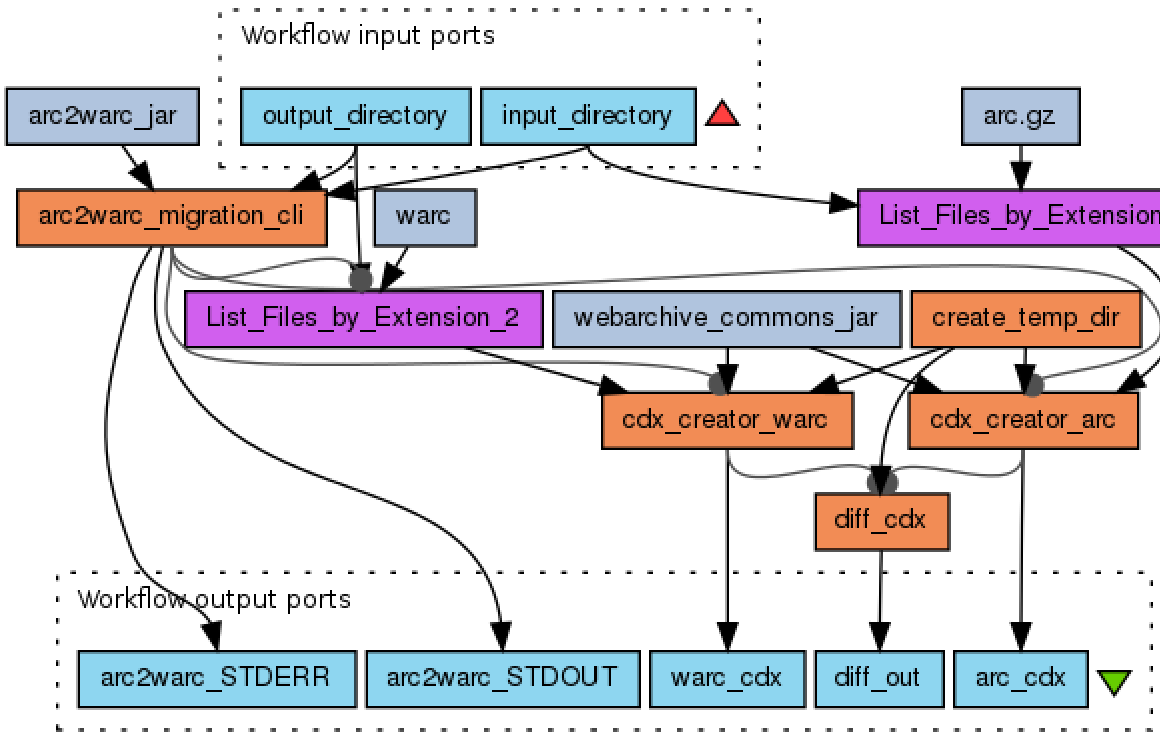


Figure 3: Taverna Workflow for migrating ARC to WARC and checking differences regarding the CDX index files, http://www.myexperiment.org/workflows/4267

The workflow has an input port "input_directory" which is a local path to the directory containing the ARC files, and an input port "output_directory" which is the directory where the workflow outputs are created. The files in the input directory are migrated using the "arc2warc_migration_cli" tool service component to perform the migration. The "cdx_creator_arc" and "cdx_creator_warc" tool service components create cdx index files for both, the original ARC file and the migrated WARC file which, subsequently, are compared by the "diff_cdx" tool service component that uses the CSV file comparison tool csvdiff[31] to compare defined columns of the two CSV files.

The creation of valid and equivalent versions of the CDX index files is a necessary condition to enable quality control of the migration result in two respects: First, the CDX index files are compared if they contain the same records in terms of the information used to access the records by the Wayback machine and the checksum information of the individual record payloads. Second, the CDX index file is used to allow selective rendering of archived web pages and to compare the rendering outcomes. The scalable variant of this workflow executes the individual steps, as they are, ARC to WARC migration, ARC and WARC CDX index creation, and ARC and WARC CDX index comparison separately using ToMaR with the same executables used in the tool service components of the presented workflow.

3.2 Comparison of Web Snapshots Solution

The solution used to conduct QA within a web archiving workflow by comparing two snapshots of a web page relies on a tool developed by the UPMC and enhanced during the project by both the

[31] http://csvdiff.sourceforge.net

UPMC and the IMF teams. As outlined in deliverable D15.1, the solution is strongly linked with work accomplished by both teams as part of WP 11 (PC.WP.3 Quality Assurance Components) and WP12 (PW.WP.1 Automated Watch) of the SCAPE project. Deliverable D15.2 is therefore based on input from both work packages (see D11.2[32] and D12.2[33]).

As part of the SCAPE project, the UPMC team developed a tool, the Marcalizer[34], that allowed a visual and structural (based on an analysis of web pages DOM tree) comparison of snapshots of web pages and that would provide a similarity score as an output.

The tool evolved since D15.1 to solve issues such as a dependency on VIPS[35] (Vision-based Page Segmentation Algorithm) for the structural comparison part, or the scalability issue that was required to be used within a real use case workflow. The Marcalizer as described in D15.1 gave birth to several solutions, all available as open source tools.

The Marcalizer is defined within D11.2 as a tool containing a supervised framework; the Pagelyzer[36] is defined as a tool that allows for comparisons of two web pages and that provides a similarity score as an output.

Both tools are fully described by UPMC within D11.2 and work through:

- 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

The final version of the Pagelyzer tool[37] was recently released and now integrates a complete workflow that allows for taking snapshots of web pages, analyses these snapshots (visually and structurally) and finally provides a global similarity score as an output of the visual and structural comparison. The IMF team performed preliminary tests using this new version of the tool and is currently working on integrating its use within the SCAPE platform for larger scale experiments. Figure 4 below shows the related Taverna workflow.

[32] SCAPE Deliverable D11.2
[33] SCAPE Deliverable D12.2
[34] https://github.com/lechervy/MarcAlizer
[35] http://research.microsoft.com/apps/pubs/default.aspx?id=70027
[36] https://github.com/openplanets/pagelyzer
[37] http://scape.lip6.fr/pagelyzer_1.0.0_all.deb

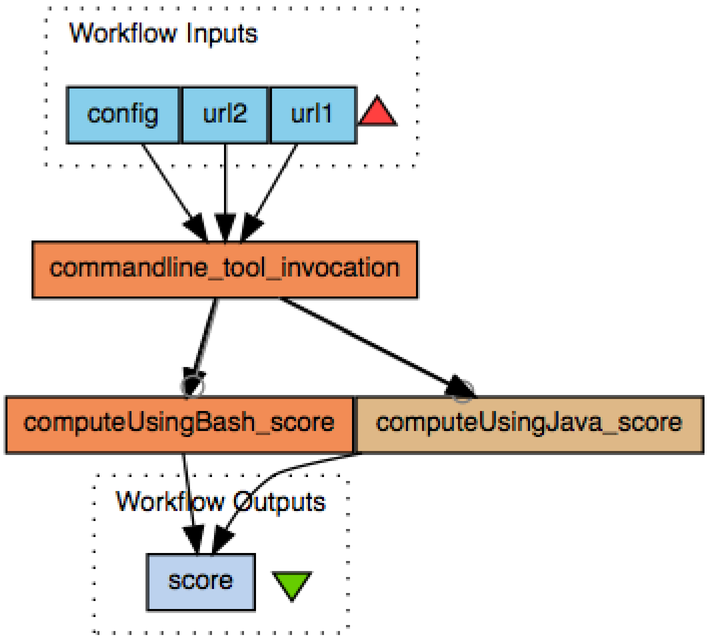


Figure 4: Taverna workflow for detecting changes in web pages and their rendering

As part of the WP12 and in order to solve performance and usability issues inherent to the first versions of Marcalizer, the IMF team developed a wrapper application, the browser-shots tool[38].
The browser-shots tool was built with the aim of performing automated visual comparisons of snapshots of web pages within a web archive. The application orchestrates several tools and scripts that allow a whole QA workflow to be performed automatically. The first version of the browser-shots uses Selenium[39] to automatically take snapshots of web pages combined with the first version of the Marcalizer for the comparison of images.
Selenium was chosen among other similar frameworks as it allows the use of several browsers and browser versions, which is a strong requirement when applying QA to web content. So far, the browsers that have been tested are Firefox (all versions), Chrome (latest version) and Opera (versions 11th and 12th). Selenium also allows to set as many instances as needed and to run these in parallel, which is critical when looking at a tool that should be scalable.
The workflow is as follows:

1. Snapshots of two URLs to compare are taken using the Selenium framework. This step is configurable depending on the user need and resources (comparison can be made with one or several browsers).
2. Each pair is compared using the Marcalizer (this version of the tool is to be replaced by the Pagelyzer).
3. Provide as an output a list of the similarity score of each pair of snapshots compared.

The first implementation of the browser-shots was made on a Debian Squeeze (64 bits) platform. Our tests made on the IMF platform (3 nodes) on around 440 pairs of URLs confirmed the need to improve the browser-shots further to make it more scalable. Indeed, the average processing time for

[38] https://github.com/crawler-IM/browser-shots-tool
[39] http://docs.seleniumhq.org/

a pair of URLs was of 16 seconds. On the Selenium side, we met robustness issues (it failed regularly during tests) and performance issues (taking screenshots was actually time consuming).
The second implementation therefore has tried to solve or at least reduce these performance issues by using an optimised version of the Marcalizer (any new optimised version will continue to be added to the browser-shots). As one of the bottlenecks was due to the orchestration between tools, the implementation was modified so that snapshots taken by the Selenium are directly sent to Marcalizer using streams (instead of having intermediary parameters). The Selenium is also represented as a MapReduce job running on a Hadoop cluster to parallelise the processing of the input (list of URLs together with a list of browsers' versions).
Following this implementation, tests were made showing an improvement in terms of performance and quality of results. The browser snapshot step and the Marcalizer comparison were reduced to 2 seconds each. Of course, as Selenium renders the web page before producing the snapshot, the overall performance greatly depends on the size and complexity of the web pages chosen.
A larger experiment was launched on two worker nodes of the SCAPE central instance with this enhanced implementation. The test was made on around 13 000 URLs with Firefox and Opera browsers. Future larger experiments will be detailed in the next benchmarking deliverables of the SCAPE project. The next steps also include testing the latest version of the Pagelyzer tool (java version) and comparing it in terms of performance and scoring quality to the browser-shots currently in use.

3.3 File Format Identification and Characterisation Solution

It was pointed out in the predecessor deliverable D15.1, that for a solution intending to do identification and characterisation of web content at scale, it was crucial to find an efficient way to unpack content from ARC files and apply content identification.
However, apart from the approach that unpacks a file in order to apply tools, there was also the option to read ARC container files and access file streams of the web content payload records without having to make them available as files beforehand.
In the following sections, several workflows are presented which represent different possible approaches, each depending on the properties of the tool that is used to perform the file format identification or characterization task.

3.3.1 Unpack ARC container files and apply file format identification using DROID

In deliverable D15.1, several alternatives for ARC unpacking were presented[40]. One obvious option was to use unpacking as the first step in developing large scale workflows that make the individual files contained in the ARC container file available in the file system first, in order to run the DROID identification subsequently. The ARC Unpacker had been identified as the best candidate for this purpose.[41]
Because of the fact that the SCAPE Execution Platform is based on Apache Hadoop, we must take into consideration that Hadoop's strength is the processing of very large files. If the file format identification is applied to a set of files originating from an office or web context, as it is naturally the case in a web archiving context, we are usually dealing with many small files, like HTML files, PNG or JPEG images. There might also be large multimedia files, but large does not necessarily mean adequate for Hadoop processing. It must be possible to split the input files of a Hadoop job in to

[40] See SCAPE deliverable D15.1, section 2.6, p. 14.
[41] See SCAPE deliverable D15.1, section 2.6, p. 15.

independent parts so that they can be processed in parallel during the Map phase of a MapReduce job.
In this first approach the ARC container files are completely unpacked to HDFS and the individual files must be are being made available as complete and undivided units.
In order to achieve efficient processing using the SCAPE Execution Platform, it was essential to determine how small files would ideally be processed using the Hadoop framework. If the small files would have been made available in the *Hadoop Distributed File System* (*HDFS*) and defined as input for a Map function performing the file identification, Hadoop would create one task per file which – given the additional time required for initiating a task – would have resulted in a bad runtime performance.
One possible approach to overcoming this obstacle is to put references to all files that are going to be processed into a text file and then use this text file as input for the Hadoop job. This requires that all worker nodes of the cluster can access the referenced file paths, e.g. by adding mount points to the cluster nodes so that a file path references the same file on each cluster node. Using this method the Hadoop framework does not generate one task per file, but the size of the task only depends on the split size of the input text file, i.e. all file paths contained in a 64 Megabyte section (default split size) of the text file.
However, using DROID as it was released in version 6.1.3 and with the *Pronom Signature File* version 67[42] for certain file formats (e.g. PDF), the files were accessed on the file system using a file path specified in a metadata object assigned to the file.[43]

[42] http://www.nationalarchives.gov.uk/documents/DROID_SignatureFile_V67.xml
[43] For example, the BinarySignatureIdentifier used file path related metadata of the RequestMetadata object. The evidence was given by differences regarding the identification results if the file path of the RequestMetadata object was not pointing to an existing PDF file. See https://github.com/digital-preservation/droid/blob/master/droid-core/src/main/java/uk/gov/nationalarchives/droid/core/BinarySignatureIdentifier.java

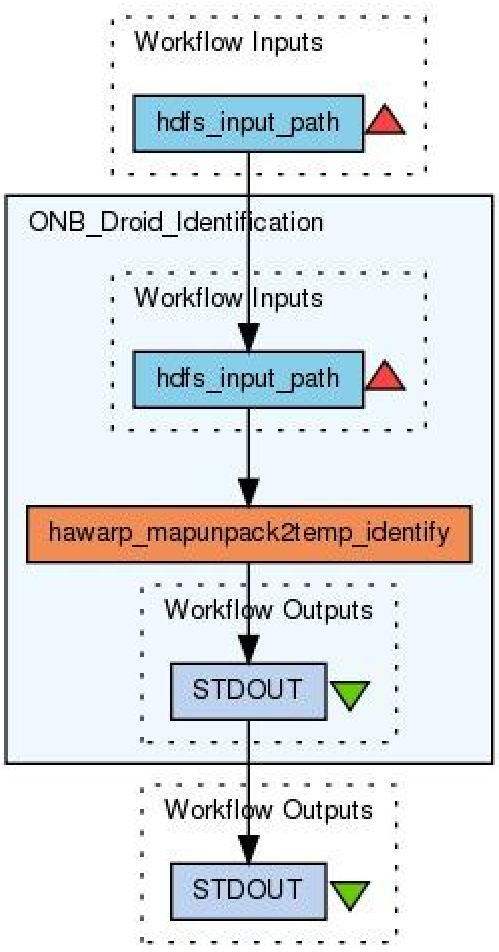
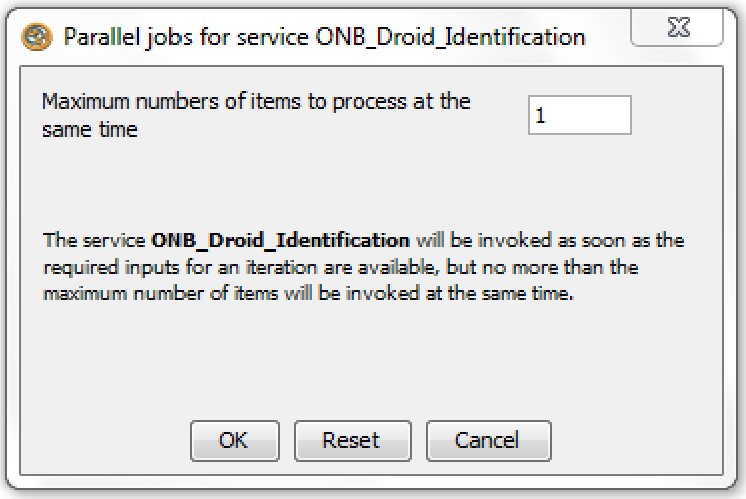


Figure 5: Taverna Wrapper workflow for a Droid Identification Hadoop job, http://www.myexperiment.org/workflows/4222

The Taverna workflow shown in Figure 5 represents a Hadoop job that takes a list of HDFS directories containing text files with paths to ARC container files as input. The main workflow is a wrapper around the nested workflow "ONB_Droid_Identification", which uses an HDFS directory containing text files with absolute paths to ARC container files as input. As shown in Figure 6, the number of parallel jobs for the nested workflow is set to 1; this way the Hadoop job processing can be split into several jobs that are executed sequentially in case this is required due to limited memory or local storage capacity available on the cluster nodes.


Parallel jobs for service ONB_Droid_Identification

Maximum numbers of items to process at the same time: 1

The service **ONB_Droid_Identification** will be invoked as soon as the required inputs for an iteration are available, but no more than the maximum number of items will be invoked at the same time.

Figure 6: Configuration of the number of parallel jobs for the nested workflow "ONB_Droid_Identification",, http://www.myexperiment.org/workflows/4222

The files are unpacked to a temporary directory on the worker nodes in order to apply DROID identification subsequently to the individual files.
The workflow component "hawarp_mapunpack2temp_identify" is based on the Hawarp[44] software module "unpack2temp-identify"[45] which is not limited to DROID but can be used to apply a set of identification tools to the unpacked file items in parallel.
The major drawback of the approach presented by this workflow is that there is no benefit of "data locality" which is one of the most important features of Apache Hadoop. "Data locality" refers to the fact that Hadoop tries to assign map tasks to nodes that are close to the data, i.e. the processing cores are on the same machine as the hard disk storing the data blocks. As data is stored redundantly[46], Hadoop might be able to choose between alternative nodes that have a certain data block available and then select the one where the task tracker has less workload. However, if data is unpacked to the local file system of one worker node, it is not possible to take advantage of this benefit. This way the workflow only provides the means to run the unpacking of ARC files and the identification process in parallel.
An alternative to this approach is to apply DROID directly on file streams. In this case it would not be required to unpack the ARC files completely beforehand. This comes also with the advantage that in most cases it is not required to read the complete file stream in order to detect the file format. At least for simple and commonly used file formats and in use cases where file streams contained in container files are not going to be analysed, it is sufficient to read the first byte chunks to be able to determine the format reliably. But for this purpose, the DROID application needs to be adapted, which is the approach of Nanite, a software tool that uses the DROID API and which will be presented in the next section 3.3.2.

[44] https://github.com/openplanets/hawarp
[45] https://github.com/openplanets/hawarp/tree/master/unpack2temp-identify
[46] The replication factor in the HDFS configuration determines how many copies of data blocks exist in the file system. The default configuration is replication factor 3.

3.3.2 File format identification using Nanite (DROID API)

Nanite[47] is an identification/characterisation tool for web archives, written in Java that brings together several different technologies and takes a very different approach to using ToMaR (as described below). Initially developed within SCAPE, it has an easy to use API around DROID that is both fast and allows for streams of data to be identified (rather than just files). These features make it well-suited for integration into a MapReduce program. Nanite currently consists of two main modules;

1. Nanite-Core: an API for DROID[48]
2. Nanite-Hadoop: a native MapReduce program for identification/characterisation

Nanite can also make use of other identification/characterisation modules such as;

- Libmagic-jna[49]: identification according to `file` command's identification data
- Apache Tika: identification
- ProcessIsolatedTika[50]: characterisation using Apache Tika[51]

Nanite uses the alternative method for accessing files within ARC/WARC files as described above, i.e. it does not unpack the contents of the files before use. It makes use of a Hadoop RecordReader that can directly open the ARC files and operate on its contents, without an intermediate step via the file system. As all of the libraries used by Nanite are written in Java, data is easily passed to them by InputStreams.

The identification and characterisation tools that will be used can be configured at runtime via a properties file in the jar.

A Mapper is responsible for running through one ARC file, with as many Maps executed as there are files contained within the ARC. A typical runtime of a Mapper is approximately five minutes. The setup of Nanite-Core, ProcessIsolatedTika, etc. forms part of that runtime; however, the setup time is small compared to the time it takes to run all the Maps.

A high level overview of the execution workflow is as follows:

1. A list of ARC/WARC files is passed to Nanite-Hadoop
2. A Mapper is initialised, one per input file
3. A Map is executed, once for each file inside the archive
    a. The year the file was harvested is extracted
    b. The MIME type the server sent for the file is extracted
    c. The file extension is derived from the URI, if possible
    d. The default settings then use Nanite-Core, Tika and ProcessIsolatedTika to identify and characterise the file[52]

[47] https://github.com/openplanets/nanite

[48] http://www.nationalarchives.gov.uk/information-management/our-services/dc-file-profiling-tool.htm

[49] https://github.com/openplanets/libmagic-jna-wrapper

[50] https://github.com/willp-bl/ProcessIsolatedTika

[51] Web archives are almost certain to contain data that is corrupt and otherwise damaged, this places stresses on software and it can make Tika prone to crashing or hanging. To ensure that the Nanite process does not fail due to its dependencies failing, we created a library that transparently uses Tika in a separate operating system process. This ensures that the JVM Nanite is running in is not affected by crashes in Tika. A similar process could be used to run FITS, if it provided a REST interface. Other approaches to using Tika for characterisation were considered and tested , with the process isolation method deemed most robust (http://www.openplanetsfoundation.org/blogs/2014-03-21-tika-ride-characterising-web-content-nanite).

4. The Reduce phase creates a tab separated values file that contains the file extension, year of harvest and various MIME types, along with the number of times each record line is produced

Testing has shown that there are be differences in how files are identified by various tools. Nanite-Core (and therefore DROID) will only identify files that exactly match its signatures. This means that if, for example, a JPG file had no end of file marker (as required by the specification) then it will not be identified by DROID, whereas Tika will recognise the file by header alone. Web archives are certain to contain files that do not adhere to specifications, and thus DROID is unable to identify those files. This has been reported to the DROID developers[53] but using DROID to identify files from a web archive falls outside their use case.

Nanite currently uses Tika for characterisation, however, support could potentially be extended to FITS. Characterisation output from Tika can be loaded in to c3po[54], but this has not been tested.

Initial testing has indicated that identifying and characterising files using Nanite is fast and its approach to be promising[55]. It should be noted that Nanite makes use of fewer SCAPE components than other approaches, in favour of tighter integration with Hadoop as a native MapReduce job. Therefore although it does not integrate with other SCAPE projects as well, it does offer very fast execution.

It would be possible to integrate other characterisation tools, such as FITS into Nanite, using the same method as ProcessIsolatedTika. For this to happen FITS, or other tools, would need to provide a REST, or other API at runtime.

3.3.3 File format characterisation using Fits

FITS (File Information Tool Set)[56] was chosen as a test case for ToMaR for two reasons: First, the FITS approach of producing "normalised" output on the basis of various file format characterisation tools makes sense, and therefore, enabling the execution of this tool on very large data sets will be of great interest for many people working in the digital preservation domain. Second, the application is challenging from a technical point of view, because it starts several tools as sub-processes. Even if a process takes only one second per file, we have to keep in mind that web archives often have billions of files to process.

The workflow in Figure 7 is an integrated example of using several SCAPE outcomes in order to create a profile of web archive content. It leverages existing digital preservation functionality and develop a data processing chain starting by unpacking the content making it available to ToMaR, use ToMaR to apply the FITS characterisation and then ingest the FITS characterisation output into C3PO[57] in order to allow browsing the results in a web-based application and to see aggregated statistics.

[52] When ProcessIsolatedTika is used it stores the characterisation output in a SequenceFile in HDFS, one per input archive. So 100 input files will result in 100 output files. One test showed that the size of this (compressed) output was approximately 1/80th of the total size of the input archives.

[53] https://groups.google.com/forum/#!topic/droid-list/sUCwaO1k1kk

[54] https://github.com/openplanets/c3po

[55] http://wiki.opf-labs.org/display/SP/EVAL-BL-WCT-01

[56] https://code.google.com/p/fits/

[57] https://github.com/peshkira/c3po

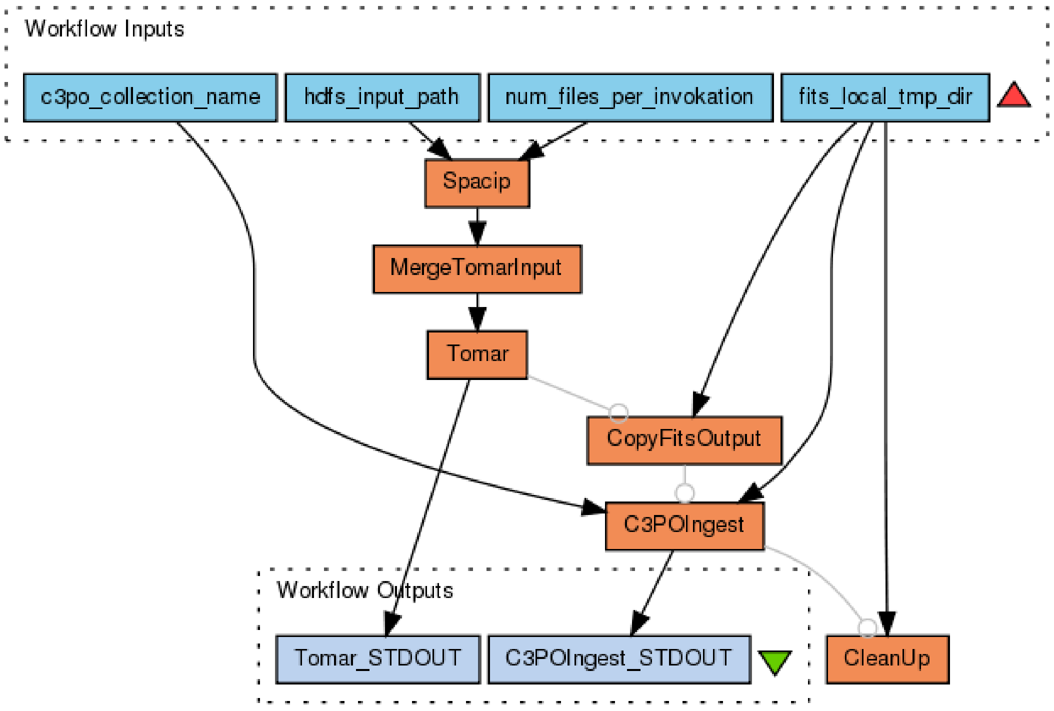


Figure 7: Web Archive FITS Characterisation using ToMaR, available on myExperiment: www.myexperiment.org/workflows/3933

The inputs in this workflow are defined as follows:

- "c3po_collection_name": The name of the C3P0 collection to be created.
- "hdfs_input_path", a Hadoop Distributed File System (HDFS) path to a directory which contains text file(s) with absolute HDFS paths to ARC files.
- "num_files_per_invocation": Number of items to be processed per FITS[58] invocation.
- "fits_local_tmp_dir": Local directory where the FITS output XML files will be stored

The workflow uses a Map-only Hadoop job[59] to unpack the ARC container files into HDFS and creates input files which subsequently can be used by ToMaR.[60] This job is invoked from the Spacip[61] Taverna tool service component. After merging the Mapper output files into one single file (MergeTomarInput), the FITS characterisation process is launched by ToMaR as a MapReduce job. ToMaR uses an XML tool specification[62] document which defines inputs, outputs and the execution of the tool. The tool specification document for FITS[63] used in this experiment defines two operations, one for single file invocation, and the other one for directory invocation.
FITS comes with a command line interface API that allows a single file to be used as input to produce the FITS XML characterisation result. But if the tool were to be started from the command line for

[58] https://code.google.com/p/fits/
[59] https://github.com/openplanets/hawarp/tree/master/tomar-prepare-inputdata
[60] https://github.com/openplanets/tomar
[61] Spacip has now been integrated as a module in hawarp and is available as the module tomar-prepare-inputdata at https://github.com/openplanets/hawarp/tree/master/tomar-prepare-inputdata
[62] https://github.com/openplanets/scape-toolspecs/blob/master/toolspec.xsd
[63] https://github.com/openplanets/hawarp/blob/master/tomar-prepare-inputdata/toolspecs/fits.xml

each individual file in large a web archive, the start-up time of FITS including its sub-processes would accumulate and result in a poor performance. Therefore, it comes in handy that FITS allows the definition of a directory which is traversed recursively to process each file in the same JVM context. ToMaR supports this functionality by defining an operation that processes a set of input files and produces a set of output files.

The question how many files should be processed per FITS invocation can be addressed by setting up a Taverna experiment like the one shown in Figure 8. The workflow presented above is embedded in a new workflow in order to generate a test series. A list of 40 values, ranging from 10 to 400 in steps of 10 files to be processed per invocation is given as input to the "num_files_per_invocation" parameter. Taverna will then automatically iterate over the list of input values by combining the input values as a cross product and launching 40 workflow runs for the embedded workflow.

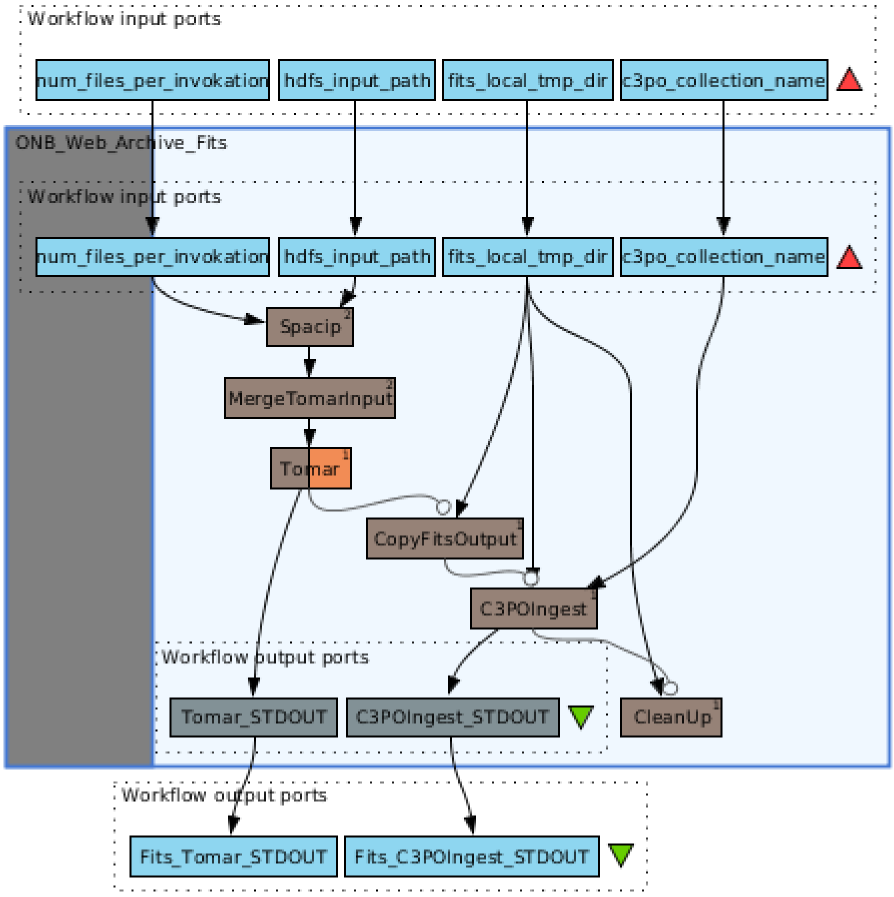


Figure 8: Wrapper workflow to produce a test series, available on myExperiment: www.myexperiment.org/workflows/4229

Processing larger data sets can be done in a similar manner to the one that is shown in Figure 8, only that a list of input directory HDFS paths determines the sequence of workflow runs and the number of files per FITS invocation is set as a single fixed value.

4 Conclusion

In this deliverable the large-scale workflow development outcomes of the Web Content Testbed (TB.WP.1) have been presented. The focus of this presentation lied on the question how the SCAPE Execution Platform was used to achieve the processing of very large data sets. Furthermore, aspects of interoperability of the preservation components developed in the PC sub-project in the context of various forms of the SCAPE Execution Platform (i.e. the so called local instances at memory institutions) were of primary interest. The goal was to show how different types of preservation components, from the areas of file format identification and characterisation, file format migration, and quality assurance, can be used together in large-scale data-flow oriented workflows.

At the moment of writing this deliverable, concrete evaluation results are still in the process of being created and the results are going to be presented in SCAPE deliverable D18.2. Nevertheless, from a general point of view, a positive conclusion can be drawn here regarding the use of the SCAPE Execution Platform which together with the individual tools allowed scaling out the various types of digital preservation tasks.

To highlight only some of the SCAPE outcomes, the toolset on top of the Apache Hadoop framework has proven to be a solid basis to ensure stability of long-running processes, hardware-failure resilience, and provided an inexpensive option to keep large amounts of data securely stored by keeping data redundantly and allowing immediate processing of large data sets.

The development of workflows undertaken in this work package helped defining the requirements for adapting the APIs and interfaces of legacy digital preservation tools to be able to use them in the context of the SCAPE Execution Platform.

The Taverna Workflow design and execution workbench turned out to be of great use in the phase of finding the appropriate processing tool chain and allowed a detailed feasibility study when evaluating alternative approaches and the tuning of parameters of the tools.

The SCAPE toolwrapper[64] allowed to easily deploying a large number of digital preservation components on a *SCAPE Local Instance* on various nodes of the computer cluster, because in certain digital preservation scenarios it is required that each individual node is able to execute a specific preservation component.

C3PO as a web content profiling tool based on FITS characterisation results applied to web content showed a promising new direction in achieving detailed information about the content stored in large web archives and a way to make use of the extracted information for the purpose of doing preservation planning. The integration of these components with Planning and Watch (PW) outcomes allows memory institutions to make use of the extracted information for preservation planning in a standardised manner using well-defined digital preservation measures and digital object properties.

[64] https://github.com/openplanets/scape-toolwrapper

5 Glossary

Term	Abbreviation	Definition
Action Service		An action service is a type of a digital preservation service that performs some kind of action on a digital object, e.g. migrating the object to a new file format.
Apache Hadoop		Framework for processing large data sets on a computer cluster. See http://hadoop.apache.org
Apache Tika		Software for identifying file formats. See https://tika.apache.org/
ARChive format	ARC	ARC is a lossless data compression and archival format which was originally used by the Heritrix Web Crawler developed by the Internet Archive. See http://archive.org/web/researcher/ArcFileFormat.php
CDX file format		Index file of ARC (see corresponding glossary entry) or WARC (see corresponding glossary entry) container files used by the Wayback machine (see corresponding glossary entry) to render archive web pages. See http://archive.org/web/researcher/cdx_file_format.php
CDX index		See CDX file format.
Characterisation Service		A characterisation service is a type of a digital preservation service that extracts any kind of information from a digital object, as an identifier or file related properties, for example.
Data locality		"Data locality" refers to the fact that Hadoop tries to assign map tasks to nodes that are close to the data, i.e. the processing cores are on the same machine as the hard disk storing the data blocks.
DROID		Software developed by the National Archives (UK) to determine a unique file format identifier (PUID, see corresponding glossary entry). DROID is a software tool developed by The National Archives (UK) to perform identification of file formats. See http://digital-preservation.github.io/droid/

Execution Platform	EP	An extensible infrastructure for the execution of digital preservation processes on large volumes of data (using a combination of Apache Hadoop and Taverna)
(SCAPE) Experiment Evaluation		Findings and results, both measurable and non-measurable, of a particular execution of an Experiment, within the Testbed sub-package.
(SCAPE) Experiment		A unit of work that combines a dataset, one or more preservation components, a workflow and a processing platform that can be used to evaluate SCAPE technology and provide evidence of scalable processing
File Format Characterisation		The process of determining the properties of a file format, for example, the bit depth, colour space, width of an image, the frames per second of a video, etc.
File Format Identification		The process of determining the identity of a file format instance, typically by assigning an identifier, as the PUID (see corresponding glossary entry) as a precise identifier or a MIME Type (see corresponding glossary entry) identifier as a vague file type identifier.
Hadoop		See Apache Hadoop.
HDFS	HDFS	Hadoop Distributed File System. This is Hadoop's file system which is designed to store files across machines in a large cluster.
HBase	HBase	Distributed database on top of Hadoop/HDFS, see https://hbase.apache.org
Heritrix Web Crawler		Web crawler engine used to harvest content from the internet and store it in a web archive. The Heritrix Web Crawler was originally developed by the Internet Archive (see corresponding glossary entry). See https://webarchive.jira.com/wiki/display/Heritrix/Heritrix
Internet Archive		The Internet Archive is a digital library which provides permanent storage of and free public access to collections of digitized materials, including websites, music, moving images, and nearly three million public-domain books. See https://archive.org

Map/Reduce	MR	A programming paradigm for processing large data sets using a parallel, distributed algorithm on a SCAPE cluster.
MIME Type		A standard identifier used on the Internet to indicate the type of data that a file contains.
MyExperiment		A web application to allow users to find, use and share scientific workflows and other Research Objects, and to build communities around them.
PRONOM		PRONOM is an information system about data file formats and their supporting software products. See https://www.nationalarchives.gov.uk/PRONOM
Pronom Unique Identifier	PUID	The PRONOM Persistent Unique Identifier (PUID) is an extensible scheme for providing persistent, unique and unambiguous identifiers for records in the PRONOM registry. Such identifiers are fundamental to the exchange and management of digital objects, by allowing human or automated user agents to unambiguously identify, and share that identification of, the representation information required to support access to an object. This is a virtue both of the inherent uniqueness of the identifier, and of its binding to a definitive description of the representation information in a registry such as PRONOM. From: http://www.nationalarchives.gov.uk/aboutapps/pronom/puid.htm
Pronom Signature File		Signature files are generated by PRONOM (see corresponding glossary entry) and used by DROID (see corresponding glossary entry) for file format identification. The signature file contains a subset of the information from the PRONOM knowledge base required by the DROID software to perform the file format identification. See https://www.nationalarchives.gov.uk/aboutapps/pronom/droid-signature-files.htm
Preservation Component	PC	See SCAPE Component
Quality Assurance Component		A Quality Assurance Component is used to determine a quality measure related to the outcome of applying an Action Service (see corresponding glossary entry) to a digital object.

Scalable Preservation Environments	SCAPE	An EU funded project developing scalable services for the planning and execution of institutional preservation strategies on an open source platform that orchestrates semi-automated workflows for large-scale, heterogeneous collections of complex digital objects.
SCAPE Characterisation Component		Characterisation components are a family of *SCAPE Components* (defined to wrap tools produced in WP9) that compute one or more properties of a *single* instantiated digital object or file. The output ports that produce measures are always annotated with the metric (in the *SCAPE Ontology*) that describes what the component computes.
SCAPE Component		SCAPE components are *Taverna Components*, identified by the SCAPE Preservation Components sub-project, that conform to the general SCAPE requirements for having annotation of their behaviour, inputs and outputs. SCAPE components may be stored in the *SCAPE Component Catalogue*.
SCAPE Local Instance		The local instance in the SCAPE project is an environment in a memory institution where the SCAPE Execution Platform together with SCAPE Preservation Components is deployed.
SCAPE QA Component		QA components are a family of *SCAPE Components* (defined to wrap tools produced in WP11) that compute a comparison between *two* instantiated digital objects or two files. They produce at least one output that has a measure of similarity between the inputs, and that output is annotated with the metric (in the *SCAPE Ontology*) that describes the nature of the similarity metric.
SCAPE Story		A short and succinct high-level statement of the preservation issue encountered by a partner institution.
Selenium		Selenium is a software framework to simulate and automate a great variety of web browsers. See http://docs.seleniumhq.org/
Taverna Components		Taverna components are *Taverna workflow* fragments that are stored independently of the workflows that they are used in, and that are semantically annotated with information about what the behaviour of the workflow fragment is. They are logically related to a programming language shared library, though the mechanisms involved differ.

		Taverna components are stored in a component repository. This can either be a local directory, or a remote service that supports the Taverna Component API (e.g., the *SCAPE Component Catalogue*). Only components that are stored in a publicly accessible service can be used by a *Taverna workflow* that has been sent to a system that was not originally used to create it.
Taverna Workbench		The Taverna Workbench is a desktop application for creating, editing and executing *Taverna workflows*.
Taverna Workflow		A Taverna workflow is a parallel data-processing program that can be executed by *Taverna Workbench* or *Taverna Server*. It is stored as an XML file, and has a graphical rendering.
Toolspec		An XML file written to a standard API that contains details of how to execute a tool for a particular purpose; for example txt2pdf might define how to use a command line tool to convert text to pdf. Toolspecs can have different types such as migration or QA.
Toolwrapper		The toolwrapper is a Java tool developed in the SCAPE Project to simplify the execution of the following tasks: Tool description (through the toolspec); Tool invocation (simplified) through command-line wrapping; Artefacts generation (associated to a tool invocation, e.g., Taverna workflow); and Packaging of all the generated artefacts for easier distribution and installation
(SCAPE) User story		A user story, in the context of the SCAPE Testbeds, is a description of a digital preservation issue related to a concrete data set that is used to derive requirements for tool development.
Wayback Machine		In the context of web archiving, the term Wayback Machine refers to a software used to render archived web pages, originally developed by the Internet Archive (http://archive.org).
Web Archive Record		Information unit contained in an ARC (see corresponding glossary entry) or WARC (see corresponding glossary entry) container file. This information unit, can hold, for example, the record payload (bitstream of HTML file or image) together with the HTTP response metadata and some additional metadata related to the record (date, checksum, etc.).
Web ARChive file format	WARC	The WARC (Web ARChive) file format offers a convention for concatenating multiple resource records (data objects), each consisting of a set of simple text headers and an arbitrary data

		block into one long file. See http://bibnum.bnf.fr/WARC/.
Web Crawler		Software used to capture and store web pages used by Web Archiving institutions to build their archives.
Web Content Testbeds	WCT	The Web Content Testbed is one of the Testbeds of the SCAPE project. The Testbeds are represented by memory institutions holding large data sets that are used to test the applicability of tools, workflows, and solutions developed in the SCAPE project.
Web Snapshot		The image capture of a web page that is taken when a web page is rendered in a web browser.