Quality Assurance Workflow, Release 2 + Release Report

Deliverable D11.2 of the SCAPE Project

Authors

Zeynep Pehlivan, Alexis Lechervy, Andres Sanoja, Denis Pitzalis (Universitè Pierre et Marie Curie, Paris), Bolette A. Jurik (The State & University Library Aarhus), Alexander Schindler (AIT Austrian Institute of Technology), Lynn Marwood (The British Library), Johan van der Knijff (The National Library of the Netherlands), Leila Medjkoune (Internet Memory Foundation), Natasa Milic-Frayling, Ivan Vujic (Microsoft Research Limited)

May 2013

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

Quality assurance is an essential aspect of preservation methodology. Its objective is to assess whether specific preservation goals are met. Thus, the objective of the PC.WP3 is to provide automated, scalable methods for quality assurance, applicable to a range of preservation workflows that are aimed at ensuring long term access to digital content. For example, after a migration operation of content from an obsolete format to a more up-to-date one, QA tools will ensure the quality of operation and detect the related errors if they exist.

In the past, large-scale quality assurance has relied on a combination of human intervention and statistical methods. In order to meet the challenges posed by large and heterogeneous digital collections, we will research and develop automated quality assurance approaches. This involves designing tools and methods to measure different properties of digital objects. Furthermore, comparison of digital objects often requires transforming digital objects into a common intermediary format that can be used for a specific analysis. By aggregating measurements from different methods we expect to produce rich characterization of digital objects and effective metrics from their comparison.

This document reports on the task of identifying, developing and testing Quality Assurance (QA) tools to be used in the SCAPE project. The report presents the work in the context of QA for different media types: Audio, Images, Web, Document, in addition, the knowledge based QA approach and QA tools. For each media type, this report contains the description of issue, proposed methods, deployment guides for the related tools and benchmarks. Correctness based benchmarking are released based on ground truth.

This deliverable will have a third version to be published on project month 42 and will include the scalability tests (Hadoop based) of the tools and the integration of the tools in the SCAPE platform.

Table of Contents

1 Introduction

The final objective of PC.WP.3 is to develop and assess methods for quality assurance that are as much as possible automated and scalable in terms of quantity of data and machines that can be used. To do so the work in the workpackage is not only to build the tools, but also to identify a range of preservation workflows that are aimed at ensuring long term access to digital content.

According to the description of work, the workpackage is split into six top-level tasks, while this document, will present our results "per media" more in line with the horizontal structure of the development.

The 6 tasks are listed as follows:

- Task 1 aims at producing quality assurance workflows that will integrate the tools developed in tasks 2, 3 and 4;
- Task 2 will output a range of tools capable of converting objects of various media types into a temporary format capable of being analysed and compared with another instance of that object, i.e. before and after any preservation action;
- Task 3 is intended to select and integrate appropriate analysis tools to measure the quality into the workflows defined in task 1;
- Task 4 provides new algorithms to quantifiably compare the converted objects;
- Task 5 will examine the possibility of automatically repairing errors that can appear during migration;
- Task 6 consists of validating and assessing conversion tools and processes. In the past, large-scale quality assurance has relied on a combination of human intervention and statistical methods.

Research activities are organized in "Quality Assurance modules": Audio, Image, Web and Document, corresponding to the media types object of the research project. Furthermore a section dedicated to a "Knowledge base" collecting all the different approaches, ideas and development linked to our research and another on "Quality Assurance tools", collecting the tools used to measure and wrap the tools developed during SCAPE are part of the project documentation.

The activities in the workpackage are carried out side by side with the other subprojects, i.e. with PW we share the common aim to define an interface to deliver QA measures to the planning and the watch components for assessment and successive reaction and feedbacks.

In order to meet the challenges posed by large and heterogeneous digital collections, we will research and develop automated quality assurance approaches. This involves designing tools and methods to measure different properties of digital objects. Furthermore, comparison of digital objects often requires transforming digital objects into a common intermediary format that can be used for a specific analysis.

2 Audio QA

2.1 Introduction

When working with audio, we need quality assurance (QA) when we migrate files. At SB Audio QA has also been used to detect overlaps and automatically cut recordings into single shows to enhance user experience. We also use QA for validating correctness of format and compliance with institutional policies when receiving new material.

The *xcorrSound*[1] tool suite is used in both overlap analysis and migration QA. The tools use cross correlation to compare sound waves and find the best overlap. The package contains several tools. *overlap-analysis* (earlier xcorrSound) is a tool to find the overlap between two audio files. By moving the first of the sound waves, it tests how well the two sound waves match, when the first sound wave starts at different sample numbers of the second sound wave. The tool outputs best, second best sample number and value of match. *waveform-compare* (earlier migration-qa) is a tool that splits two audio files into equal sized blocks (default 1 second) and outputs whether the two files are similar or not.

The xcorrSound overlap-analysis tool is used at SB for an annotation end-user web-tool for the DR (Danish Radio) broadcasts. The broadcasts were recorded on 2 hour tapes. In order not to lose any data, one tape was put in one recorder and a few minutes before it reached the end, another recorder was started with another tape. This yields two tapes with some overlap. All these tapes have now been digitized and the digitized dataset is 20Tbytes of audio files in 2 hour chunks with unknown overlaps. The library uses the overlap-analysis tool to find the exact timestamps for these overlaps, and then the files are cut and put together in 24 hour blocks. This is done to enhance the user experience in the annotation web-tool.

The xcorrSound waveform-compare tool is used in a more complex audio migration and QA Taverna workflow including *FFmpeg*[2] migration, *JHove2*[3] format validation, *FFprobe*[4] property extraction and comparison as well as *mpg321*[5] conversion. Over the last year, both the xcorrSound tool and the workflow have been improved.

2.1.1 Related Scenarios

The user story *Large Scale Audio Migration*[6] describes the audio migration scenario in outline. This is to be further elaborated, and our experiments are to be added.

The *LSDRT6 Large scale migration from mp3 to wav*[7] scenario describes the work in progress including issue and proposed solutions.

1 http://wiki.opf-labs.org/display/TR/xcorrSound
2 http://ffmpeg.org/
3 http://www.jhove2.org
4 http://ffmpeg.org/ffprobe.html
5 http://mpg321.sourceforge.net/
6 http://wiki.opf-labs.org/display/SP/Large+Scale+Audio+Migration
7 http://wiki.opf-labs.org/display/SP/LSDRT6+Large+scale+migration+from+mp3+to+wav

2.2 Tools and Interfaces

The tools used in our audio QA work now are primarily Taverna, JHove2, FFmpeg, FFprobe, mpg321 and xcorrSound waveform-compare. Taverna[8] is used throughout the SCAPE project and requires no further introduction.

JHove2 is a command-line tool available for download from http://www.jhove2.org. JHove2 requires you to be in its install directory when invoking it, which can be difficult, when invoking from another tool. In our workflow, we have a small "bug fix script", which can be called from any directory and run JHove2. We note that we have plans to replace JHove2 with a SCAPE profile validation component still in the planning phase, but expected to be available this fall.

FFprobe is part of FFmpeg, which is available in most Linux distributions. The SCAPE migration tool specification for mp3 to wav FFmpeg migration is available on Github[9]. There is also an FFprobe characterisation SCAPE tool specification[10]. The Debian packages will be available from OPF later in the year 2013.

The mpg321 MPEG audio player is used for conversion and is available in most Linux distributions. The SCAPE tool specification is available from Github[11]. The Debian package will be soon available from OPF.

The xcorrSound waveform-compare tool takes two wav files as input, and outputs 'success' if they are similar, 'failure' otherwise. Wav is a raw sound format, which means that the tool compares the raw sound waves, which are also the ones the human ear hears, when we play the sound files. When we use waveform-compare in migration quality assurance, we use two independent players to play or convert the original and the migrated file to wav files, which we then compare. If one of the files is already wav, one of the players becomes unnecessary (but the remaining player must be independent of the migration tool).

In waveform-compare the sound waves are compared in blocks. The block-size can be set as an optional input parameter. If the block size is large, a short audible mismatch may be missed. We have set the default block size down from 5 to 1 second in the correctness benchmarking. The algorithm compares the first waveform to the second using different sample offset 0, 1, 2, …, block-size. That means for each k, the second sound wave is shifted to start at sample offset k and then compared. Note however that if the sound is shifted by more than the block-size, the tool will not discover if they match.

The cross-correlation algorithm computes a match value between 0 and 1 for the two sound waves at a given offset. The offset giving the best value and the best value is kept. The best offset should be the same in all blocks, or the two sound waves are not similar. In the end the average match value for the best offset is compared against a threshold, which can also be given as input. The default

8 http://www.taverna.org.uk/

9 https://github.com/openplanets/scape-toolspecs/blob/master/digital-preservation-migration-audio-ffmpeg-mp32wav.xml

10 https://github.com/openplanets/scape-toolspecs/blob/master/digital-preservation-characterisation-video-ffprobe-video2xml.xml

11 https://github.com/openplanets/scape-toolspecs/blob/master/digital-preservation-migration-audio-mpg321-mp32wav.xml

match value threshold is set to 0.98 in the correctness benchmarking, as we require the migrated waveform to be very similar to the original to avoid sound quality loss.

The tool outputs the best offset and the match value when the result is 'Success'. When the result is failure, the irregular offset or the low match value is output.

The xcorrSound waveform-compare source code is available as GPL-2 from Github[12]. This includes guides for installing and running the tool. There is also a Debian package available from OPF. And there is a QA SCAPE tool specification[13]. The tool uses the FFTW3[14] Fast Fourier transform library, and inherits the GPL-2 licence from this program.

2.2.1 Deployment guides

xcorrSound waveform-compare: There is an xcorrSound Debian package available from OPF[15]. The new (May 2013) version is called scape-xcorrsound_0.9_amd64.deb. To install

```
sudo dpkg -i scape-xcorrsound_0.9_amd64.deb
```

On other systems download the source from Github and follow the instructions to "make waveform-compare". Note the old (April 2013) version was called migration-qa_0.9_amd64.deb.
FFmpeg: available as Debian package (includes FFprobe).

Mpg321: available as Debian package.

JHove2: available for download from http://www.jhove2.org.

2.3 Taverna Workflow

The full Taverna Workflows are available from myExperiment. The audio migration + QA workflow is called "Mp3ToWav Mig+QA"[16]. The "only QA" workflow used in the correctness benchmarking is called "Validate Compare Compare List"[17], as it validates migrated files using JHove2, compares properties extracted with FFprobe, compares content using waveform-compare and works on lists.

In detail the workflow takes a list of migrated files and a list of 'compare to files' as input and performs quality assurance using JHove2 to validate file format of the migrated files, FFprobe to extract properties from both 'compare to files' and migrated files, a Taverna beanshell to compare these, and waveform-compare for content comparison.

The workflows are also available within a Github project *scape-audio-qa*[18]. The Github project includes the workflows and some scripts to deploy and run the workflows from the command line including checks that the necessary tools are available.

12 https://github.com/openplanets/scape-xcorrsound
13 https://github.com/openplanets/scape-toolspecs/blob/master/digital-preservation-qa-audio-xcorrsound-migrationqa .xml
14 https://github.com/FFTW/fftw3
15 http://deb.openplanetsfoundation.org
16 http://www.myexperiment.org/workflows/2914.html
17 http://www.myexperiment.org/workflows/3521.html
18 https://github.com/statsbiblioteket/scape-audio-qa

To run the "Validate Compare Compare" QA workflow from the *scape-audio-qa* Github project, you need a list of migrated files and a list of 'compare to files'. You can then run

```
./bin/validateCompareCompare.sh <comparetowav_list> <migratedwav_list> $PWD
```

For further instructions see the project READMEs.

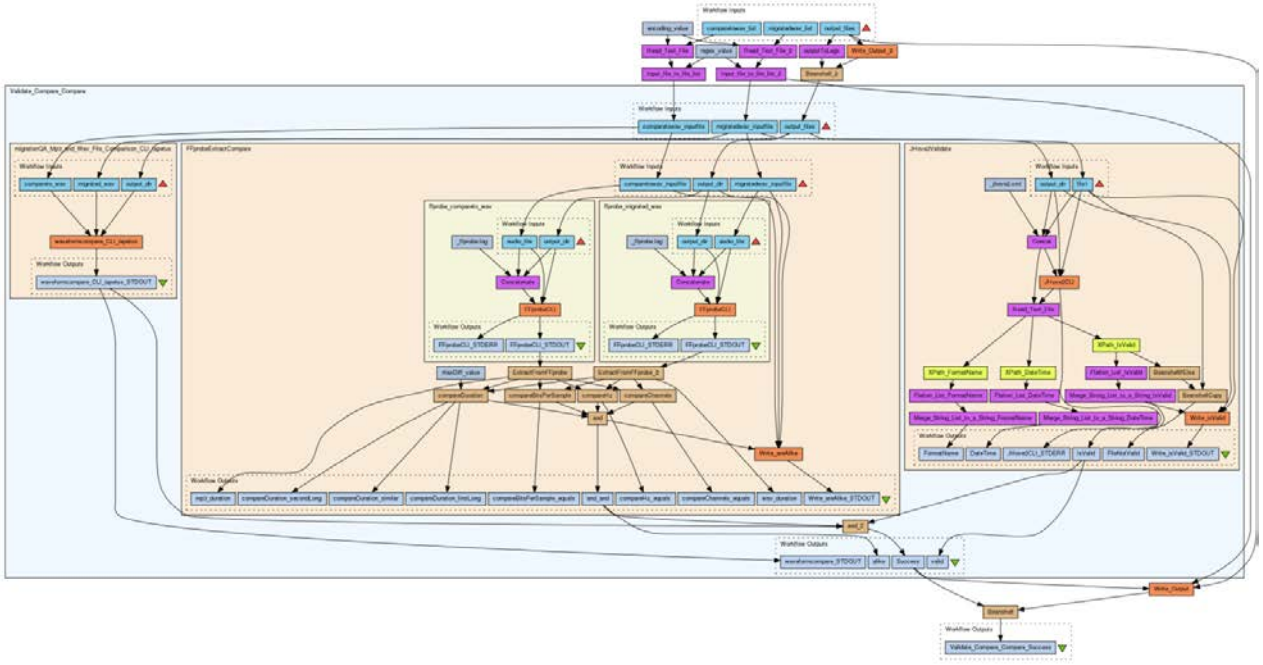

Figure 1: Audio QA workflow "Validate Compare Compare List"

2.4 Correctness based Benchmarks and Validation Tests

The challenge in a correctness baseline test set for audio migration QA is that migration errors are rare. In connection with large scale testbed experiments in November 2012, we did succeed in finding a migration error using waveform-compare. It turned out that this was caused by a bug in the version of FFmpeg, which we had used (0.6-36_git20101002). This bug had been fixed in the 0.10 version, which is the one we used for the benchmark. The error was a short bit of noise. This error is the one we test for with the file partly_garbage.wav (see Dataset Table below).

We also experienced that different conversions tools added different (not audible) short pieces of silence to the end of the files. The waveform-compare tool reported these as 'not similar', but we decided that these files are similar, and the tool was updated accordingly. Short bursts of silence in different places in the migrated file are tested using the file partly_silence.wav (see below).

2.4.1 Dataset

We decided that we needed a more diverse annotated dataset for the correctness benchmarking. This was accomplished by issuing an internal challenge at the Digital Preservation Technologies

Department at SB. The challenge was, given a correct wav file, to introduce a simulated migration error that our workflow would miss. This resulted in a number of very different test files.

The full simulated annotated dataset used for the correctness benchmarking consists of 28 test files and 2 comparison files along with annotations and is now available from Github[19].

Test file	Comparison	Annotation	Similar
challenge-KFC-2.wav	challenge.wav	One channel shifted a little less than 0.1 second; cut to original length and header updated to correct length	false
challenge-pmd.wav	challenge.wav	hidden jpg image in the least significant bits in the wave file using Mathematica; not audible to the human ear	true
challenge-TEG-1.wav	challenge.wav	File with random bits, same length as original	false
challenge-KFC.wav	challenge.wav	One channel shifted a little less than 0.1 second	false
challenge-TE-1.wav	challenge.wav	Audacity Amplify 0.1; not audible	true
challenge-TEG-2.wav	challenge.wav	File of double length, song repeated	false
challenge-KTC.wav	challenge.wav	normalize 0dB channel0; normalize -16dB channel1	false
challenge-TE-2.wav	challenge.wav	Audacity Compressor 10:1 + amplify -5; similar to radio broadcast quality loss	true
challenge-TEG-3.wav	challenge.wav	Metadata changed (author)	false
challenge-TE-3.wav	challenge.wav	Audacity Vocal remover + amplify 5	false
challenge-TEG-4.wav	challenge.wav	Echo effect in the beginning	false
Challenge-nbr-1.wav	challenge.wav	LAoE major changes in sound	false
challenge-TE-4.wav	challenge.wav	Audacity WahWah	false
challenge-TEG-5.wav	challenge.wav	Corrupt Wave-header	false
Challenge-nbr-2.wav	challenge.wav	LAoE major changes in sound	false
challenge-TE-5.wav	challenge.wav	Audacity AM Pitch Shifter 2	false
challenge-UKH.wav	challenge.wav	hex editor edit values beginning and end of file (hidden messages); audible as short scratches (milliseconds)	false
Challenge-nbr-3.wav	challenge.wav	LAoE major changes in sound	false
challenge-TE-6.wav	challenge.wav	Audacity Echo 1 second	false

19 https://github.com/statsbiblioteket/xcorrsound-test-files

challenge.wav	challenge.wav	Exact match	true
challenge-nbr-7.wav	challenge.wav	Free Audio Server minor not audible silence	false
challenge-TE-7.wav	challenge.wav	Audacity change pitch to A	false
challenge-KFC-3.wav	challenge.wav	One channel shifted a little less than 0.1 second; cut to original length and both file and stream header updated to correct length	false
DER259955_mpg321.wav	DER259955_ffmpeg.wav	Converted to wave using different tools	true
complete_garbage.wav	DER259955_ffmpeg.wav	one complete garbage	false
complete_silence.wav	DER259955_ffmpeg.wav	one complete silence	false
partly_silence.wav	DER259955_ffmpeg.wav	one partly silence (see also partly_silence_info.txt)	false
partly_garbage.wav	DER259955_ffmpeg.wav	one second garbage 3 places (see also partly_garbage_info.txt)	false

Table 1: Audio Dataset

All sound files with file names starting with 'challenge' are derived from a snippet of a Danish Radio P3 broadcast from October 31st 1995 approximately 8:13 till 8:15 am. The rest of the files are derived from a snippet of "Danmarks Erhvervsradio" (Danish Business Radio) from a show called "Børs & Valuta med Amagerbanken" from 8th of January 2002 from 8:45 till 9 am.

2.4.2 Experiments and Results

The experiences from the correctness benchmarks showed that the classification of files into similar and not similar is certainly debatable.

The complete audio QA correctness benchmarking results can be found in Appendix B.
The challenge-pmd.wav test file has a hidden jpg image in the least significant bits in the wave file. The difference in challenge-pmd.wav and challenge.wav is not audible to the human ear, and is only discovered by the waveform-compare tool. This tool compares the sound waves in two files, and tells us that the migration failed, if the sound waves are not similar. Here the hidden image has added so little noise, that the sound waves are similar both to the tool and the human ear. If the match threshold is set to at least 0.9999994 (the default is 0.98, and waveform-compare reports 'migration successful' with this threshold). We think these files are similar! This means that our tool does not always 'catch' hidden images. For the fun story of how to hide an image in a sound file, see http://theseus.dk/per/blog/2013/04/07/hiding-stuff/.

The challenge-TE-1.wav test file was made setting Audacity Amplify 0.1. The difference between challenge-TE-1.wav and challenge.wav is also not audible and only discoverable with threshold greater or equal to 0.99993, and we think they are similar.

The challenge-TE-2.wav test file was made setting Audacity Compressor 10:1 and amplify -5, which is similar to radio broadcast quality loss. The difference between challenge-TE-2.wav and challenge.wav is audible, but only discoverable with threshold>=0.99. The question is whether to accept these as similar. They certainly are similar, but this test file represents a loss of quality, and if we accept this

loss of quality in a migration once, what happens if this file is migrated 50 times? The annotation is similar=true, and this is also our test result, but perhaps the annotation should be false and the default threshold should be 0.99?

And then there is challenge-KFC-3.wav, where one channel is shifted a little less than 0.1 second, the file then cut to original length and both file and stream header updated to correct length. The difference here is certainly audible, and the test file sounds awful (imagine your left speaker being just a little bit behind your right speaker, when listening to something in stereo). The waveform-compare tool however only compares one channel (default channel 0) and outputs success with offset 0. The correctness benchmark result is thus similar=true, which is wrong. If waveform-compare is set to compare channel 1, it again outputs success, but this time with offset 3959. This suggests that the tool should be run on both (all) channels, and the offsets compared. This may be introduced in future versions of the workflows. Note that the DER files are all mono, and thus this issue does not arise.

Some settings are also relevant for the checks of 'trivial' properties. We for instance have a slack for the duration. This was introduced as we earlier have experienced that tools insert a short bit of silence to the end of the file in a migration. The default slack used in the benchmark is 10 milliseconds, but this may still be too little slack depending on the migration tool.

3 Document QA

3.1 Introduction

The objective of MSR participation in SCAPE is to provide automated, scalable methods for quality assurance that can be applied to format migration of documents created by *office productivity tools* such as Adobe publishing tools, Microsoft Office suite, Open Office applications, and similar.

Common practice in large-scale quality assurance efforts is to combine human intervention with statistical methods. However, large and heterogeneous digital collections would require extensive human effort. Thus, we need to put an effort in research and development of automated quality assurance approaches. We aim at designing tools and methods to measure different properties of the digital objects. That provides a basis for comparing digital objects. Comparison often requires transforming digital objects into a common intermediary format that can be used for a specific analysis. By aggregating measurements from different methods we expect to arrive at a rich characterization of digital objects and effective metrics for their comparison.

3.2 Tools and Interfaces: Format Migration and OCR-based Migration Analysis

This section provides an overview of the SCAPE architecture implementation on the Windows Azure platform. This solution makes use of a number of key Microsoft technologies including Silverlight, RIA Services, WCF, SQL Azure and Windows Azure detailed in Section 3.3.
Incremental work that has been done since last report on D11.1 mostly focuses work on:

1. Implementation of converters and chain conversions on the SCAPE Azure architecture
2. Defining algorithm for OCR analysis
3. Exposing metadata from conversions on SCAPE Portal UI

3.2.1 Implementation of converters and chain conversions on the SCAPE Azure architecture

SCAPE Azure architecture includes file migration services that use different format converters (Figure 2):

- Open source format translators (e.g., conversion of ODF to Open XML)
  - Office Binary Translator to Open XML – b2xtranslator (http://b2xtranslator.sourceforge.net)
  - Open XML to ODF Translator – ODF converter (http://odf-converter.sourceforge.net)
- Conversion features available through the SharePoint Word Automation Services (WAS) for transforming binary MS Office formats to OOXML
- Proprietary and licensed converters

	docx	Docx (B2X)	odt	docm	dotx	dotm	doc	dot	rtf	mht	mhtml	xml	png	dz	pdf	XPS
docx		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
odt	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
docm	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
dotx	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
dotm	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
doc	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
dot	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
rtf	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
mht	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
mhtml	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
xml	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
png														✓		
pdf													✓	✓		

- Word Automation Service Converters - blue
- Open Source Conversion tools red and orange
- Proprietary and licenced converters – green.

Figure 2: Format migration functions enabled within the SCAPE Azure architecture

The file migration services support chain conversions, i.e., application of multiple converters sequentially. For example, chaining of converters enables conversions into the DeepZoom (DZ)[20] formats as shown in Figure 3.

Figure 3: Chain migration process from the Word 2010 (Docx) to the DeepZoom (DZ) format

Migration from DOCX to DZ format is performed through several automated steps:

1. Ingest DOCX document
2. Schedule batch job in WAS, put document on the queue
3. Start WAS and convert document to PDF
4. Send document to PDF2PNG converter and convert each page as separate PNG
5. Create appropriate collection with PNGs that can be processed by DZ converter
6. Generate DZ file and store it on the BLOB (Microsoft Azure file system)

20 http://msdn.microsoft.com/en-us/library/cc645050(v=vs.95).aspx

3.2.2 Document Migration QA: OCR-Based Analysis

The architecture is extendable and enables easy incorporation of new conversion and comparison methods. The current implementation supports comparison of two documents based on several characteristics of the textual content and the document layout: the character count, passage count, page count, alignment of text boxes, etc. The comparison is achieved by:

- Generating an XPS version of the document through a printing service
- Applying OCR analysis to the resulting image
- Parsing OCR output to extract information about the layout and textual features of the document
- Computing comparison metrics based on the statistics from the OCR analysis.

In the current implementation, the *Comparison service* uses *WAS* to generate an XPS representation of each file. XPS files are processed by the *OmniPage* OCR engine[21] which produces XML representation of the OCR analysis, stored by the system. Further comparison analysis uses the OCR output and generates an XML file containing the coordinates and the style of the text rectangles with the mark up related to the comparison results. The two XPS files are displayed side by side. The text passages are annotated with the accuracy information contained in the comparison XML file.

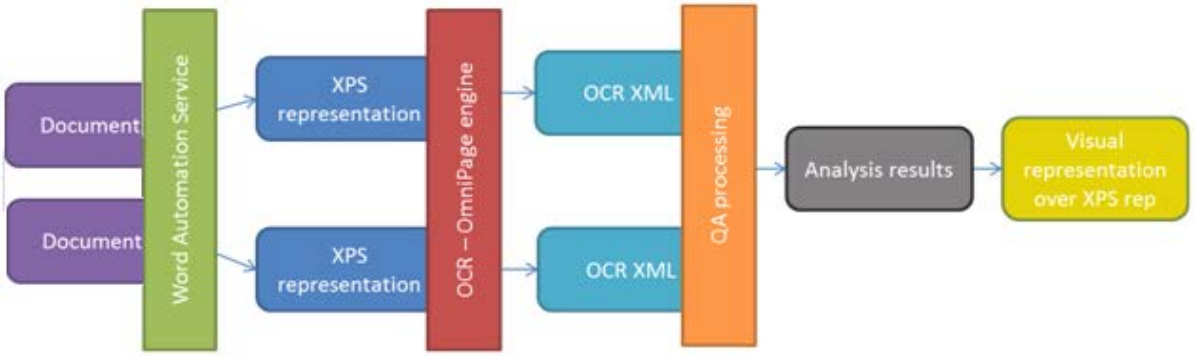

Figure 4: A schematic view of the document comparison process

3.2.2.1 Comparison Algorithm

Comparator utilizes OmniPage OCR tool for document scanning. It uses XPS fixed-layout documents as input and creates an XML output describing document properties. The XML output from the OCR process is simplified in a way that every paragraph is represented as a "rectangle" with its text and position on the page.

To compare two documents we do the comparison at the rectangle level. For each rectangle of one document we generate a set of potentially matching rectangles from the second documents. For that we consider all the rectangles in the second document that are located the same page, the preceding page, and the following page (essentially we use a three-page span to compile a list of possible matches).

For each pair of rectangles, we compute several comparison metrics: Levenshtein text comparison, *rectangle area comparison* and the *rectangle distance comparison*.

21 http://www.nuance.com/for-business/by-product/omnipage/csdk/index.htm

- The Levenshtein metric determines the level of similarity between two strings. It does so by computing the number of character operations (insertion, deletion and substitution) needed to transform one string to other.
- The rectangle area metric compares areas of the rectangle and its match candidates.
- The rectangle distance measures the (spatial) distance between a rectangle and its match candidates.

All comparison measurements are normalized to have values within [0, 1] interval.
We implemented the *ToleranceAnalyst* function to select the best match candidate based on the available comparison metrics. For example, two rectangles could have identical text content and area but their positions in the document may not be the same. In that case *ToleranceAnalyst* would dismiss the result and look for the next best match candidate.
The best match candidate is placed in the XML file next to the original rectangle, together with the match coefficient value, the text, size, and position of individual documents and their comparison values. The XML file is the input for *Comparison Viewer*.

```
      <rectangle-of-interest type="Paragraph" position-x="295" position-
y="385" height="168" width="1832">
      <page number="2" height="3510" width="2482" resolution="300" />
      <text>Thisdocumentdescribesthedocumentcomparisonframework.</text>
      <best-matches type="Paragraph" position-x="295" position-y="640"
height="155" width="1832" sum-weighting="3.9675035045240219" best-match-
quality="Good">
        <page number="2" height="3510" width="2482" resolution="300" />
        <text>Thisdocumentdescribesthedocumentcomparisonframework.</text>
        <weightings>
          <weighting type="Distance" value="0.9675035045240219" />
          <weighting type="Area" value="1" />
          <weighting type="Levenshtein" value="1" />
          <weighting type="Overlap" value="0" />
        </weightings>
      </best-matches>
    </rectangle-of-interest>
```

Figure 5: Sample XML code from the XML file encoding the comparison analysis

Figure 5 shows the XML code example: the rectangles of the document that have a matching pair in the other document are included in the resulting XML file. The rectangles in the first document and their counterparts in the second document are marked with the colour indicators. All the rectangles that do not have a matching pair are omitted from this viewer.
The colour is determined by the “best-match-quality” property from the XML file. Green for a “good”, match; red is for a “bad” match. These two values are added by the *ToleranceAnalyst*. The match is “bad” if the value of a given comparison method is lower than the best value of that method for all match candidates minus the predefined “tolerance”, which is here set to 0.35. These are initial settings that will be empirically evaluated and set based on testing over large collections of documents.

3.2.3 Exposing metadata from conversions on SCAPE Portal UI

Metadata is currently extracted from the file properties or generated through OCR comparison techniques (described in the previous section). The SCAPE Azure Portal provides two main views for displaying document metadata:

1. Document comparison view
2. Collection Pivot view.

3.2.3.1 Document comparison view

By selecting highlighting options, using check boxes from the *Show highlights* pane (*Match, Missing* and *Error)* one can see the match quality as a text overlay in the *Document Comparison View*.

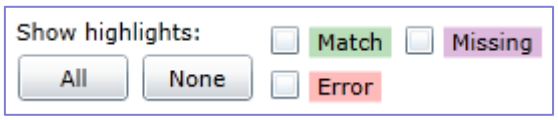

Figure 6: Highlight options in the document Comparison View.

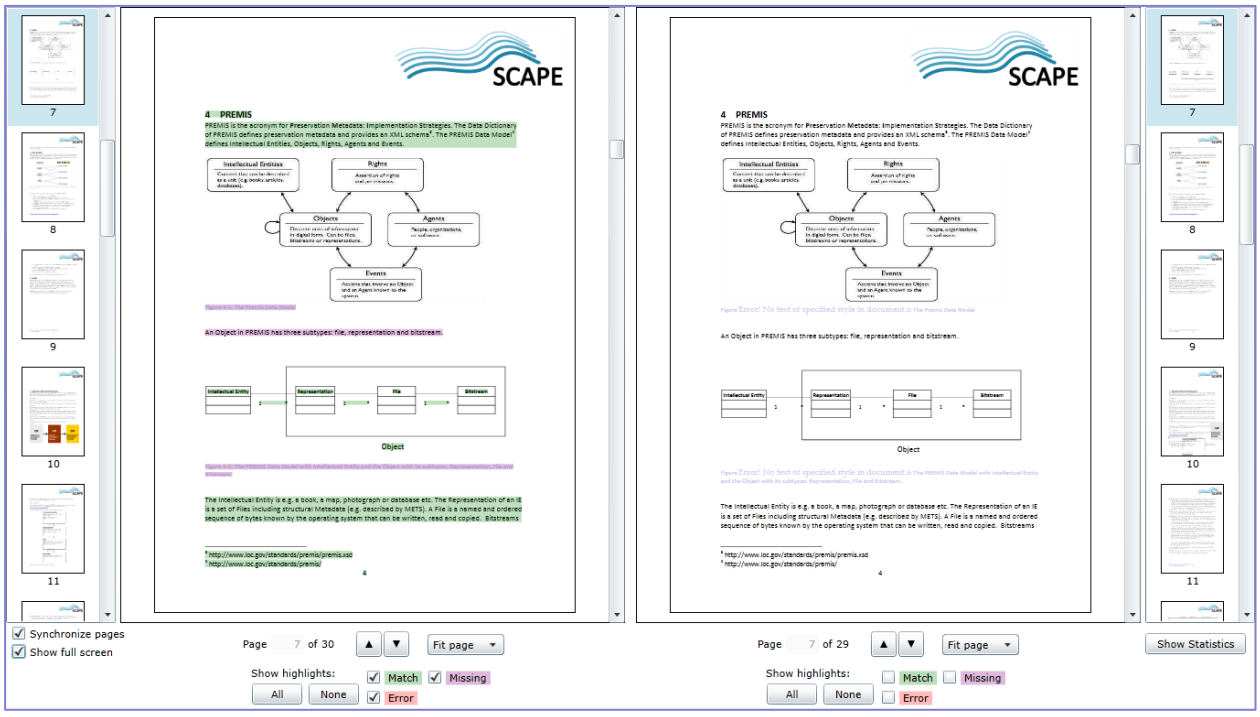

Figure 7: Document Comparison view

By clicking the *Show Statistics* button additional metadata is shown in the *Comparison Statistics* pop up:

- Matches (%)
- Errors (%)
- Missing (%)
- Pages, paragraphs, lines and words in uploaded file
- Pages, paragraphs, lines and words in B2X file

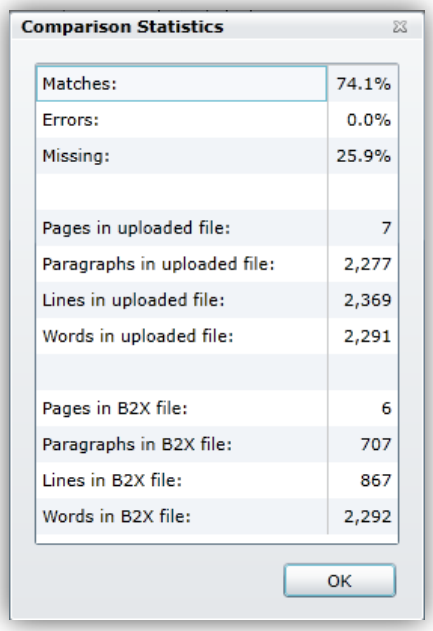

Figure 8: Comparison statistics displayed in the pop up window

3.2.3.2 Collection Pivot Viewer

Collection Pivot Viewer is an application that enables browsing, sorting, and filtering of a large collection of objects represented as images within a zoomable canvas.
In the *Collection pivot* view, following metadata is shown:

- Size
- Author
- Date created
- Page, paragraph, line and character counts
- Title
- Company
- Subject
- Page, paragraph, line and word count differences
- Good, bad partial and no match qualities (%)
- Average mismatched rectangles per page

More information about the Pivot Viewer control can be found in *External libraries* section below.

3.3 Development guides: Tools and External Libraries

3.3.1 Tools

The tools needed to develop and run the Azure services include:

- Visual Studio 2010
- Silverlight 4 Tools
- WCF RIA Services
- Expression Blend (optional)
- Silverlight Toolkit
- Windows Azure Tools & SDK
- SQL Server 2008 R2
- Windows Azure Free Trial

Figure 9: Collection Pivot View showing the metadata associated with the selected document

Silverlight

We have developed a Silverlight application for comparing two documents. During the implementation we faced the issue of secure access to the cloud data. We noted that the application could communicate directly with the BLOB storage system (primarily for file upload) via a REST API, without having to go via a service on the server. The typical way to access BLOB storage is to construct a web request and then sign that request with the storage account's shared key. This model is fine when the shared key is embedded within a trusted code. However placing the shared key within a Silverlight application makes it possible for anyone to extract the key and take full control of the storage account.

To address this issue, one option is to proxy all requests from the client via a web service. However, then all requests must be transferred via that service and that can potentially become a bottleneck. The solution to this problem is to use **Shared Access Signatures** (SAS) which allow for access rights to be provided to the containers and blobs at a more granular level instead of simply setting a

container's permissions for public access. Using SAS allows for separation of the code that signs the request from the code that executes it. Thus, the users can be granted access to a specific blob or any blob within a specific container for a specified period of time.

WCF RIA Services

WCF RIA Services are used for data management (CRUD) and user authentication, the data store being an instance of SQL Azure. User authentication is provided by the ASP.NET membership system and the RIA authentication client is configured to use it via forms authentication.
Worker roles are used to self-host WCF services which perform conversion / OCR / pagination (much like a console app or windows service). It should be noted that the TCP protocol cannot be used if Web Roles (IIS7 and SVC endpoints) are used to host the service; the speed gains, in exchange for the management of the service, is acceptable in this situation. The endpoint types are **internal** and, as such, do not hook into the load balancer. A simple randomizer is used to select the endpoint to use. Input endpoints would require security and be externally facing.

Local storage

Local storage refers to the file system directory on the server on which a particular role instance is running. The contents of the local storage are not persistent across instance failures and restarts as the role may be redeployed to another virtual machine due to a failure or restart. It cannot be shared between role instances but it is faster than other storage mechanisms as it resides on the same VM as the role instance is running. The size is limited to the size of the VM as specified for the service model (smallest is 250GB) which can also be configured via the property pages for the role.
A possible alternative to using the local storage system is to make use of **Windows Azure Drive** which allows Azure applications to use existing NTFS APIs to access a durable drive in the cloud. They are implemented as Azure Page Blob containing an NTFS-formatted Virtual Hard Drive (VHD). Since the drive is an NTFS formatted page blob the standard blob interfaces can be used to upload / download the VHDs to the cloud; speed however would be a consideration for this scenario.

Windows Azure

Windows Azure supports a **native execution**, allowing role hosts to PInvoke into system Dll's or to a proprietary packaged native DLL. There is however a caveat to this: Windows Azure runs on a 64bit OS and role hosts (both worker and web) are also 64 bit executable and as such will run as 64 bit binaries. This means that to PInvoke to a native DLL it must be compiled for a 64 bit platform. That, however, is not always an option (for example if the original source code is not available). The solution is to host the 32bit DLL in a 32bit program that runs on WoW (windows on windows). WCF is then used to marshal a call between the role process and the DLL. The two processes can both access local storage as they are executing on the same VM and communicate via named pipes for the same reason. Note that, at the time of writing, in order for WCF to set up a local communication pipe, the hosting application must be built against .NET 3.5.

Development Environment

OS: Windows 7
IDE: MS Visual Studio 2010
External libraries and tools: detailed below

Visual Studio, Silverlight 4 Tools, Silverlight Toolkit and WCF RIA Services will need to be installed before solution is deployed. The solution can be run locally, within the Azure development environment, if a live implementation is not necessary or desirable. Also, one needs to configure a local SQL server to provide membership information if that option is taken (although it is also

possible to make use of SQL Express with VS2010). Please refer to this article for more details on setting up the membership schema in SQL server: http://www.asp.net/security/tutorials/creating-the-membership-schema-in-sql-server-vb.
One should be cognizant of an important issue when using the development environment in combination with Silverlight and REST. Within development storage, Silverlight will be unable to find the client access policy since $root (where the access policy would be placed in the cloud) is actually located at http://127.0.0.1:10000/devstoreaccount1/$root.
In order to overcome this limitation we use of *Ag.AzureDevelopmentStorageProxy* which forwards request correctly. We use the port 11000 instead of 10000, as in http://127.0.0.1:11000/$root; it can be downloaded from: http://agazuredevstoreproxy.codeplex.com/.

3.3.2 External libraries

Silverlight 4 Tools
Silverlight is a development platform for creating interactive user experiences on the Web, desktop, and mobile applications when online or offline. Silverlight is a plug-in, powered by the .NET framework and compatible with multiple browsers, devices and operating systems. Silverlight 4 adds features like webcam, microphone, and printing.

- Version: 4
- Homepage: http://www.silverlight.net/
- License: n/a

Expression Blend (optional)
Microsoft Expression Blend is a user interface design tool for creating graphical interfaces for web and desktop applications that blend the features of these two types of applications. It is an interactive, WYSIWYG front-end for designing XAML-based interfaces for Windows Presentation Foundation and Silverlight applications. It is one of the applications in the Microsoft Expression Studio suite. Expression Blend supports the WPF text engine with advance OpenType typography and ClearType, vector-based 2D widgets, and 3D widgets with hardware acceleration via DirectX.

- Version: 4
- Homepage: http://www.microsoft.com/expression/products/Blend_Overview.aspx
- License: n/a
- License-URL: http://download.microsoft.com/documents/useterms/Expression Blend_2_English_81c9717a-ad3c-4f43-a018-dc2ffec80107.pdf

Silverlight toolkit
Silverlight Toolkit is an open source project that shares new components and functionality for designers, developers, and the community. Its aim is to facilitate efficient ways of product development. Toolkit releases include full open source code, samples, documentation, and design-time support for controls, focusing on both Silverlight 4 and the Windows Phone.

- Version: February 2011
- Homepage: http://silverlight.codeplex.com/
- License: Microsoft Public License (Ms-PL)
- License-URL: http://silverlight.codeplex.com/license.

Windows Azure Tools & SDK
The Windows Azure Platform is a Microsoft cloud platform used to build, host and scale web applications through Microsoft data centers. Windows Azure Platform is classified as a “platform as a

service" and forms part of Microsoft's cloud computing strategy, along with their software as a service offering, Microsoft Online Services. The platform consists of various on-demand services hosted in the Microsoft data centers: Windows Azure (an operating system providing scalable compute and storage facilities), SQL Azure (a cloud-based, scale-out version of SQL Server) and Windows Azure AppFabric (a collection of services supporting applications both in the cloud and on premise).

- Version: April 2011
- Homepage: http://www.microsoft.com/windowsazure/sdk/
- License: n/a

PivotViewer Silverlight control

PivotViewer makes it easier to interact with massive amounts of data on the web in ways that are powerful and informative. By visualizing thousands of related items at once, users can see trends and patterns that would be hidden when looking at one item at a time.

Because PivotViewer leverages Deep Zoom, it displays full, high-resolution content without long load times, while the animations and natural transitions provide context and prevent users from feeling overwhelmed by large quantities of information. This simple interaction model encourages exploration and applies broadly to a variety of content types.

- Version: October 2012
- Homepage: http://www.microsoft.com/silverlight/pivotviewer/
- License: n/a.

3.4 Workflow

As the first step towards a document preservation workflow, we have designed and implemented an interactive workflow for conversion and comparison of office documents.

Which is extended and now the workflow supports five main types of activities:

1. Authentication
2. Ingest and management of the collections
3. Conversion process
4. Comparison of the converted documents
5. Reporting and analysis

The workflow supports three main types of activities: (1) management of the collections – importing, previewing and selecting documents for conversion, (2) conversion process – selecting the operators for the conversion, and (3) analysis of the converted data – selecting the comparison tools that provide conversion analysis and viewing the comparative reports.

Using MS Tools, we have implemented a user interface that supports interactive document conversion and analysis workflow. The Web client connected to the SCAPE Azure services supports batch mode content conversion. The system is designed to record data processing information and use it to provide reporting services.

Information about the difference between two documents is provided based on the analysis of the rendered documents. They are indicated on the document viewers. Furthermore, statistics on various aspects of the documents are presented and compared.

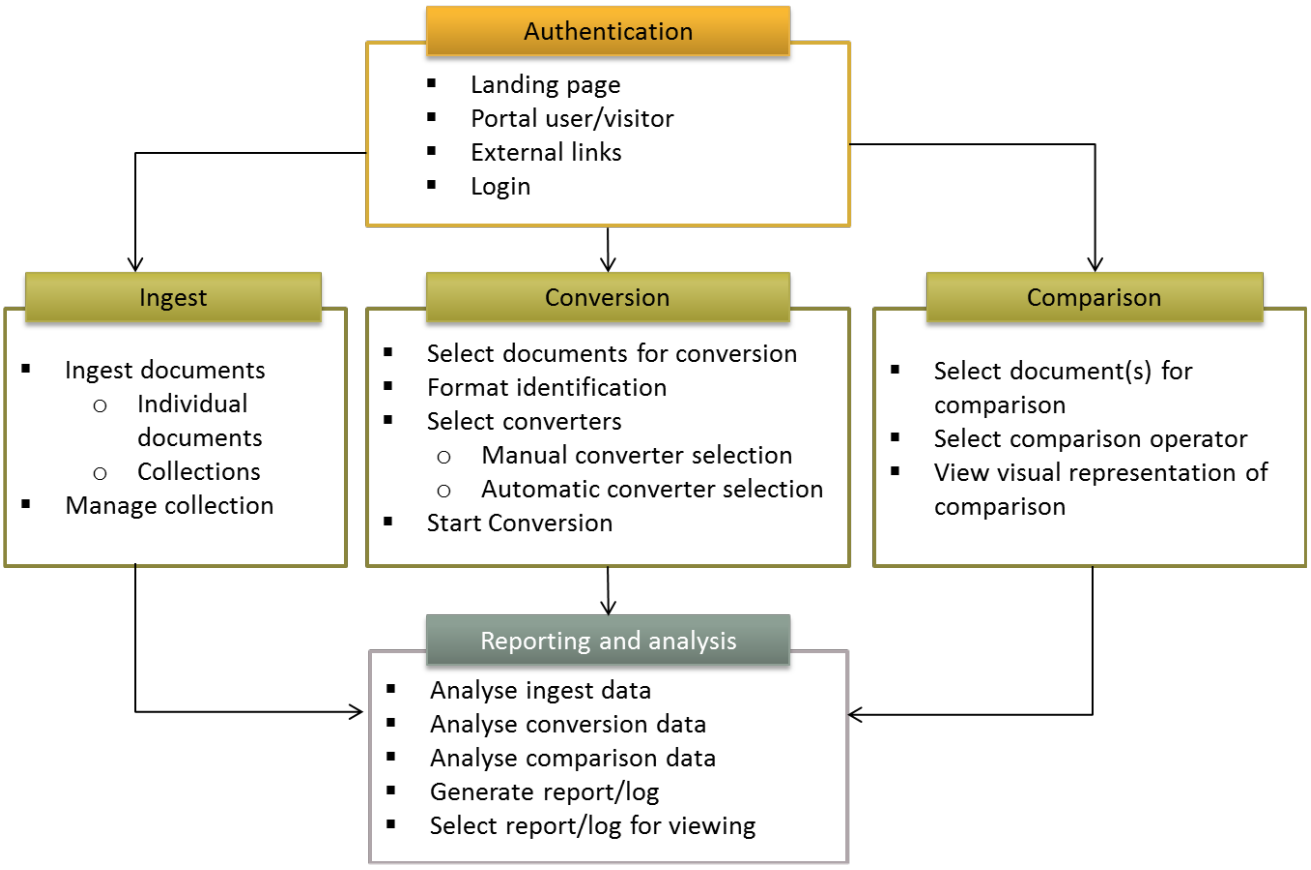

Figure 10: Document preservation workflow supported in SCAPE Azure

3.5 Correctness based Benchmarks and Validation Tests

MSR used the Windows Azure VM Role to run applications in Azure. The VM Role feature provided better control of the operating system and applications running within a virtual machine. However, the Windows Azure VM Role is not supported by Azure any more. Instead, the applications now need to run using Windows Azure Virtual Machine. This requires changes to the existing implementation. Please find more information in this MSDN article: http://msdn.microsoft.com/en-us/library/windowsazure/dn133483.aspx.

For that reason, the performance testing and benchmarking work is postponed after the migration to Windows Azure Virtual Machines.

3.5.1 Dataset

For performance testing we have prepared a data corpus comprising 1,000,000documents that were http://digitalcorpora.org/corpora/filesextracted from the Web crawl of the US Government sites www.digitalcorpora.org.

The dataset has been analysed using the FITools product[22]. The output is considered a 'ground truth' file format analysis[23]. Furthermore, we have a more comprehensive FITools analysis of every file in the corpus [24] and MD5 hashes for first 512 bytes and 4096 bytes of every file in GOVDOCS1 corpus[25] The analysis of the NPS GOVDOCS1 dataset does contain errors which need to be addressed during testing:

- File extensions may not reflect the type of the original file. For example, many files that were labelled '.xls' did not contain Microsoft Excel spreadsheets. Instead, they contained HTML error messages from US government web servers indicating that the file was no longer available.

- File extension chosen when the document was created no longer matches the current usage of the extension. For example, several files that had a '.doc' extension, typically used by MS Word, are actually WordPerfect files.

22 http://www.forensicinnovations.com/fitools.html
23 http://digitalcorpora.org/corpora/files/govdocs1-simple-statistical-report
24 http://digitalcorpora.org/corp/nps/files/govdocs1/groundtruth-fitools.zip
25 http://digitalcorpora.org/corp/nps/files/govdocs1/govdocs1-first512-first4096-docid.txt

4 Image QA

4.1 Introduction

National libraries maintain huge archives of artefacts documenting a country's cultural, political and social history. Providing public access to valuable historic documents is often constrained by the paramount task of long-term preservation. Digitization is a mean that improves accessibility while protecting the original artefacts. Assembling and maintaining such archives of digitized book collections is a complex task. It is common that different versions of image collections with identical or near-identical content exist. Due to storage requirements it might not be possible to keep all derivative image collections. Thus the problem is to assess the quality of repeated acquisitions (e.g. digitization of documents with different equipment, settings, lighting) or different downloads of image collections that have been post processed (e.g. de-noising, compression, rescaling, cropping, etc.). A single national library produces millions of digitized pages a year. Manually cross-checking the different derivates is an almost impossible endeavour.

Traditionally this problem was approached by using Optical Character Recognition (OCR) to compare the textual content of the digitized documents. OCR, though, is prone to many errors and highly dependent on the quality of digitized images. Different fonts, text orientations, page layouts or insufficient pre-processing complicate the process of text recognition. It also fails on handwritten and especially foreign, non-western texts. Pixel-wise comparison using digital image processing is only possible as long as no operations changing the image geometry, e.g. cropping, scaling or rotating the image were applied. Furthermore, in cases of filtering as well as colour or tone modifications the information at pixel level might differ significantly, although the image content is well preserved (see Figure 11).

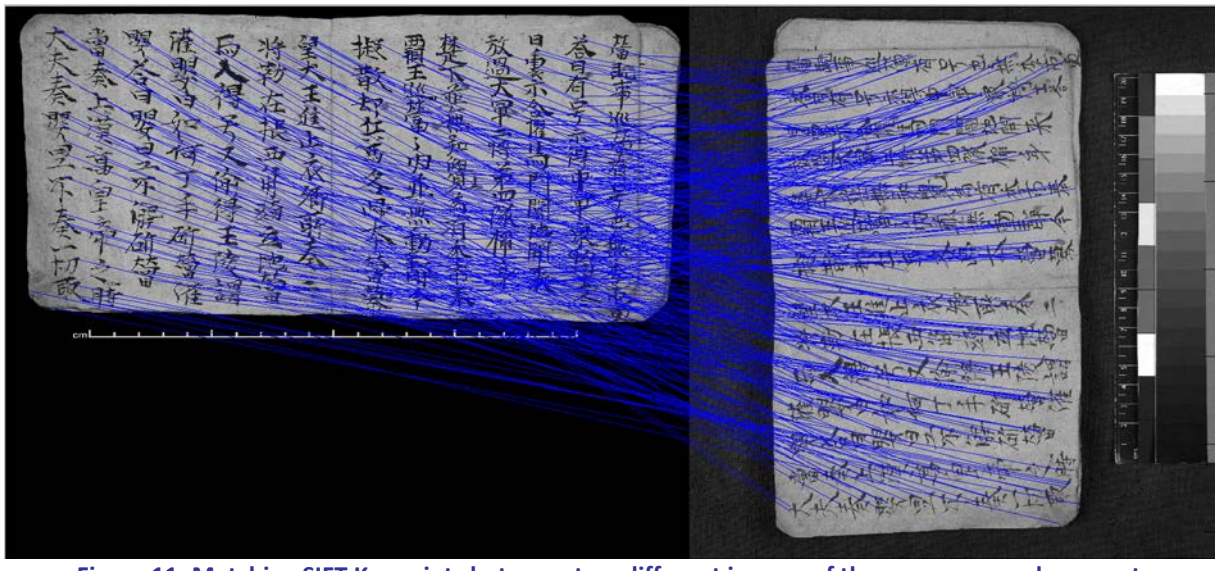

Figure 11: Matching SIFT Keypoints between two different images of the same source document

The provided Matchbox toolset can be used to efficiently detect corresponding images between different image collections as well as to assess their quality. By using inter point detection and derivation of local feature descriptors, which have proven highly invariant to geometrical, brightness and colour distortions, it is possible to successfully process even problematic documents (e.g. handwritten books, ancient Chinese texts).

Scale-invariant Feature Transform (SIFT) [LOWE] is an algorithm widely used in digital image processing to detect descriptive local features of images that can be used to identify corresponding points in different images. Such points are generally located at high-contrast regions. Thus, this approach is ideally for document processing because high contrast is an intrinsic property of readability. In order to be able to identify the correspondence of images between two different collections without reference to any metadata information, a comparison based on a visual dictionary, i.e. a Visual Bag of Words (BoW) approach is suggested. Following this approach a collection is represented as a set of characteristic image regions. Ignoring the spatial structure of these so called visual words, only their sole appearance is used to construct a visual dictionary. Using all descriptors of a collection the most common ones are select as a representative bag of visual words. Each image in both collections is then characterized by a very com-pact representation, i.e. a visual histogram counting the frequencies of visual words in the image. By means of such histograms different collections can be evaluated for correspondence, duplication or absence of images. A further option is the detection of duplicates within the same collection, with the exception that a visual dictionary specific to this collection is constructed beforehand. It is worth to mention that the Visual BoW is related to the BoW model in natural language processing, where the bag is constructed from real words.

Having solved the image correspondence problem using the visual BoW, a detailed comparison of image pairs, i.e. an all-pairs problem with respect to all pairs of possibly matching descriptors be-tween two images, is performed. Corresponding images are processed using descriptor matching, affine transformation estimation, image warping and computation of the structural similarity between the images.

4.1.1 Related Scenarios

LSDRT11 Duplicate image detection within one book

Due to specific processes in a digital book production process (e.g. different scanning sources, various book page image versions, etc.), it can occur that book image duplicates are introduced into the compiled version of a digital book.

The issue presented here is the need to identify books within a large digital book collection that contains duplicated book pages, and to know which book page images are actually duplicate book page image pairs.

Further information:

http://wiki.opf-labs.org/display/SP/LSDRT11+Duplicate+image+detection+within+one+book

LSDRT12 Quality assurance in redownload workflows of digitised books

The production of digitised versions of books or newspapers usually included that results and any attached property rights had mostly been transferred entirely to the originator's institution.

But public-private partnerships exist, where the digitisation is carried out by the commercial partner who keeps the original master copy and then produces derivates which are provided to the cultural heritage institution. New derivates of master copies are continuously being made available. The derivates can be downloaded and ingested into the repository as a new version which is either added or replaces the original derivate.

Quality assurance is responsible to assess the quality of the new derivates and to provide means for decision making if to keep or to abandon the newly available version.

Further information:

http://wiki.opf-labs.org/display/SP/LSDRT12+Quality+assurance+in+redownload+workflows+of+digitised+books

LSDRT13 Potential bit rot in image files that were stored on CD

Digitised master image files (TIFFs) from a legacy digitisation project were stored for a number of years on CD. Corresponding service/access images (JPEGs, at a lower resolution, cropped, scale added, and colour balanced) were stored on a web server during this period. Consequently there is a higher confidence in the bit integrity of the service copies. Without checksums, the only method of checking the master images for bit rot is to open each one and visually inspect it.

Also this issue aims at supporting of digital preservation quality assurance. It handles the image based document comparison challenges like detection of differences in file format, colour information, scale, rotation, resolution, cropping, slight differences in content.

Further information:

http://wiki.opf-labs.org/display/SP/LSDRT13+Potential+bit+rot+in+image+files+that+were+stored+on+CD

LSDRT18 Large scale digital book ingest

The Austrian Books online project of the Austrian National Library is currently creating a digital book collection from the 16th to the 19th century of about 600.000 books that will be digitised over the coming years.

This project states problem concerning Scalability issues, because image pairs can be compared within a book (book pages can be duplicated or missing) and between different master scans. Automated decision making support is needed for overwriting existing collection items with new items based on quality criteria.

Further information:

http://wiki.opf-labs.org/display/SP/LSDRT18+Large+scale+digital+book+ingest

4.2 Tools and Interfaces

No specialized or general solution has been known that could have been used to realize the required subtask of the proposed solution. Thus, a specialized toolset has been implemented that can be used in various scenarios and workflows. A special focus was set on adaptability concerning different scenarios and scalability.

4.2.1 Matchbox Toolset

The Matchbox Toolset is a collection of small command line tools that are intended to work independently.

These tools comply with the standard pattern of a typical retrieval task. At first descriptive information has to be extracted from unstructured data. In this case this corresponds to extracting information from images. These so called extracted features are typically numbers in vector format and are the basis for further calculations. The previously unstructured and huge amounts of data have been transformed into a condensed vector space representation upon which various machine learning techniques can be applied for further processing. Depending on the scenario these techniques can be used to classify the images into different categories, find duplicates or compare different collections.

The Matchbox Toolset consists of three executables: extractfeatures, train, compare. Each of it is responsible for a certain task within a workflow that is described by a scenario.

extractfeatures

```
extractfeatures
USAGE:
   extractfeatures  [--bowidx <int>] [--bow <filename>] [--downsample
                    <int>] [--binaryonly] [--binary] [--clahe <int>] [--sdk
                    <int>] [--numbins <int>] [-o <string>] ...  [-n
                    <string>] ...  [-l <int>] [-v] [-d <string>] [--]
                    [--version] [-h] <file>
```

The command line tool extractfeatures can be used to extract different image processing features and store them in a machine readable format. By default extracted data is stored in zipped xml format to maintain cross-compatibility with other tools.

The following features are supported by Matchbox:
Image Metadata: A small set of image metadata is extracted. The emphasis on this feature set has been dropped because this work package provides further tools that specialize on metadata.
Image Histogram: These features are simple colour histograms that are traditionally used in image processing to describe the distribution of colour within the image.
Image Profile: Image profiles are vertical and horizontal projections and represent the average colour intensity value for each pixel row or column.
SIFT: Scale Invariant Feature Transforms describe local features in images that are invariant to changes in scale, rotation, illumination and viewpoint.
BoW Histogram: Bag of Words Histograms are subsequently created from mapping all SIFT key points of an image to a previously calculated visual vocabulary.

train

```
train
USAGE:
   train  [-v] [-i] [-b <int>] [-p <int>] [-f <string>] -o <string> [--]
          [--version] [-h] <string>
```

This command line tool is used to create the visual vocabulary. It takes SIFT features as input that have been previously extracted from each image of a collection using the *extractfeatures* tool. *train* loads these features and create a visual Bag of Words (BoW). By applying a clustering algorithm that identifies a given number of cluster centres within a multidimensional vector space, the tool creates the descriptive vocabulary from these centres and stores it to a XML-file.

compare

```
compare
USAGE:
   compare  [--bowmetric <str>] [--binary] [--sdk <int>] [--metric <str>]
            [-v] [--] [--version] [-h] <file1> <file2>
```

This command line tool can be used to compare images by their extracted features. Depending on the type of features used for comparison, either distance metrics or similarity measures are calculated.

The results are outputted to standard output which is typically the screen. This output can be relayed into another tool (e.g. decision making)

4.2.2 Deployment guides

Currently two ways of obtaining the Matchbox Toolset are provided:

4.2.2.1 Building Matchbox from source

This is the most flexible approach and provides the best solution for all systems. Matchbox uses the build tool CMake[26] which enables cross compiling on different operating systems. Thus, Matchbox can also be built for Windows and OSx systems, if all dependencies are satisfied.
To build the Matchbox Toolset from source, download the sources from Github[27].
The Matchbox sources are in the pc-qa-matchbox sub-directory. Change to this directory and run CMake to create all necessary build files required by your build system to compile and install the binaries. It may be necessary to install further libraries Matchbox depends on in advance.
Further instructions on how to compile Matchbox from source are provided in the INSTALL file in the source tree.

4.1.2.1 Installing Matchbox on Linux from a Debian package

Debian packages are provided to conveniently install the Matchbox Toolset on a Linux operating system. These packages are intended to distribute Matchbox with minimal effort on a cluster with multiple nodes. Two packages are provided. One installs only the Matchbox binaries and expects all required libraries to be available or to be installed by the package manager. The second package also contains precompiled binaries of dependant third party libraries that are installed with Matchbox.

4.3 Taverna Workflow

The Taverna workflow engine wraps the Matchbox tool installed on remote Linux VM server for duplicate search in digital document collection. The result of duplicate search is returned in text format. This tool is a part of Matchbox Toolset, is written in C++ and provided as a Python script using associated DLLs on Windows or shared objects on Linux.

An example of a Taverna workflow[28] (see Figure 12) for the 'FindDuplicates' tool has two input parameters: the URL to the digital document collection and the parameter 'all'. In this scenario Matchbox will find duplicates in passed digital collection. All Matchbox workflow steps are executed automatically in one turn. User will get a list of duplicates in result. Matchbox tool support Python in version 2.7. Execution starts from the directory where python scripts are located. If you use source code from Github[29] then it is a scape/pc-qa-matchbox/Python/ directory. The python script supports different parameter. In this workflow the default parameter is 'all' in order to execute the whole workflow in one turn. Matchbox in this scenario is installed on remote Linux VM. Digital collection is stored on Windows machine. The output is a text data that comprise found duplicate pairs. Also the workflow provides output of return code (which is equal to 0 if completed successful). The presented

26 CMake – Cross Platform Make - http://www.cmake.org
27 https://github.com/openplanets/scape
28 http://www.myexperiment.org/workflows/3309.html
29 https://github.com/openplanets/scape/tree/master/pc-qa-matchbox,

scenario is designated for duplicate search in one turn in order to make the using of Matchbox tool possibly simple for regular user.

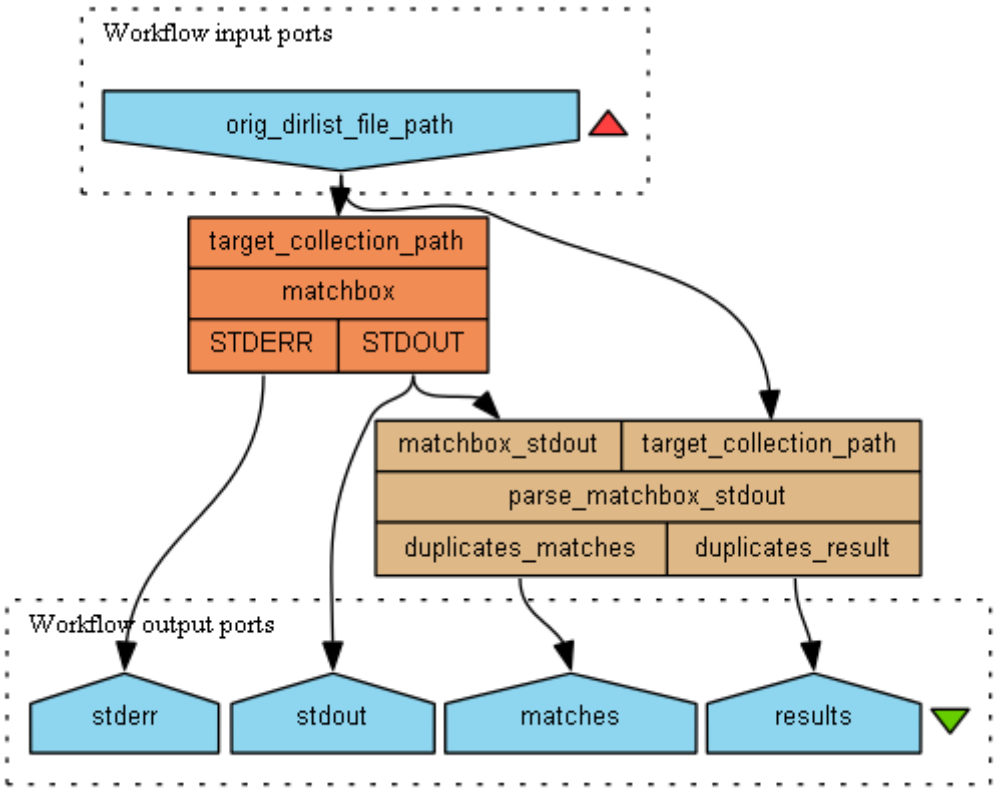

Figure 12: Taverna Workflow for 'FindDuplicates' tool

In order to provide more sophisticated professional scenario we designed the Taverna workflow[30]. In this scenario Matchbox will find duplicates in passed digital collection employing additional parameter. Each Matchbox workflow step can be executed separately. User will get a list of duplicates in result. This workflow starts duplicate finding process using the 'FindDuplicates' python script of the Matchbox tool. The Python script supports different parameter. Experienced user can apply 'clean', 'extract', 'train', 'bowhist' and 'compare' parameters in order to execute associated step in the Matchbox workflow for duplicate search. The order of execution steps should not be changed, because each next step requires an output from a previous step. E.g. if you are going to repeat the comparison step you should have calculated required BOWHistogram files from bowhist step.

The nested Taverna workflow[31] in Figure 13 is a workflow that implements the professional scenario described above and allows additional changes and extensions in every workflow step.

30 http://www.myexperiment.org/workflows/3308.html
31 http://www.myexperiment.org/workflows/3309.html

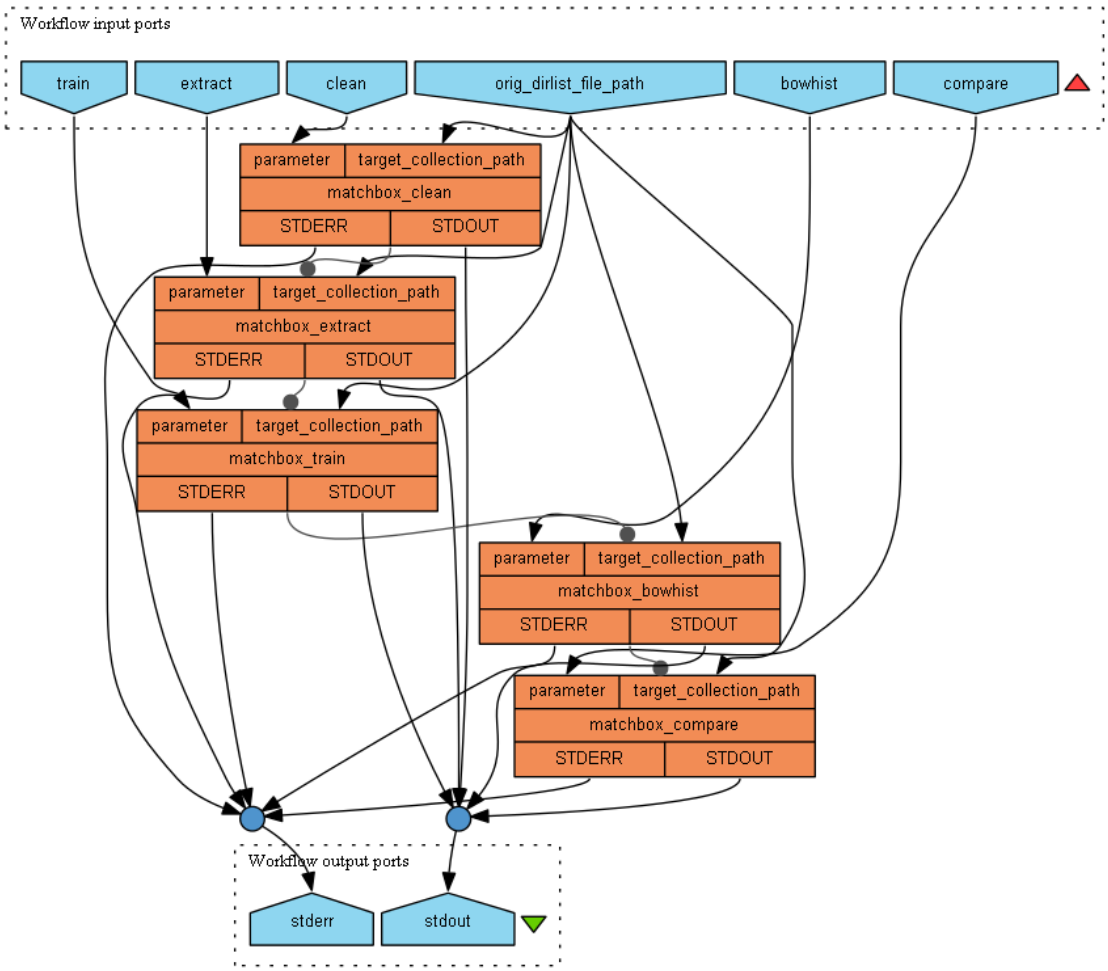

Figure 13 : Taverna Workflow for 'FindDuplicates' tool using nested commands.

4.4 Correctness based Benchmarks and Validation Tests

Correctness of the Matchbox toolset will be evaluated on scenario *LSDRT11 Duplicate image detection within one book*.

The corresponding workflow has been implemented as an executable script in the programming language Python. The workflow is described in Figure 14. The dataset described in Section 4.4.1 provides ground truth data that lists duplicates for each collection. The implemented workflow will process every collection using the Matchbox toolset and searches for duplicates. The result will be evaluated against the ground truth data. Accuracy will be based on how many duplicates are detected correctly.

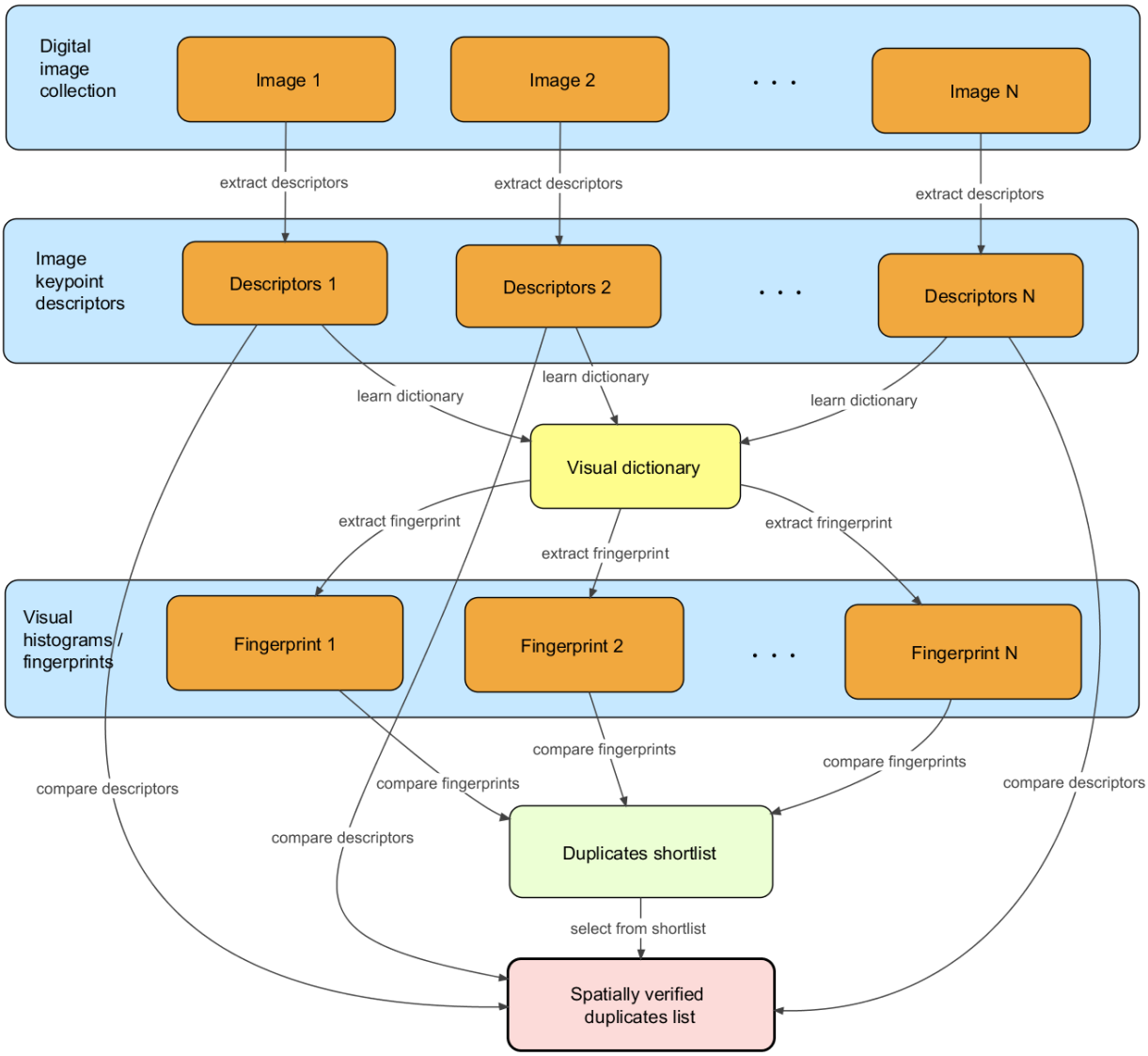

Figure 14 : Duplicate detection workflow

4.4.1 Dataset

The benchmark dataset consists of 52 digitized books provided by the Austrian National Library. It is a representable example of content that has to be processed in the different scenarios. Additionally it is a good example against the use of OCR algorithms.

The dataset contains examples for the following content classes:

- normal justified article pages
- multi-column pages
- handwritten content
- Tables
- mixed content
    - printed and handwritten letters
    - printed pages with handwritten annotations

    - images
- Different languages
    - Books in Cyrillic, Greek, Hebrew language
    - Mixed language (side by side, paragraphs, textboxes)

Collection Name	Number of Image Files	Collection Name	Number of Image Files	Collection Name	Number of Image Files
+Z119528906	922	+Z137196001	94	+Z150976104	158
+Z119529601	486	+Z137203807	568	+Z150976207	208
+Z119565605	858	+Z137205804	814	+Z151616508	1070
+Z119566804	624	+Z137205907	858	+Z151638401	415
+Z119567602	256	+Z13721930X	986	+Z151671106	482
+Z119572300	1016	+Z137220404	368	+Z151687606	58
+Z119575003	888	+Z137237301	484	+Z151694209	772
+Z119586608	820	+Z137239607	554	+Z151694702	730
+Z136403308	306	+Z137247707	862	+Z151698604	490
+Z136417009	226	+Z13727100X	706	+Z151699207	1000
+Z136424403	798	+Z137274000	292	+Z152200609	1628
+Z136432308	484	+Z150450702	876	+Z152213008	868
+Z136432400	348	+Z150709801	190	+Z153936506	272
+Z136436600	714	+Z150711807	462	+Z162507508	192
+Z13646520X	568	+Z150800701	712	+Z150930402	556
+Z136465508	592	+Z150803805	110	+Z150964102	152
+Z136466203	1086	+Z150816800	1054	+Z136905909	1068
+Z137114501	416				

Table 2: Image QA Dataset

5 Web QA

5.1 Introduction

The Web has become one of the most important mean of spreading information. Archiving the Web is thus crucial to preserve some useful information for future generations of historians, researchers... or citizens.

Web archiving is usually performed using Web crawlers (bots) that capture Web resources by parsing Web pages. Once content is captured, it is rendered to the user through an "access tool" that allows an online navigation as on live. Although tools and method exist for capture, access and preservation of web content, web archivists are still facing great challenges. First of all, web content remains extremely complex to capture and access. There is a constant need to optimize the crawlers and access tools to overcome new technological challenges and manage the ephemerality of content published on the web. Second, Web Archives are growing in size and preservation of Web content is becoming more and more challenging. Therefore, developing scalable quality assurance methods is crucial. Indeed, quality assurance work is currently very costly and time consuming as it is most of the time done "manually", if done at all.

In Web QA, we try to provide solutions to these challenges as illustrated with scenarios in the next section by proposing tools that automatically compare two web pages and decide if they are similar or not. For the D11.2, we also took the scalability challenge into account and updated our tools to work with streams instead of files, which requires disk access.

5.1.1 Related Scenarios

WCT1 Comparison of Web Archive pages

The best practice in preserving websites is by crawling them using a web crawler like Heritrix[32]. However, crawling is a process that is highly susceptible to errors. Often, essential data is missed by the crawler and thus not captured and preserved. So if the aim is to create a high quality web archive, doing quality assurance is essential.

Currently, quality assurance requires manual effort and is expensive. Since crawls often contain thousands of pages, manual quality assurance will be neither very efficient nor effective. It might make sense for “topic” crawls but remains time consuming and costly. Especially for large scale crawls, automation of the quality control processes is a necessary requirement.

For further information:

http://wiki.opf-labs.org/display/SP/WCT1+Comparison+of+Web+Archive+pages

WCT2 ARC to WARC migration & WCT6 (W)ARC to HBase migration

ARC to WARC migration and (W)ARC to HBase migration scenarios aim to check the quality of migration operation by comparing two web pages from the source version to target version. However these scenarios are actually deleted, in the case of need Web QA tools can be used.

For further information:

http://wiki.opf-labs.org/display/SP/WCT2+ARC+to+WARC+migration

http://wiki.opf-labs.org/display/SP/WCT6+%28W%29ARC+to+HBase+migration

32 https://webarchive.jira.com/wiki/display/Heritrix/Heritrix

WCT7 Format obsolescence detection

The rendering issues due to format obsolescence within the archive and monitor technological landscape can be detected by the comparison of web pages. This scenario aims at finding possible solutions such as using of image comparison to detect rendering errors within web archives(compare reference snapshots of web pages in different browser versions (WP11-WP12) and/or using of pattern recognition to establish a complex signature for an HTML page (WP9). Statistics produced would then be fed to the watch component to be used by librarians, archivists, etc., to trigger preservation event.

For further information:
http://wiki.opf-labs.org/display/SP/WCT7+Format+obsolescence+detection

5.2 Tools and Interfaces

In Web QA, we have two main tools. Marcalizer, a supervised framework and Pagelyzer , a tool which compares two web pages versions and decides if they are similar or not.
It 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.

Figure 16 shows overall procedure for Web QA. For each URL given as inputs, Pagelyzer gets screen capture in PNG format and also produces an HTML document with the visual cues integrated, called Decorated HTML. This allows saving the state of a browser at the moment of capture and permits to decouple the solution from a particular browser. Then, each page is segmented based on their DOM tree information and their visual rendering and saved as XML files called Vi-XML. They are compared and finally a delta file (called XML Delta) describing the change is produced to be used as input to our supervised framework, called Marcalizer. In the next sections, we explain how Marcalizer and Pagelyzer work in detail.

5.2.1 Marcalizer

Marcalizer is a tool containing a supervised framework that decides if two web pages are similar or not. This tool take two images x,y and compute a score d(x,y). The images are capture with Pagelyzer. Marcalizer can be downloaded from https://github.com/lechervy/MarcAlizer. The new version of Marcalizer (V2.0) uses streams as inputs to analyse the web pages.

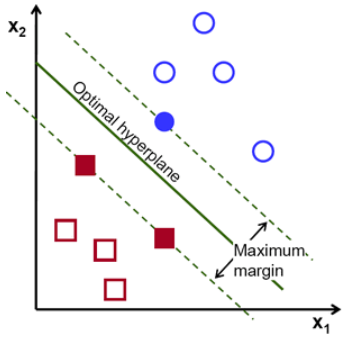

Figure 15: Linear SVM for binary classification

Similarity score displayed by the tool is calculated by linear SVM (Support Vector Machine) (Vapnik). SVM perform well in binary classification in particularly for generalization. Like other supervised machine learning algorithms, an SVM works in two steps. In the first step — the training step — it learns a decision boundary in input space from pre-classified training data. In the second step — the classification step — it classifies input vectors according to the previously learned decision boundary. A single support vector machine can only separate two classes — a positive class (y = +1) (in our context pairs similar) and a negative class (y = -1) (in our context pairs dissimilar). Thus, a similarity score greater than 0 (positive) means that the pages are similar. A similarity score smaller than 0 (negative) means that the pages are dissimilar. However, as the score is based on a learned function, a margin is described as 1. The scores in (-1.1) are considered as “not very sure”. Maximum-margin hyperplane and margins for an SVM trained with samples for two classes is shown in Figure 15. Samples on the margin are called the support vectors.

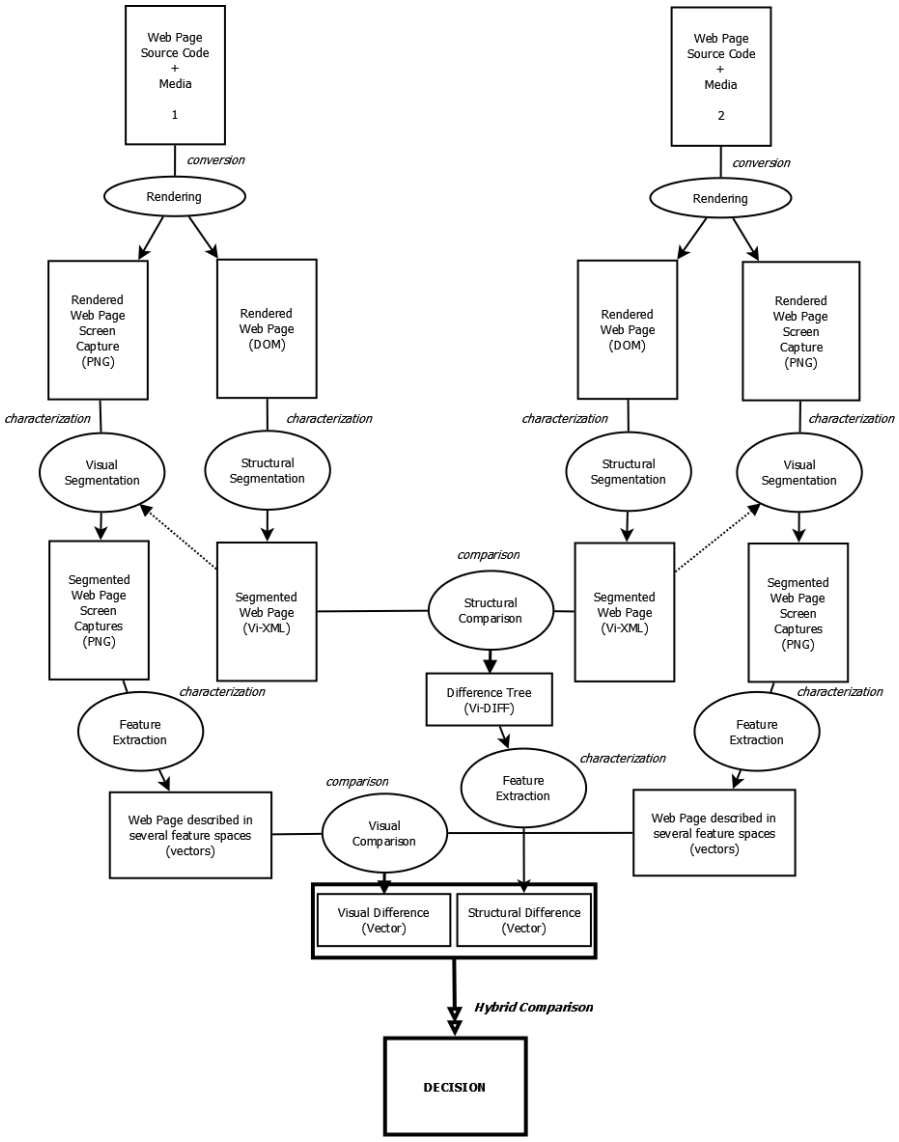

Figure 16 : Overall procedure for Web QA

Our SVM works using visual descriptors of web pages. These descriptors are global and they are computed only on the visible parts of the web page. For each web page, we compute a color histogram with a bag of word (Sivic), we count how many times each visual word appears. We use two dictionaries, one with 100 words and one with 200 words. The vocabulary is constructed by K-means clustering. Then we compute the descriptor of a pair of pages. Each dimension is defined as the distance between two color histograms of web pages (we use L2 distance and chi 2 distance). That means our visual descriptor of web page has 4 dimensions.

We learn the SVM with 202 couples of web page provided by IM, 147 are in positive class and 55 are in negative class.

5.2.2 Pagelyzer

It is an open-source, platform-independent and functional prototype composed of three components: capture, analyser and change detection tools.

The capture tool, for each URL given as input, gets a screen capture in PNG format and also produces an HTML document with the visual cues integrated, called Decorated HTML. This allows to save the state of a browser at the moment of capture and permits to make the solution independent from a particular browser. Web pages are processed using Selenium Web driver. The visual cues are obtained with a set of JavaScript scripts that are injected to the browsers and the screen-shots using selenium features.

The analyser tool implements the Block-o-Matic segmentation algorithm described in (Sanoja et al, 2013). Having as input a decorated HTML file, it produces a Vi-XML file as output. At the earlier versions of the tool, VIPS (Cai) is used to segment web pages. However, we developed a new algorithm called Block-o-Matic, which removes the VIPS restriction of using IE as a web browser and also enhances the precision of visual block extraction and the hierarchy construction. Ruby 1.9.2 was used as programming language for implementing the segmentation algorithm, Nokogiri libraries are used for HTML/XML manipulation.

In the change detection tool, visual and structural descriptors are extracted. Images (snapshots) are first described by colour descriptors and also by SIFT descriptors. Structural descriptors are based on Jaccard indices and also based on the Vi-XML files differences. The structural and visual differences are merged to obtain a similarity vector according to (Law et al, 2012).

For the current version 0.9.1 in order to improve efficiency on use of resources in high performance computing environments, the option to use input data as streams and headless browsers have been introduced. Screen-shots and source code are captured directly from their hosting sites and processed internally with no extra files needed.

5.2.3 Deployment guides

There are two ways for using Pagelyzer:

- The standalone version is suitable for all systems. It is developed using Ruby 1.9.2 and has dependencies to Java JDK 1.6. It has been tested in Ubuntu 12 and Debian stable. It can be accessed in:

- https://github.com/openplanets/pagelyzer
- http://www-poleia.lip6.fr/~sanojaa/pagelyzer-ruby-0.9.1-standalone.zip
- Usage and dependencies installation information can be obtained in the README file

The Debian package version includes the same functionality as the standalone version but is easier to install because dependencies are resolved by package manager. For the construction of the Pagelyzer Debian package it was needed to use a different approach. Ruby uses neither the make nor the Makefile directive. It is place the rubygems packaging, as source, has been introduced to Debian style using make's equivalent setup.rb . That allows constructing source Debian packages compliant to SCAPE expectations. More detail on this can be found in
http://www-poleia.lip6.fr/~sanojaa/pagelyzer-package-guide.pdf

Pagelyzer

```
./pagelyzer [--help|--version] [<command> <command_options>]
```

Pagelyzer command is a combination of following commands that can be used separately:

- capture
- analyzer
- changedetection

Capture

```
./pagelyzer capture --url=URL [--output-folder=FOLDER][-browser=BROWSER_CODE]
[--thumbnail] [--help]
```

Analyzer

```
./pagelyzer analyzer --decorated-file=FILE [--output-file=FILE]
[-pdoc=(0..10)] [--version] [--help]
```

Changedetection

```
./pagelyzer changedetection --url1=URL --url2=URL [--doc=(1..10)]
[--output-folder=FOLDER] [--browser=BROWSER_CODE | --browser1=BROWSER_CODE
--browser2=BROWSER_CODE] [--verbose] --type=[hybrid|images|structure]
```

Examples of use

Capture a web page with default parameters:

```
$ ./pagelyzer capture --url=http://www.google.fr
```

It will copy to $HOME_FOLDER/pagelyzer the outcome. If the folder does not exist it will be created. It will create three files:

- firefox_www_google_fr.html (rendered version of the web page)
- firefox_www_google_fr.dhtml (rendered version with visual cues included for segmentation algorithm)

- firefox_www_google_fr.png (webshot of the page)

Change detection on two pages with default parameters

```
$ ./pagelyzer changedetection --url1=http://www.host.com/page1.html
--url2=http://www.host.com/page2.html
```

Change detection on two pages with hybrid method

This command will create the same files as above for each URL and also the Vi-XMLs and delta file.

```
$ ./pagelyzer changedetection --url1=http://www.host.com/page1.html
--url2=http://www.host.com/page2.html --type=hybrid
```

Change detection with different browsers

```
$ ./pagelyzer changedetection --url1=http://www.host.com/page1.html
--url2=http://www.host.com/page2.html --browser1=firefox --browser2=chrome
url1 will be evaluated with browser1 and url2 with browser2
```

Change detection with one browser (the most common case)

```
$ ./pagelyzer changedetection --url1=http://www.host.com/page1.html --
url2=http://www.host.com/page2.html --browser=firefox
```

5.3 Taverna Workflow

The Web QA workflow is a workflow that uses screenshots of web pages as input, analyse them using visual and structural comparison and decides if the two pages are similar or not. A score smaller than 0 (negative) means that the pages are dissimilar while a score greater than 0 (positive) means that the pages are similar. However, as the score is based on a learned function. The scores in (-1.1) are considered as “not very sure”.

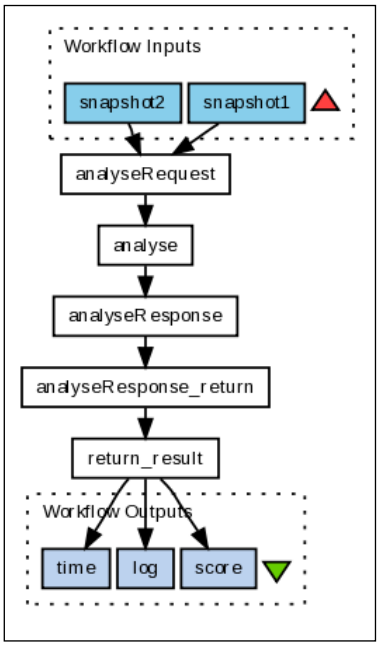

Figure 17 : Web QA workflow

5.4 Correctness based Benchmarks and Validation Tests

5.4.1 Dataset

The dataset is composed of 449 pairs of URLs freely accessible from the Internet Memory Web Archive. The URLs composing the dataset were selected in a random manner within a list containing the most accessed URLs within the web archive for a given period. This list was built using IM's web archive monthly usage statistics.

For this evaluation work, we decided to compare web pages rendered by two separate accesses in our archive rather than comparing a URL from the web archive and its live version. This allowed us to ensure a permanent access to all URLs of the corpus. It also avoided issues such as seeing a live link disappear from the live or changing entirely during the evaluation period.

Content Type	Number	Percentage
HTML	273	60.80
Index page	52	12.69
Blank page	57	11.58
Js file	2	6.46
Error message	29	3.34
Error page	15	2.67
Xml file	6	1.34
Image	1	0.45
Zip file	1	0.22
Doc file	1	0.22
PDF file	1	0.22

Table3: Content type repartition

IM QA team performed the "manual" annotation. IM team is trained to perform visual comparison of web pages and compares hundreds of pages every day to ensure a high level of quality to our web archive. During this annotation phase, the team was asked to compare visually two URLs from two separate web archive access and state if these were similar or dissimilar:

-URLs were annotated as similar and given the score 0 when these were either slightly different or entirely identical.
A slight difference would be a line added as part of a text for instance.
-URLs were annotated as dissimilar and given a score from 1 up to 5 when quite or completely different.
A quite strong difference would be an image missing in an important area of the page.

Table 4 below shows how the assessors presented results after the annotation phase.
Column 1 and 2 provide the pairs of URLs compared. The assessor then filled column 3 or 4 to show the similarity or dissimilarity of the pairs. Finally, column 5 provides a dissimilarity score as described above and column 6 qualifies the content type (script, image, html page, etc.).

Url1	Url2	Similar	Dissimilar	Score	Content type
http://...	http://...		1	3	html page
http://...	http://...	1		0	xml file

Table 4: Annotation table

Overall distribution of annotation by type can be found in Table 5.

Type	Similar	Dissimilar
Blank page	52	0
doc file	1	0
error message	15	0
error page	12	0
HTML	227	46
image	1	0
image png	1	0
Index page	20	37
javascript file	29	0
PDF file	1	0
xml file	6	0
zip file	1	0
TOTAL	**366**	**83**

Table 5: Overall Distribution of annotations

5.4.2 Experiments and Results

In parallel to the manual annotation, the 449 URLs were annotated using the Pagelyzer tool. IM performed tests with Pagelyzer based on visual comparison. As Pagelyzer for hybrid and structural changes is still on beta version, these tests are realized at UPMC. The results are presented according to different annotations (Similar, Dissimilar) and according to page types. The results by categories can be found in Table 4. Total values are calculated as a weighted mean by type based on Table 5. We do not take into account the small number of page types like zip (only 1 zip file overall collection) while discussing the results and will focus on bigger figures such as for instance HTML format.

For HTML pages, visual and content comparison results are promising with success 75% for visual and 84% for content. However, hybrid approach, which is still on beta version, stays behind the other two approaches. As a future work, we will try to see the reason behind the hybrid results and relearn the related function in supervised framework.

Type	Similar			Dissimilar			Overall		
	Visual	Content	Hybrid	Visual	Content	Hybrid	Visual	Content	Hybrid
Blank page	0,98	1,00	1,00	-	-	-	0,98	1,00	1,00
Doc file	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
Error message	0,27	0,13	0,40	-	-	-	0,27	0,13	0,40
Error page	0,42	0,08	0,17	-	-	-	0,17	0,08	0,17
HTML	0,75	0,83	0,60	0,78	0,89	0,76	0,75	0,84	0,63
Image	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
Image png	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
Index page	0,70	0,00	0,00	0,03	1,00	1,00	0,26	0,65	0,65
Javascript file	0,83	0,41	0,24	-	-	-	0,83	0,41	0,24
PDF file	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
XML file	0,00	0,00	0,00	-	-	-	0,00	0,00	0,00
Zip file	1,00	1,00	1,00	-	-	-	1,00	1,00	1,00
TOTAL	**0,75**	**0,71**	**0,57**	**0,45**	**0,94**	**0,87**	**0,68**	**0,76**	**0,62**

Table 6: Benchmarking Results for Web QA

6 Knowledge Base

Digital file formats may have features that are in conflict with the aims of long-term preservation. Examples are:

- encryption, password protection and use of non-embedded fonts in *PDF*;
- features in *JPEG 2000* images that are known to cause interoperability problems (location of resolution in headers, embedding methods for *ICC* profiles, *JPX* images that are identified as *JP2*).

Some of these issues are the result of software bugs (e.g. JPX identified as JP2), others are caused by ambiguous information in the format specification (e.g. the exact location of resolution information in a JP2), whereas others still are simply user-defined options that are not related to any software of file specification shortcomings at all (e.g. font embedding and encryption in PDF, which are user-specified options). If an institution formulates format-specific (control) policies, these policies are likely to focus on exactly these aspects.

6.1 Aim of knowledge base work

The aim of the work here is the creation of an open knowledge base that addresses exactly the above features. For a selected number of formats it provides the following information:

1. specific features of the format that imply a possible preservation risk;
2. a link to (characterisation) tools that are able to detect if a particular file instance is affected by these features;
3. a link to sample files that serve as evidence.

Within SCAPE the primary focus will be on the *JP2* and *PDF* formats, but ultimately anyone can contribute input on any possible format.

6.2 Current status: OPF File Format Risk Registry

Following a couple of name-changes (partially in order to avoid confusion with an entirely different 'Knowledge Base' that is part of Automated Watch), the 'Knowledge Base' work was released under the moniker 'OPF File Format Risk Registry' (*FFRR*). It is available here:
http://wiki.opf-labs.org/display/TR/OPF+File+Format+Risk+Registry
The entries on the *JP2* format are now mostly finalised. The following entry demonstrates the main principles of the risk registry:
http://wiki.opf-labs.org/display/TR/Handling+of+ICC+profiles
At the very minimum, each 'risk' entry contains a brief description, and an explanation of the implications for long-term accessibility. In most cases there is also an 'assessment' section that explains how the presence of a risk can be identified. In our example, it points us to the *Jpylyzer* tool, and explains the relevant fields in *Jpylyzer* 's output. Since the handling of *ICC* profiles is dependent on creator applications, a table is provided that shows the behaviour of the most widely-used implementations. Most 'risk' entries also contain recommendations that are relevant for pre-ingest (and QA), and existing collections.

6.2.1 File evidence

A very important aspect of the *FFRR* is that wherever possible, reported risks should be supported by evidence. To this end, most 'risk' entries contain an 'Example files' section that links to actual files in the OPF Format Corpus . These files were specifically created for this purpose, and the dataset (including a detailed description) can be found here:
https://github.com/openplanets/format-corpus/tree/master/jp2k-test

7 QA Tools

This chapter describes QA tools that are being developed in this work package.

7.1 Document tools

7.1.1 Apache Preflight

The *PDF* format contains a number of features that may make it difficult to access content that is stored in this format in the long term. Examples include (but are not limited to):

- Encryption features, which may either restrict some functionality (copying, printing) or make files inaccessible altogether.
- Multimedia features (embedded multimedia objects may be subject to format obsolescence)
- Reliance on external features (e.g. non-embedded fonts, or references to external documents)

7.1.1.1 PDF/A-1 profile as a basis for identifying preservation risks

The *PDF/A-1* profile defines a set of features that are widely considered to be 'safe' from a preservation point of view. A comparison of any *PDF* document against this profile reveals all features that are not allowed in *PDF/A-1*, and which can be seen as 'potentially risky'. Existing *PDF/A-1* validator tools allow us to do such a comparison, and as a result the output of these tools could be helpful to identify preservation risks in *PDF*.

7.1.1.2 Preliminary assessment of Apache Preflight

As a firsts step we assessed the feasibility of this approach using the open-source *Apache Preflight* validator[33] that is part of the *Apache PDFBox* library[34]. Detailed results are given in a separate technical report [Preflight 2012]. The conclusion of this work was that *Apache Preflight* showed great promise for detecting preservation risks in *PDF*, but that the software was not mature enough for operational use. However, following the report's publication date, the developers of *Apache Preflight* have promptly corrected the bugs and problems that we reported. Some further work with *Apache Preflight* during a hackathon in March 2013 showed that the software had improved greatly since our initial tests [SPRUCE PDF]. So, more work on this will follow in year 3 of the project.

7.1.1.3 Correctness based Benchmarks and Validation Tests

If we want to say anything meaningful about *Preflight*'s ability to detect specific preservation risks, we need ground truth: *PDF* files that are *known* to contain encryption, password protection, non-embedded fonts, and so on. One relevant dataset in this regard is the Isartor Test Suite [Isartor]. This is a set of small *PDFs*, each of which violates *PDF/A-1* in one particular way. However, the specific aim of the SCAPE work implies that we will mostly be dealing with *PDF* files that contain multiple violations of *PDF/A-1* (since they are not *PDF/A-1* to begin with!), which meant that somewhat more realistic ground truth was needed. This prompted the creation of *The Archivist's PDF Cabinet of Horrors*, which is a small, annotated, openly-licensed test corpus of *PDF*s with encryption, non-

33 https://pdfbox.apache.org/userguide/preflight.html
34 http://pdfbox.apache.org/

embedded fonts, external references and embedded scripts and multimedia content. It is part of the OPF Format Corpus, and can be found here:

https://github.com/openplanets/format-corpus/tree/master/pdfCabinetOfHorrors
This dataset was also used as a basis for evaluating *Apache Preflight* in the aforementioned technical report [Preflight 2012].

7.1.1.4 PDFEh - JavaFX GUI Wrapper for PDF Preflight

PDFEh is the result of work carried out by the British Library, Johan Van de Knijff and Sheila Morrissey during a hackathon at Leeds University in March 2013. The aim of the work was to identify PDF preservation risks. However, the term 'risk' is open to interpretation and one organisation's idea of what constitutes a risk is unlikely to be the same for other organisations. With this in mind a policy-driven approach was taken that enables users to categorise issues flagged by Preflight as warnings, errors (failures) or ignorable. This allows the results returned from pre-flight to be filtered according to the policies of a particular organisation.
Some of the enhancements made as part of this work include:-

- output of Preflight results in XML rather than unstructured text
- a policy file to categorise all the outputs returned by Preflight (error, warning, ignorable)
- a JavaFX Gui Wrapper which allows the user to see all possible Preflight output codes and specify the filtering rules.

It is expected that work on PDFEh will continue into year 3.
The PDFEh code developed during the hackathon can be found at
https://github.com/openplanets/pdfeh
There is also a Wiki post on the subject
http://wiki.opf-labs.org/display/SPR/PDF
Pete Cliff's Blog provides more information about the work carried out during the hackathon.
http://www.openplanetsfoundation.org/blogs/2013-03-15-pdf-eh-another-hackathon-tale

It is expected that in year 3 the work on PDFEh will be extended and be applied to eBooks, in particular eBooks of format ePub version 3. The aim of this work is to:

a) Identify and flag preservation risks associated with ePub books. These risks will be policy-driven, as was the case with the PDFEh work.

b) Ensure that the content of eBooks migrated to ePub has not been compromised, i.e. carry out a comparison of the content of two eBooks, the original version and the migrated version, and report any inconsistencies.

7.2 Imaging tools

7.2.1 Jpylyzer

Jpylyzer is a validator and feature extractor for JP2 (JPEG 2000 Part 1 / ISO/IEC 15444-1) images. This tool serves two purposes:

1. *Format validation* – does an image really conform to the format's specifications?
2. *Feature extraction* - what are an image's technical characteristics?

Jpylyzer's development was prompted by the observation that the validation component of the only existing similar tool (*JHOVE*) turned out to have shortcomings that severely limit its usefulness in imaging workflows.

7.2.1.1 Functionality and scope

Within the context of imaging workflows, *Jpylyzer*'s validation functionality can be used to ensure that created images are standards-compliant; besides, it will also detect common types of byte-level corruption. Examples are truncated or otherwise incomplete images, or images that are affected by file transfer errors (e.g. images that were mistakenly transferred over a network connection in *ASCII* mode). The feature extraction functionality makes it possible to test whether a *JP2* conforms to a technical profile. For example, *Jpylyzer*'s output gives detailed information on whether an image uses lossless or lossy compression, its compression ratio, colour space, resolution, and so on.

Importantly, *Jpylyzer* is *not* capable of decoding the compressed bitstream information in an image, which means that it is (at least theoretically) possible for a *JP2* to pass each of Jpylyzer's tests, while at the same time the image may fail to render correctly in a viewer application. For the latter, it is recommended that the image QA workflow also includes some pixel-based image comparison between the source and destination images (in case of a format migration), or otherwise at least a test that checks if an image renders at all.

Jpylyzer is written in Python, and runs under both Python 2.7.x and Python 3.2 and more recent. It can either be used as a command-line tool, or as a library that can be imported in other Python programs. In addition, Debian packages and Windows executables exist, both of which remove any dependency on Python whatsoever.

7.2.1.2 Correctness based Benchmarks and Validation Tests

Jpylyzer's main purpose is to establish whether an image is valid *JP2*. This raises the question if we can define a set of test data that can be used as 'ground truth' for establishing *Jpylyzer*'s correctness. This ground truth would serve two purposes:

1. It would back up Jpylyzer's claims by actual evidence, and it would allow (potential) users of the software to make a first assessment of its suitability to their needs.

2. It would help (future) developers and other contributors, by providing a reference dataset with known (intended) behaviour.

Part 4 of the JPEG 2000 standard is concerned with testing of conformance to JPEG 2000 Part 1 (JP2) [JPEG 2000 Part 4]. Since this part of the standard includes a suite of test images, at first sight this might look like a possible candidate for a correctness based benchmark. However, early tests revealed that none of the test images of JPEG 2000 Part 4 actually conform to the standard[35]! This is (at least partially) explained by the fact that these images were primarily created for testing the conformance of reader/viewing applications. JPEG 2000 Part 1 states that conforming readers

35 More details on this can be found here: http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=664747#18

should, under specific conditions, ignore some of the header information in JP2. Because of this, some subtle deviations from the standard were deliberately introduced into the test images. However, this also makes them unsuitable as ground truth for a validator tool.
Because of this, we decided to create a new, openly licensed set of annotated reference images. It is accessible on the Open Planets Github repository:
https://github.com/openplanets/jpylyzer-test-files
The files in the dataset fall in a number of categories:

- conforming *JP2* images that were created by different encoders (6 files);
- images that violate the *JP2* format specification in various ways, including bit- or byte-level corruption (8 files);
- images that are not actually *JP2* (or even part of the JPEG 2000 format family) at all (5 files).

The last category may appear odd, but in principle *Jpylyzer* should be able to deal with *any* file without crashing, which makes these images quite useful for developers.
At present the dataset contains 19 images, but this number is likely to increase in the future.

7.2.2 Bitwiser

The Bitwiser tool was developed by the British Library as a means of analysing the performance of preservation tools. It works by systematically flipping every single bit in the given input file, and re-running a preservation tool (for example, conversion software) on the input file for each flipped bit.

The aim is to understand how the preservation tool processes the data and effectively carry out QA on the tool itself.

No work has been carried out on the Bitwiser tool in year 2 as the British Library is of the opinion that it is not a suitable subject for QA as it is more of a research project rather than a QA scenario.

8 Roadmap and Conclusion

In this section you can find the future work for each task.

8.1 Audio QA

The focus of Audio QA in the third year of the project will be scalability and integration. This means experiments with waveform-compare and possibly the full audio QA workflows using Hadoop and integrated with the SCAPE Platform.

8.2 Document QA

During the third year, Office Document QA will focus on measuring the migration quality of content and layout over corpora containing diverse types of office document. The objective is to identify the optimal parameter settings for the QA metrics in order to ensure they are useful for the management of the migration process.

8.3 Image QA

According to the presented benchmark results, further effort will be put into enhancing the precision of the Matchbox Toolset. One step will be to enhance the handling of white pages, by either contrast enhancement or preceding categorization of the collection into different content classes.
A second objective will be to apply the workflow onto a distributed environment. Thus, the workflow will be implemented in Hadoop-Script and first experiments towards scalability benchmarks will be initiated.

8.4 Web QA

Web QA will focus on the scalability and integration in the next year of the project. Besides that, with more annotated data provided by IM, supervised framework will be improved for Pagelyzer. The related updates for integration of Pagelyzer to Hadoop will be done and the scalability benchmarking will take a place at IM.

8.5 Knowledge based QA

During the third year of the project work on the 'OPF File Format Risk Registry' (*FFRR*) will focus on the creation of entries for the *PDF* format, including an annotated dataset that serves as evidence for the listed preservation risks. The latter will be done largely in tandem with the work on identifying preservation risks in *PDF*.

8.6 QA Tools

Jpylyzer

Additional work will be done on the further improvement of *Jpylyzer*, which will include improving the tool's robustness in the presence of bit-level corruption.

Apache Preflight

The work on identification of preservation risks in *PDF* with *Apache Preflight* will continue. This will include an updated evaluation of the most recent version of the software using the *Archivist's PDF Cabinet of Horrors* corpus, and additional performance tests on a large dataset of open-access *PDF*s that will be released later during 2013. In addition to this we will use the results of the *Preflight* analysis for 'validating' *PDF*s against institutional (control) policies. This will expand on earlier work on this that was done during the SPRUCE characterisation hackathon in Leeds [SPRUCE PDF].

The British Library is also expecting to continue the work on Apache Preflight in the context of validating and identifying risks in PDF format eBooks.

Bitwiser
No further work will be carried out on the Bitwiser tool.

9 Bibliography

[Isartor] PDF Association: Isartor Test Suite. Link: http://www.pdfa.org/2011/08/isartor-test-suite/
[JPEG 2000 Part 4] JPEG 2000 Conformance (Part 4) Link: http://www.jpeg.org/jpeg2000/j2kpart4.html

[Preflight 2012] Van der Knijff, Johan: Identification of preservation risks in PDF with Apache Preflight - a first impression. Link: http://www.openplanetsfoundation.org/system/files/pdfProfilingJvdK19122012.pdf

[SPRUCE PDF] Cliff, Pete: PDF Eh? - Another Hackathon Tale. Link: http://www.openplanetsfoundation.org/blogs/2013-03-15-pdf-eh-another-hackathon-tale

[Law] M. T. Law, C. S. Gutierrez, N. Thome, and S. Gancarski. Structural and visual similarity learning for web page archiving. In Content-Based Multimedia Indexing (CBMI), 2012 10th International Workshop on, pages 1#6. IEEE, 2012.

[Pehlivan] Z. Pehlivan, M. Ben-Saad, and S. Gançarski. Vi-diff: understanding web pages changes. In Database and Expert Systems Applications, pages 1-15. Springer, 2010.

[Sanoja] A. Sanoja, S. Gançarski. Block-o-Matic a web page segmentation algorithm. Technical Report. LIP6-UPMC. Unpublished. 2013.

[Cai] D.Cai, Shipeng Y., J. Wen, and W. Ma, VIPS: a Vision-based Page Segmentation Algorithm. Tech.Report MSR-TR-2003-79, November 2003.

[Sivic] J. Sivic and A. Zisserman. Video google : A text retrieval approach to object matching in videos. In IEEE International Conference on Computer Vision, volume 2, pages 1470–1477, 2003)

[Vapnik] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

Appendix A

Collection Name	Precision	Recall	Ground-Truth	True Positive	False Negative	False Positives	White Pages
+Z151698604	**0,8947**	**0,7727**	22	17	5	2	0
+Z119565605	**0,5263**	**1,0000**	20	20	0	18	2
+Z136466203	**0,9032**	**1,0000**	28	28	0	3	9
+Z150964102	**1,0000**	**0,5000**	8	4	4	0	6
+Z136424403	**0,7692**	**1,0000**	20	20	0	6	3
+Z137247707	**0,9565**	**1,0000**	22	22	0	1	10
+Z137205804	**0,9600**	**0,7500**	32	24	8	1	10
+Z151616508	**0,4762**	**1,0000**	20	20	0	22	3
+Z151687606	**1,0000**	**0,1304**	23	3	20	0	0
+Z150976104	**1,0000**	**0,1579**	19	3	16	0	5
+Z151694702	**0,9474**	**1,0000**	18	18	0	1	6
+Z119528906	**0,7143**	**0,9524**	21	20	1	8	8
+Z137114501	**1,0000**	**0,8125**	16	13	3	0	4
+Z13721930X	**0,0784**	**1,0000**	4	4	0	47	3
+Z136432400	**1,0000**	**0,7000**	10	7	3	0	9
+Z136403308	**1,0000**	**0,4000**	20	8	12	0	1
+Z137274000	**1,0000**	**0,6316**	19	12	7	0	0
+Z136465508	**1,0000**	**0,6818**	22	15	7	0	8
+Z137239607	**1,0000**	**1,0000**	16	16	0	0	5
+Z151699207	**0,5238**	**1,0000**	22	22	0	20	1
+Z151638401	**1,0000**	**0,9231**	13	12	1	0	8
+Z150930402	**0,7000**	**0,7000**	20	14	6	6	3
+Z150711807	**1,0000**	**0,7143**	21	15	6	0	3
+Z152200609	**0,2899**	**1,0000**	20	20	0	49	5
+Z137203807	**1,0000**	**1,0000**	22	22	0	0	0
+Z13646520X	**1,0000**	**0,8750**	24	21	3	0	4
+Z150803805	**1,0000**	**0,2500**	20	5	15	0	2
+Z152213008	**0,6154**	**0,9231**	26	24	2	15	1
+Z119567602	**1,0000**	**0,4583**	24	11	13	0	1
+Z150816800	**0,7692**	**0,9091**	22	20	2	6	9
+Z136905909	**0,4082**	**1,0000**	20	20	0	29	2
+Z137205907	**0,8095**	**0,9444**	18	17	1	4	12

+Z137237301	**1,0000**	**0,5909**	22	13	9	0	5
+Z119572300	**0,3061**	**0,7500**	20	15	5	34	4
+Z136432308	**1,0000**	**0,5769**	26	15	11	0	7
+Z150709801	**1,0000**	**0,2857**	21	6	15	0	3
+Z119575003	**0,9524**	**1,0000**	20	20	0	1	10
+Z137220404	**1,0000**	**0,7000**	20	14	6	0	1
+Z162507508	**1,0000**	**0,6667**	6	4	2	0	2
+Z119529601	**1,0000**	**0,7308**	26	19	7	0	1
+Z119586608	**0,9600**	**1,0000**	24	24	0	1	8
+Z150800701	**0,9091**	**1,0000**	20	20	0	2	5
+Z150450702	**0,2105**	**1,0000**	4	4	0	15	13
+Z136417009	**1,0000**	**0,3846**	13	5	8	0	7
+Z153936506	**1,0000**	**0,6875**	16	11	5	0	0
+Z136436600	**1,0000**	**0,9167**	24	22	2	0	3
+Z151671106	**0,9375**	**0,6818**	22	15	7	1	1
+Z119566804	**1,0000**	**0,7917**	24	19	5	0	4
+Z13727100X	**0,6538**	**0,8500**	20	17	3	9	5
+Z150976207	**1,0000**	**0,5000**	16	8	8	0	2
+Z151694209	**0,4545**	**0,8333**	18	15	3	18	2
Mean	**0,8378**	**0,7673**					
Median	**1,0000**	**0,8125**					

Table 3: Image QA Benchmark results

Appendix B

Test File	Test Result	JHove2 isValid	Taverna alike (extract properties using ffprobe, compare using Taverna)	Test Result waveform-compare	Test Result waveform-compare channel 1
challenge-KFC-2.wav	false	true	false (duration)	Success Offset: 0	Success Offset: 3959
challenge-pmd.wav	true	true	true	Success Offset: 0 (Failure Block: 74 if threshold = 0.9999994)	Success Offset: 0
challenge-TEG-1.wav	false	false	Invalid data found when processing input	Error, channel not available	Error, channel not available
challenge-KFC.wav	false	true	false (duration)	Success Offset: 0	Success Offset: 3959
challenge-TE-1.wav	true	true	true	Success Offset: 0 (Failure Block: 80 if threshold=0.99993)	Success Offset: 0
challenge-TEG-2.wav	false	true	false (duration)	Success Offset: 0	Success Offset: 0
challenge-KTC.wav	false	true	true	Failure Block: 120	Failure Block: 120
challenge-TE-2.wav	true	true	true	Success Offset: 0 (Failure Block: 120 if threshold=0.99)	Success Offset: 0
challenge-TEG-3.wav	false	true	true	Failure Block: 120	Failure Block: 120
challenge-TE-3.wav	false	true	true	Failure Block: 120	Failure Block: 120
challenge-TEG-4.wav	false	true	true	Failure Block: 8	Failure Block: 8
Challenge-nbr-1.wav	false	false	true	Error, channel not available	Error, channel not available
challenge-TE-4.wav	false	true	true	Failure Block: 120	Failure Block: 120

challenge-TEG-5.wav	false	false	Invalid data found when processing input	Error, channel not available	Error, channel not available
Challenge-nbr-2.wav	false	false	true	Error, channel not available	Error, channel not available
challenge-TE-5.wav	false	true	true	Failure Block: 120	Failure Block: 120
challenge-UKH.wav	false	true	true	Failure Block: 120	Failure Block: 120
Challenge-nbr-3.wav	false	false	true	Error, channel not available	Error, channel not available
challenge-TE-6.wav	false	true	true	Failure Block: 120	Error, channel not available
challenge.wav	true	true	true	Success Offset: 0	Success Offset: 0
challenge-nbr-7.wav	false	true	true	Failure Block: 52	Success Offset: 3
challenge-TE-7.wav	false	true	false (duration)	Failure Block: 120	Failure Block: 120
Challenge-KFC-3.wav	true	true	true	Success Offset: 0	Success Offset: 3959
DER259955_mpg321.wav	true	true	true	Success Offset: 0	NA mono
complete_garbage.wav	false	true	true	Failure Block: 921	NA mono
complete_silence.wav	false	true	true	Failure Block: 921	NA mono
partly_silence.wav	false	true	true	Failure Block: 512	NA mono
partly_garbage.wav	false	true	true	Failure Block: 718	NA mono

Table 4: Audio QA Benchmark Results