Characterisation technology, Release 3 + release report

Authors
William Palmer (British Library), Sven Schlarb (Austrian National Library), Rune Bruun Ferneke-Nielsen, Per Møldrup-Dalum (State & University Library Denmark), Artur Kulmukhametov (Vienna University of Technology), Alan Akbik (Technical University Berlin)

September 2014

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

Executive Summary

The Characterisation Components work package from the Preservation Components sub project presents the results from the final project year.

We show solutions for characterising large-scale collections of digital objects with Apache Tika and DROID. We have created two general solutions that complement each other with different pros and cons. We present a solution to remedy the challenge of different tools giving different results on the same data. We discuss the concept of policy driven validation of digital objects according to an institutional preservation policy and gives reference to a concrete proof of concept solution.

We present an evaluation of deploying Apache Tika and DROID on the SCAPE Azure platform as an alternative to the general SCAPE Execution Platform.

We present the research project in extracting semantic information from web based text corpora and how such a system could be utilised by the digital preservation community.

Table of Contents

1 Introduction

This is the final report from the Characterisation Components work package in the Preservation Components sub project of the SCAPE project. Knowledge of the preceding reports in detail and the SCAPE project in general is assumed, especially ("Scalable Preservation Environments, Annex I - 'Description of Work'" 2013; May and Wilson 2014), as this report should be regarded as the last part of a three-part volume building on the same concepts. Section 8 contains a glossary with a short description of introduced concepts and link to further reading. During the SCAPE project the work in this work package has been divided in several strands, as documented in the previous two deliverable reports (Raditsch et al. 2012; Møldrup-Dalum et al. 2013). This report addresses the final three primary strands, namely characterisation of file objects, evaluation of the Microsoft Azure platform, and information extraction from web based text corpora.

Prior to the first deliverable report we performed an initial evaluation of a large set of tools for identifying file objects and for extracting features from such objects. That evaluation resulted in the selection of three tools for further evaluation. These three tools were Apache Tika, the Digital Record Object Identification (DROID) tool, and the Format Identification for Digital Objects (FIDO) tool. These three tools have all have been thoroughly described in detail in (van der Knijff and Wilson 2011; Raditsch et al. 2012; Møldrup-Dalum et al. 2013). The tools were evaluated against their ability to identify digital file objects and, for the relevant tools, also their ability to extract features from digital file objects. The work on evaluating the selected characterisation tools on the Azure platform has been a sub task in building a general SCAPE Azure Platform. That platform is described both outside of this work package in (Vujic 2014a) and in (Møldrup-Dalum et al. 2013). Semantic information extraction from online text corpora focusing on digital preservation was described in detail in the last deliverable report (Møldrup-Dalum et al. 2013) in addition to several scientific papers and presentations published on the subject (Akbik, Konomi, and Melnijkov 2013; Akbik and Löser 2012; Faria et al. 2013; Maqsud et al. 2014; Akbik et al. 2013).

The three strands will in turn be discussed in this report. Section 2 reports on the progress that has been made in adapting Apache Tika and DROID to the SCAPE and the Hadoop platforms. Two different approaches are presented and compared. This section will also report on how the File Information Tool kit (FITS) has been evaluated for performing feature extraction on digital objects. Following Section 2 are sections on how to handle conflicting results from running several characterisation tools on the same data and on performing preservation policy driven validation of digital objects. These two latter subjects use the data from the characterisation tools to achieve their objectives. Section 5 reports on the effectiveness of running Apache Tika and DROID on the SCAPE Azure platform in the context of performing file migration and quality assurance. Information extraction on text and web corpora is presented in section 6

2 Identification and feature extraction tools

One of the main goals of the SCAPE projects and therefore this work package is to develop tools for very large digital collections. In this project year we have focused on evaluating and optimising a few characterisation tools for very large data sets.

The SCAPE Execution Platform for performing large-scale preservation actions is based on Apache Hadoop (May and Wilson 2014). It is beyond this report to present in detail Apache Hadoop and instead we refer to ("Hadoop - Apache Hadoop 2.4.1" 2014; White 2012) but a few crucial concepts deserve mentioning as they are relevant for our discussion.

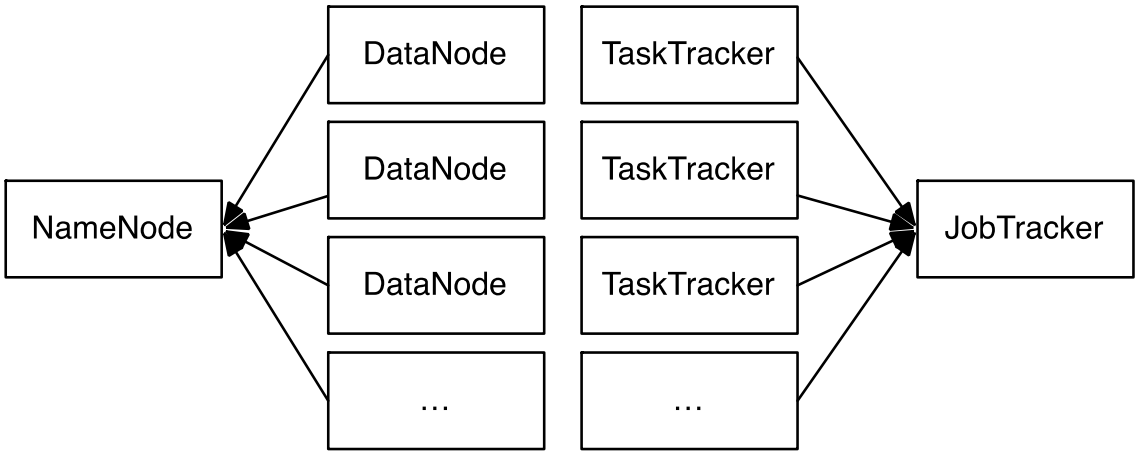

Figure 1: Some components of Apache Hadoop

A Hadoop cluster consists of multiple services running on one or several cluster nodes, those being physical or virtual. Computation tasks are controlled by one JobTracker that delegates every task to TaskTrackers, one for each cluster node. Storage is controlled by one NameNode that delegate access to DataNodes, one for each node. When the JobTracker sends a task with a corresponding data chunk to a TaskTracker, this is done such that the data chunk is as close to the actual TaskTracker CPU as possible—this concept is called *data locality*. When the JobTracker selects which TaskTracker to use for a certain data chunk, it also considers TaskTracker load and this in concert gives us a very high optimisation of the available computing power.

In the SCAPE project we have three main collection types, namely scientific data, large scale data, and web content. Web content is by far the most heterogeneous and therefore the biggest challenge and best test corpora for characterisation tools and has therefore been the focus in this project year.

On a traditional Hadoop cluster, data is stored on the Hadoop Distributed File System (HDFS) that is controlled by one NameNode and multiple DataNodes, one for each cluster node. Each cluster node contributes a certain amount of data storage to the total distributed file system. In this way, some data is always closest to one CPU node.

Under certain circumstances it is preferable to keep the data on a network attached storage system (NAS) available to all worker nodes in the cluster instead of on HDFS. This could be due to data size, access rights, or any other given political, organisational, or technical restrictions. Under such circumstances, data locality is lost, as all data has, in principle, the same distance/access latency to each worker node. As it was also pointed out in the previous deliverable report (Møldrup-Dalum et al. 2013) apart from storing temporary files in Hadoop's Distributed File System (HDFS), it is also possible to use the local file system of the worker nodes of a Hadoop cluster. This can be useful, for example, to create file instances during the map phase of a MapReduce job in the temporary file system of the worker nodes. While both these examples lessen the advantage of data locality we still benefit from the parallelisation framework and the Hadoop eco-system. In order to allow that the various worker nodes can have shared access to files, the cluster network can have a shared file server that stores global persistent data, accessed by the slave nodes as needed.

In (Møldrup-Dalum et al. 2013)we presented a solution for using Apache Tika and DROID on the Hadoop Framework and in this report we present how to use these tools on very large data sets on the Hadoop Framework.

There are several major differences between Apache Tika and DROID regarding their corresponding API's for handling data objects. On one hand, DROID accesses files on the file system using a file system path specified in a metadata object assigned to the identification request object (Møldrup-Dalum et al. 2013, 10). Such a metadata object is created by a preliminary execution of DROID. On

the other hand, Apache Tika offers methods for handling data objects by streaming the data without any need for accessing a file in the file system. This makes Apache Tika more adaptable to the Hadoop framework. To enable Apache Tika and DROID on Hadoop we have followed two paths and created tools, Hawarp and Nanite, which complement each other. Hawarp is based on copying files to the local storage on the worker node while Nanite uses streaming technology. They both use a text file for initial input. This text file contains a list of the full file system paths to the files to be characterised. These two different approaches are described in detail in the following two sections.

2.1 File characterisation using Hawarp

Hawarp is a collection of individual tools and in the following discussion we will differentiate between identification and feature extraction as these tasks are approached differently in Hawarp. Section 2.1.1 describes how file format identification can be accomplished while 2.1.2 focuses on feature extraction. Both tools need individual files to be available on the local storage of the worker node and the characterisation output is delivered in CSV files.

The fact that individual files must be available in order to apply the characterisation process limits the number of use cases where this approach actually makes sense. Especially when dealing with large amounts of content, as being required to move the data to another storage location usually takes a lot of time and might entail a significant amount of effort in terms of maintenance activities that are required to successfully shift data between different physical storage entities. Hawarp contains tools for making file objects available on the local worker nodes, but it is still very important to have reliable and robust clean-up processes as we are working on large scale—if something can go wrong, it *will* go wrong, and it *must* be handled.

2.1.1 File format identification of individual files using Hawarp

The hawarp/droid-identify and hawarp/tika-identify modules implement Hadoop jobs for executing DROID and Apache Tika respectively on a set of input files defined by a text file that contains the input paths to the files on a shared file server.

As presented in (Møldrup-Dalum et al. 2013), efficient loading of configurations and signature files can be achieved by having a singleton class that lives throughout the execution of a single TaskTracker task as the TaskTracker creates its own JVM instance. This singleton is instantiated in the map method of DroidIdentifMapper and TikaIdentifyMapper respectively.

In this sense a task refers to the set of files that is processed on one single processing core of a machine in the Hadoop cluster within an isolated JVM. Reusing the singleton created within this JVM is therefore limited to the number of files that are processed within the same task.

A special case, which has been explored extensively in SCAPE, is when files are stored in container files. Such container file formats can be the widespread ZIP or TAR formats or the more domain-specific formats like ARC and WARC. To access these packed files the hawarp/unpack2temp-identify module uses the Hadoop framework to parallelise the unpacking of the containers into a temporary directory of the worker nodes. Following this process, tools like hawarp/droid-identify can access these individual files. When using this technique only the TaskTracker that unpacked the files can process them and optimising by data locality has been lost.

There is also the caveat that files are completely written to temporary files, which creates a lot of I/O load for large archives and this is not always necessary because the file format identification process could be executed on small parts of the file stream. This will be examined in section 2.2 on Nanite

On the other hand this technique makes files packed in container files available to any kind of tool that requires the existence of actual files.

New container formats can easily be added by implementing the Container interface of the hawarp/unpack2temp-identify. To implement a container class a container must initialise a

bidirectional map by unpacking items out of the container file input stream. This bidirectional map must use some item identifier as key and a temporary file name pointing to an existing file as value. This identifier is required in order to be able to assign specific identification results to the files and analyse these individual files if need be. The container file does not need to have a flat structure; only the bidirectional map must be flat while the key can still be used to make a hierarchical structure explicit. This approach can therefore also be used to make nested file stream objects available as a set of individual files. In this way a video contained in a PDF which itself is contained in a ZIP file, can be made available in the temporary directory. New identifiers can be added by implementing the Identification abstract class from the same hawarp/unpack2temp-identify module. A requirement of such a new identifier is that it must provide methods for running the identification locally and on the Hadoop platform. This is to facilitate both command line and Hadoop user interfaces. Also the application configuration allows easily adding new identification tools by extending the same abstract class and adapting the configuration file accordingly.
In general identification tools can provide several different identifiers. For example, Apache Tika can be used only to determine the MIME type (e.g. image/png) while DROID can give both the MIME type and a PRONOM identifier (e.g. image/png and DROID PUID: fmt/11).
The output of the identification results is in tabular, comma delimited form, which allows easy import into applications for doing analysis or even, for large data sets, in a Hive or HBase table for Hadoop driven analysis. The output of the application can also easy be adapted by either changing the output syntax as documented in the application configuration or by writing a custom class implementing the OutWritable interface of the hawarp/unpack2temp-identify module. In such an implementation the command line application output and the Hadoop job key-value pair output have to be defined separately.
In order to give a simple example of how to use Hawarp for identification, let us assume we have the following container aggregating an single HTML file and with a single picture in PNG format: containerfile.zip (index.html, picture.png). Using Apache Tika, DROID and UNIX file for identification. Table 1 shows the produced output.

Table 1 Identification results from DROID, Apache Tika, and UNIX file

Identifier	Tool	Property	Value
containerfile.zip/index.html	DROID	MIME	text/html
containerfile.zip/index.html	DROID	PUID	fmt/96
containerfile.zip/picture.png	DROID	MIME	image/png
containerfile.zip/picture.png	DROID	PUID	fmt/11
containerfile.zip/index.html	DROID	MIME	text/html
containerfile.zip/picture.png	Tika	MIME	image/png
containerfile.zip/index.html	Unix file	MIME	text/html
containerfile.zip/picture.png	Unix file	MIME	image/png

In this example, DROID is the only tool that outputs two different properties: the MIME type of the file and the PUID (PRONOM unique identifier). The other tools only return the MIME type of the file.

This approach is not limited to file format identification properties as it could also be used to extract other properties and include these in the tabular output format. Hadoop can easily process the tabular text output format afterwards in order to create statistics and any kind of content properties related analysis.
An presentation of this solution, using an earlier version of the Hawarp modules, is presented in (Schlarb 2014).

2.1.2 Feature extraction using Hawarp

The hawarp/tomar-prepare-inputdata module is a tool to prepare web archive container files in the ARC format that are stored in a Hadoop Distributed File System (HDFS) in order to allow processing of the individual files by means of the SCAPE Platform tool ToMaR.
Similar to the approach of hawarp's file format identification module unpack2temp-identify the ARC container files are unpacked but Instead of unpacking them to the local file system of the worker nodes, they are unpacked to HDFS.
Figure 1 shows the process that starts with the tomar-prepare-inputdata module, which then receives a list of ARC container files as input ("ARC-Files-List"), process the ARC records with the File Information Toolkit (FITS) and generates output in the Hadoop sequence file format.

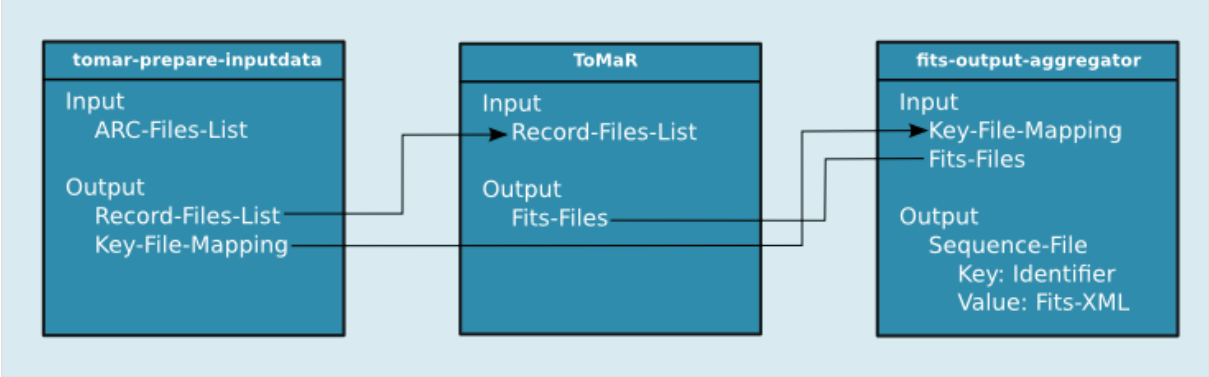

Figure 2 hawarp/tomar-prepare-inputdata module in relation to ToMaR and the intended final result

It is necessary to keep a list of file identifiers to record how files correspond to the ARC records. This map is the "Key-File-Mapping" in Figure 1. The subsequent tool ToMaR is agnostic to the record identifiers because it only receives a list of temporary files in form of a control file ("Record-Files-List") that is generated by "tomar-prepare-inputdata" and allows executing some operation on the unpacked file set. The "Key-File-Mapping" and the "Fits-Files" can then be aggregated to a SequenceFile output with the record identifier as key and the Fits-XML output as value. A complete workflow using this tool was presented in section 3.3.3 of (Schlarb, Palmer, and Medjkoune 2014) where the output is used to ingest data into C3PO.
As mentioned earlier it is trivial to implement the above for other container formats, but also the type of operation can trivially be adapted to use other tools and operations simply by changing the ToMaR configuration. This is described in detail in ("ToMaR" 2014).
Hawarp/ tomar-prepare-inputdata has two modes. When executed in local mode as a Java application, the input can be a directory containing ARC files (option -l/--local). Without this option the application is executed as a Hadoop job. In this latter case, the input directory contains text file(s) with absolute HDFS paths to the ARC container files. For example, hadoop jar ./target/tomar-prepare-inputdata-1.0-SNAPSHOT-jar-with-dependencies.jar -d /user/name/inputpaths/

will prepare the files listed in `/user/name/inputpaths` for the following ToMaR job. If the Hadoop job runs successfully, various files are created in several output directories. First, there is a directory where unpacked files are copied:

```
<HDFS USER HOME>/tomar-prepare-inputdata_unpacked/
```

Second, there is the job output directory, which contains the different output files aggregated in a timestamp directory. The default output directory is `tpi_joboutput` and it will contain the following files for Hadoop job with id `1385143157862`.

```
./tpi_joboutput/1385143157862/_SUCCESS
./tpi_joboutput/1385143157862/_logs
./tpi_joboutput/1385143157862/keyfilmapping-m-00000
./tpi_joboutput/1385143157862/part-r-00000
./tpi_joboutput/1385143157862/tomarinput-m-00000
```

The `keyfilemapping-*` file contains the container/record-identifier as key and the file name as value so that each unpacked file in HDFS can be clearly assigned to the corresponding record and the ARC container. The `tomarinput-*` file contains the input file which can be used as input by ToMaR (`-i` parameter). The `part-r-00000` is an empty reducer file that can be ignored.
Note that, depending on the Hadoop configuration, failed Hadoop tasks might be re-scheduled to run on another node. This can lead to a higher number of unpacked files than there are records in the container because Hadoop does not take care of cleaning up files that have been created by the failed task. The output files listed above keep track of the generated files, but any additional files caused by task failures will safely be ignored. It just means that in case of task failures some additional storage is required.
The job produces one output file per map task for the `keyfilemapping` and `tomarinput` output types. These files can be easily merged, e.g. for the `keyfilemapping` using the following command:

```
hadoop fs -cat ./tpi_joboutput/1385143157862/keyfilmapping-m-* | hadoop fs
-put - ./tpi_joboutput/1385143157862/keyfilmapping-aggregated.txt
```

And for the ToMaR input file correspondingly:

```
hadoop fs -cat ./tomar-prepare-
inputdata_joboutput/1385143157862/tomarinput-m-* | hadoop fs -put -
./tomar-prepare-inputdata_joboutput/1385143157862/tomarinput-aggregated.txt
```

In that way ToMaR can be invoked using the aggregated input file:

```
hadoop jar tomar.jar -i /user/onbfue/tomar-prepare-
inputdata_joboutput/1385143157862/tomarinput-aggregated.txt -r
/user/name/scape-toolspecs
```

where the value of the parameter `-i` is the aggregated ToMaR input file and the value of the `-r` parameter indicates the directory where the tool specification files for ToMaR can be found. See ("ToMaR" 2014) for further details. An experiment using the above technique has been described in detail in ("Web Archive FITS Characterisation Using ToMaR at ONB - SCAPE - Confluence" 2014)

2.2 Characterisation of web content using Nanite

As previously noted, Nanite takes a different technical approach than Hawarp. As with Hawarp, the Nanite project is written in Java but instead of containing a set of separate modules, Nanite has just two main modules

Nanite-core: An API for Droid that can process files via InputStreams, obviating the requirement for files.
Nanite-hadoop: A MapReduce program for characterising the content of (W)ARC files.

Nanite is optimised for characterising web archive data; especially data stored in ARC and WARC container files.
Nanite-Hadoop makes good use of data locality within a Hadoop cluster and uses RecordReaders[1] as a means to pass individual records from (W)ARC files for processing. These RecordReaders are created by the British Library's Web Archive project outside of the SCAPE project. The individual records are buffered in memory and streamed, and are not written to disk at any point making this approach much less I/O bound than the Hawarp project.
Nanite's approach of being virtually completely pure Java (utilisation of the Java native interface (JNI) for Libmagic integration being an exception) and of leveraging Hadoop technologies ensures that it is a performant way to characterise the content of (W)ARC files. RecordReaders for other container file formats could be used to make use of the existing characterisation logic with a simple change of input format.
The various options for outputs for identification and feature extraction within Nanite-Hadoop are as follows:

- Year the record was harvested
- MIME type as reported by the original web server
- Normalised file extension from URL, where possible
- MIME type from Apache Tika
- MIME type from Droid
- MIME type from Libmagic[2] (effectively the *nix `*file*` tool)
- MIME type following Apache Tika parsing
- A SequenceFile containing Tika Parser Metadata objects for later analysis (one SequenceFile per input (W)ARC, one entry per input record)
- A zip file containing serialized Tika Parser Metadata objects for later analysis (one zip per input (W)ARC, one entry per input record)
- A zip file containing C3PO compatible output from the Tika Parser Metadata for later analysis (one zip per input (W)ARC, one entry per input record)

The text outputs from the reduce phase in Nanite-Hadoop are written as tab-separated values in the final output. The SequenceFile and ZIP outputs are written to HDFS during the map phase in the MapReduce job.
The actual output fields in the reduce phase output file, and which SequenceFile and ZIP are to be generated can be configured before the Hadoop job is run. The latter types of output can influence the performance so only the necessary types should be configured.
When integrating the Tika Parsers, used for feature extraction, it was found that they could hang or crash the main MapReduce process. The method we found to handle this was to isolate the Tika Parser as a web service in a separate operating system process, one per map. That work is called Process Isolated Tika[3] and a discussion of developing this feature has been described in A Tika to ride; characterising web content with Nanite (Palmer 2014).

[1] https://github.com/ukwa/webarchive-discovery/tree/master/warc-hadoop-recordreaders/
[2] Libmagic is the C library behind the UNIX file tool. SCAPE has produced a JNI based wrapper to access this library from with Java programs: https://github.com/openplanets/libmagic-jna-wrapper
[3] https://github.com/willp-bl/ProcessIsolatedTika

It is worth noting the following issues that occur due to Nanite's approach. The core functionality of Nanite-Hadoop is container agnostic, and relies on being able to produce an InputStream for file data, however a RecordReader would need to be provided for container types other than (W)ARC. Secondly, if there is an issue with an input (W)ARC file, such as zero length file or corrupt compression, Nanite may crash. Introducing a separate MapReduce program that assesses the input files before a Nanite-Hadoop run has mitigated this. This assessment step runs very quickly and it ensures that a full characterisation run does not fail after several hours of runtime due to a problem with an input file. Such a verification step could be enhanced to check for other aspects of the input where there may be issues, such as malformed records.
The code has been tested successfully at large-scale on at least two different Hadoop clusters with 1TB and 15TB datasets, with the latter being written up in A Weekend With Nanite (Møldrup-Dalum 2014). Additionally, Nanite-Core is in use at the UK Web Archive in their Hadoop warc-indexer software[4].

3 Conflict resolution and normalisation of extracted features

When different characterisation tools originating from different domains, organisations, user-bases, etc., are used over the same content, the results are often conflicting and even contradictory. This leads to problematic quality of the results and makes it difficult to conduct high-quality, in-depth analysis on larger collections in a way that is suitable for topics like planning and watch.
To mitigate this problem, we evaluated and developed alternative methods to improve the quality of characterisation results by reducing conflicting measurements using a combination of controlled vocabularies, tool adaptation and post-processing techniques. We have identified three main sources of such conflicts:

1. Conflicts derived from inconsistent naming, i.e. one tool reports a file as a "pdf ver. 1.4", while another tool reports the file as "Portable Document Format version 1.4".
2. Different tools provide competing characterization results, i.e. one tool reports a file as a "pdf ver. 1.4", while another tool reports the file as "TIFF".
3. The last source of conflict is more related to the format identifiers. It may be difficult to detect differences between different but related properties. One tool may report a file as application/xml while another reports the file as application/xhtml+xml. Such two results could be deemed completely different or alike, when, in reality, the latter is a refinement of the former.

To solve this we have identified three promising techniques.

Alignment to a common vocabulary: Such a vocabulary may address the first source of conflicts according to our list. There will be no space for inconsistent naming and it will give us a solid ground for further introduction of new characterization tools, once a single mapping of properties is done. Planning and Watch also benefits from such a vocabulary as it provides a common language to describe digital objects and their features.

A rule framework: There are cases when tools provide different results without any clear definition on which one could be the best. Therefore a decision on final value has to be made. This framework is essential as we may define rules to make decisions in different complex situations when there are additional factors to consider.

Prioritisation of characterisation tools: According to a target digital object, it may be useful to specify a single characterization tool for a concrete file format which it supports best.

[4] https://github.com/ukwa/webarchive-discovery/tree/master/warc-indexer

Based on the above discussion we have created two software components

- A conflict reduction plugin for C3PO.
- Vocabulyzer—command line tool for normalising against a common vocabulary.

3.1 The conflict reduction plugin for C3PO

The C3PO plugin is based on the Drools rule framework. Drools is in ("Drools Documentation" 2014) described as "a business rule management system (BRMS) with a forward chaining inference based rules engine, more correctly known as a production rule system, using an enhanced implementation of the Rete algorithm". It is developed by Red Hat and has a strong community support within its usage domain.

Drools is able to take action according to predefined prescriptions or rules. In order to construct such a rule, four elements are required:

- Name
- Salience. It defines a priority. A rule with higher salience value is executed first.
- When clause. A conjunction of statements that define a trigger.
- Then clause. A java-like code to execute.

An example of such a rule is presented in the following quotation:

```
rule  "set format GZIP to GZIP Format from Exiftool"
      salience 160
      when
            $e : Element ()
            $md : MetadataRecord (
                  property.id == "format",
                  value == "GZIP",
                  util.isFromTool (this, "Exiftool")
            ) from $e.metadata
      then
            String newValue = "GZIP Format";
            $md.setValue (newValue);
            modify ($e) {
                  getId()
            }
end
```

This rule triggers, when

1. An element exists (and binds it to `$e`)
2. If the element `$e` has any `MetadataRecord` in its metadata property (a list of `MetadataRecords`), that suffices the following conditions:
    a. Its properties ID is equal to "format" and
    b. Its value is equal to "GZIP" and
    c. Execution of the external Java method `isFromTool` on the global variable `util` returns true for the given `MetadataRecord` and the parameter "Exiftool".

Once a rule triggers, 'then'-section is executed. In the example, this section sets the format to the new value "GZIP Format".

Currently there are five rules implemented in the C3PO plugin.

Update resolved conflicts: If all metadata records for a given property are marked as conflicting, but all values are equal, merge them to a single record and update the status to SINGLE VALUE. This rule has the highest priority, so it is always applied first after an element has been changed.

Merge conflicting records of equal value: If some, but not all, conflicts on a property can be resolved, the records reporting the same value are merged together. This is a convenience rule to ensure that distinct conflicting records have distinct values. It has the second highest priority.

Split conflicting record with multiple sources: If no more matching rules are found, this rule takes any metadata record with conflicts and more than one source. This metadata record is then split up into several records, one for each source. This is the inverse operation of the merge rule and is necessary, because the C3PO backend data structure is not able to store more than one source per conflicted record. The splitting is not only necessary because of prior merging, but also because the initially parsed data from FITS sometimes contain such merged conflicting results (see analysis below). It is important that this rule is does not mark the element as modified to avoid loops between the merge and the split rule.

Report conflicts: At the very end of rules processing, an element ($e in the above code listing) that has metadata records is reported for later analysis.

Report non conflicting elements: If the $e element has no conflicts, this rule is triggered which is mostly used for debugging and analysis (Rötzer and Schmidt 2013).

In order to evaluate the plugin, a dataset was generated from a subset of FITS characterization results of SB Web Archive Data. A desktop PC with Intel 4 cores, 8GB RAM, 250GB Hard Drive and Ubuntu 13.04 OS was used for the tests.
We took the 10 first sets of characterisation data from each year of the collection from 2005 until 2012. After extraction of these packages, we obtained a number of FITS characterization results per year, as presented in Table 2.

Table 2 Amount of FITS characterisation results per year

Collection Year	2005	2006	2007	2008	2009	2010	2011	2012
Size of a Collection	30846	19846	19918	361	43944	26820	2128	1941

We then ran C3PO including the Drools Rule Framework plugin. The plugin is called every time a new characterization result is fed into C3PO. Figure 3 demonstrates that 50-70% of detected conflicts are resolved using this plugin.

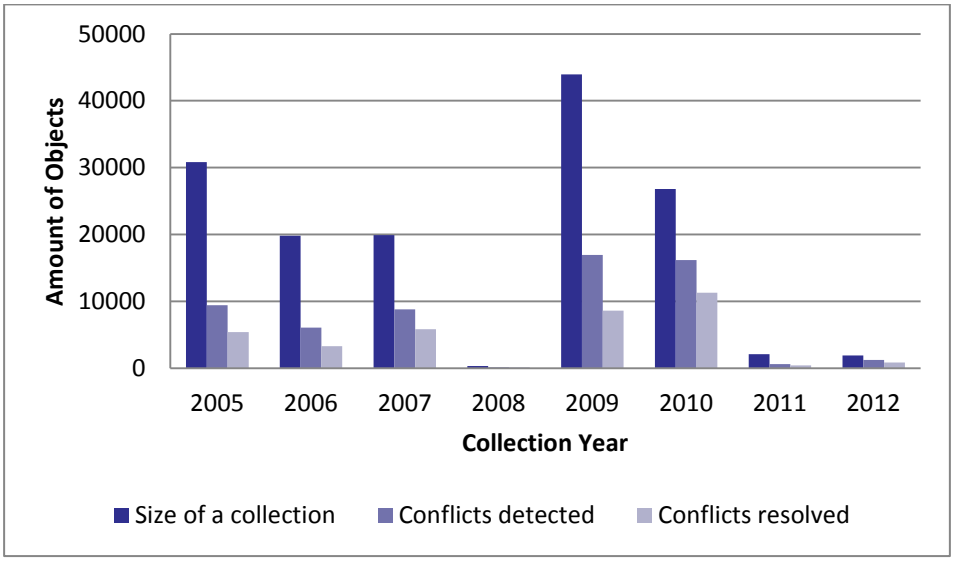

Figure 3: Results of experiments on conflict reduction

The run time of C3PO plugin was also evaluated. Figure 3 shows how much time is needed to ingest a collection of FITS characterisation results into C3PO. This time includes reading from a hard-drive, pre-processing of data, saving results in MongoDB and post-processing. To examine the cost of using the conflict reduction plugin, we ran the ingest process twice on a set of arbitrary collections with increasing size. Figure 4 illustrates that the conflict reduction comes with a cost of 25% increase in runtime. We have argued that the described conflicts are destructive to the quality of the data and a 25% increase in runtime is an acceptable cost for much better data quality.

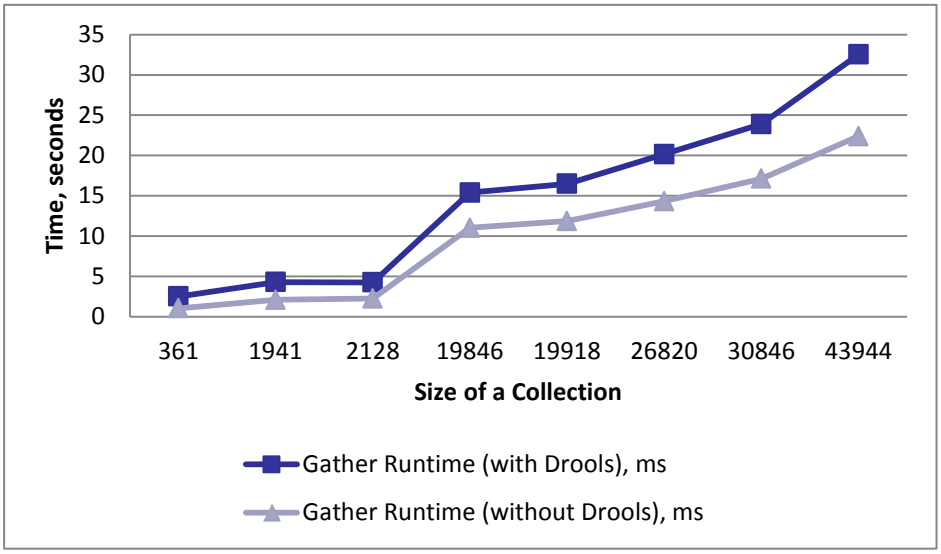

Figure 4: Execution time vs. size of collection

3.2 Vocabulyzer

Vocabulyzer is a Java command line application that enriches characterisation information that is expressed using XML. The tool performs this task by annotating the XML elements in the original data with entities from a given vocabulary. The intention of Vocabulyzer is to link general

characterisation data with the SCAPE Ontology developed in the Planning and Watch sub project ("Openplanets/policies" 2014; Sierman, Jones, and Elstrøm 2014).
If a given tool generates the following XML snippet describing the height and format of an image

```
<out>
      <property name="image height">99</property>
      <property name="format">JPEG</property>
</out>
```

we can use Vocabulyzer to improve that data. To perform this task, Vocabulyzer needs a configuration describing a mapping between the tool output and the vocabulary. Looking at the above XML, we observe that we need the vocabulary entities describing image height and image file formats. In the SCAPE ontology ("Openplanets/policies" 2014) we identify e.g. image height as http://purl.org/DP/quality/measures#52. When corresponding ontology entities are identified for all properties we can run Vocabulyzer on the tool output. If we run it on the above XML snippet with just image height in the mapping table, we get the following

```
<out>
    <property name="image height" uri="http://purl.org/DP/quality/measures#52">
      99
    </property>
    <property name="format">JPEG</property>
</out>
```

The Vocabulyzer tool is in an early, almost proof of concept, state and while filling a gap in the tool chest of characterisation tools, its usage and functionality will probably change when it is taken into production.

4 Policy driven validation

In the previous two sections we have presented how to characterise huge amounts of data and how to increase the quality of the generated metadata. Given high quality characterisation metadata we can use that for multiple preservation actions one of which is described as policy driven validation. Policy driven validation covers the process of comparing metadata from a digital file object against an institutional policy. To perform such a task the policy must be expressed in a machine-readable format and this is exactly what the previously mentioned SCAPE ontology creates a basis for. In (Sierman, Jones, and Elstrøm 2014; Sierman 2014) a three level policy framework is presented including the machine-readable control policy representation of institutional policies. Building on this work we have implemented a proof-of-concept implementation illustrating how policy driven

validation could be performed. The implementation is illustrated in Figure 4

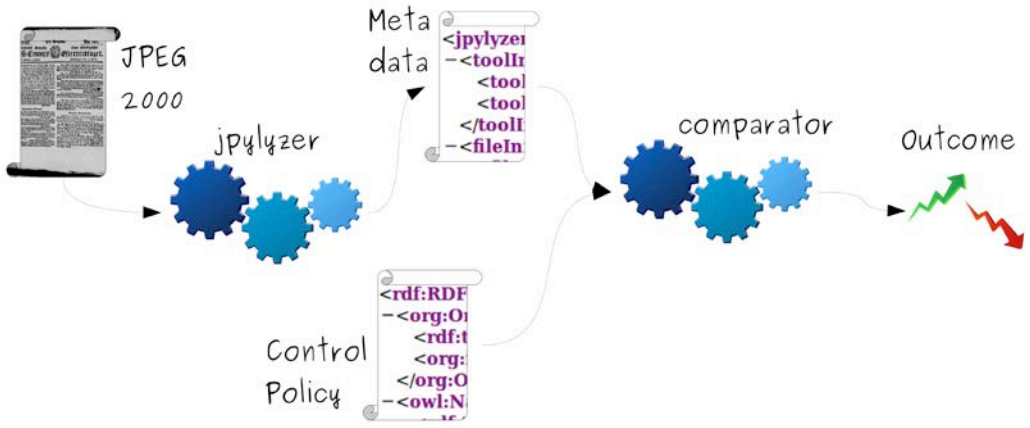

Figure 5 Schematic overview of policy driven validation

The scape-jp2-qa tool[5] implements an initial version of the comparator step in Figure 4. This implementation is presented in detail in (Jurik et al. 2014)
The comparison process consists of two steps; the first reads and parses the institutional control policy, and the second matches each element of the extracted metadata to the elements of the control policy. The main entry points in the implementation can be seen below.

```
public List<ContentProfile> generateContentProfile(String contentMetadata)
public Policy readPolicy(File policyFile)
public boolean compare(
     Policy organisationPolicy,
     List<ContentProfile> contentProfileList)
```

For each pair of elements we do a comparison of Measure, ControlPolicyType, Modality, Qualifier, and Value.

Measure A Unique URI taken from the PW Ontology
ControlPolicyType Format Objective, Authenticity Objective, Action Objective, Representation Instance Objective, Access Objective
Modality *must* or *should*
Qualifier greater than, lower than, equal, greater or equal, lower or equal
Value a value complying with Scale defined on Measure

This process generates metadata about each object describing how it adheres to an institutional policy and this information can be used for e.g. rejection of an object if done during ingest or, more practical, preservation alongside the rest of an objects metadata for later referral.

5 SCAPE Azure platform

This section describes a study that measured the speed of the Apache Tika Content Analysis Toolkit and the DROID File Format Identification Tool when they were run on a Microsoft Azure virtual

[5] https://github.com/statsbiblioteket/scape-jp2-qa

machine. The results are compared to the speed of the same tools running on a traditional on-site server.
In an earlier study, SCAPE researchers evaluated and selected characterisation tools and measured their performance (Raditsch et al. 2012). The ability of Tika and DROID to accurately identify file formats was determined by running them against a set of approximately one million files in the Govdocs1 corpus and comparing the results to a *ground truth* provided by Forensic Innovations, Inc. The speed of the tools was measured while running them on an on-site server.
This newer study measured the speed of these tools when they were run in a cloud environment, namely on a Microsoft Azure virtual machine (VM). (No effort was made to rerun the accuracy tests, as there is no reason to believe that the accuracy of the same tools would vary on different servers.) Tika and DROID are used today in an Azure-based SCAPE API Service implemented by Microsoft Research and might be used in future Azure-based projects, so the speed of the tools when running in the cloud is an important consideration.
The procedure this study used to measure the speed of the tools adhered reasonably closely to what was done in the earlier study. The one concession to expediency was the use of a subset of about 12,000 randomly chosen documents instead of the entire corpus of one million.

5.1 Results

The speed at which Tika and DROID identified file formats is shown in Table 3 below, along with the speed of an MD5 calculation utility operating on the same files. Calculating a file's MD5 checksum involves reading the entire file while performing very few calculations, so the MD5 numbers effectively compare file access times between the two servers. The results are in files per second, so larger numbers are better.

Table 3 Files processed per second

	On-site server	Azure VM
Tika	61	659
DROID	47	65
MD5	42	426

The test results are presented in detail in (Vujic 2014b). Below is a summary of these results.

Table 4 Assessment of measurable points for Apache Tika

Metric	**Description**	***February 04, 2014***	***May 21, 2014***
NumberOfObjectsPerHour	Number of objects processed in one hour	2371809	2447712
ThroughputGbytesPerHour	The throughput of data measured in GB per hour	1201.8	1240.2
NumberOfFailedFiles	Number of files that failed in the workflow	N/A	0

Table 5 Assessment of measurable points for DROID

Metric	**Description**	***February 04, 2014***	***May 21, 2014***
NumberOfObjectsPerHour	Number of objects processed in one hour	234472	217809
ThroughputGbytesPerHour	The throughput of data measured in GB per hour	118.8	110.4
NumberOfFailedFiles	Number of files that failed in the workflow	N/A	1727

Table 6 Assessment of measurable points for MD5

Metric	Description	*February 04, 2014*	*May 21, 2014*
NumberOfObjectsPerHour	Number of objects processed in one hour	1532443	1340888
ThroughputGbytesPerHour	The throughput of data measured in GB per hour	776.5	679.4
NumberOfFailedFiles	Number of files that failed in the workflow	N/A	0

The huge difference in the results for the MD5 utility of more than an order of magnitude indicates that a much faster file system was used on the Azure VM. Indeed, the Govdocs1 files were put directly on the VM's hard drive, whereas the files in the on-site study were mounted on a separate Network File System server accessed over a network. If the files in the earlier study had been put directly on the server (which probably wasn't possible), the on-site performance would likely have been much better. It must be noted that if one of the other storage options for Azure had been used BLOB storage or SQL Server database, for example the Azure performance would likely have been much worse. This suggests that if a very large collection of documents needs to be identified quickly, the location of the files is important. It's likely that the difference in Tika performance is also due primarily to storage differences. However, the DROID numbers seem odd, at least at first glance. DROID ran faster on the Azure VM, but certainly not an order of magnitude faster. Further investigation would be required to prove this, but a possible explanation is that MD5 and Tika are IO-bound, while DROID is CPU-bound. In other words, DROID spends more time calculating than reading, so much so that even a 10X increase in reading speed results in only a moderate increase in overall speed.

Note that different CPUs were used in the two servers: The on-site server used dual Xeon X5670 CPUs running at 2.93 GHz, while the Azure VM used two cores of a six-core AMD Opteron 4171 HE running at 2.09 GHz. It's not clear which should be faster, but in any case the storage differences in the two studies clearly had a much larger effect that any CPU speed differences.

The results can be easily summarized: When the files that need to be identified are local to the machine, Tika and DROID demonstrates good run-time performance on a Microsoft Azure VM.

A few additional facts became apparent during the study. First, Tika and DROID can be easily used in the Microsoft Azure environment. They were run on an Azure VM for this study, and they are currently used in an Azure Web Role in the SCAPE API Service. In both cases, no serious problems were encountered while deploying the tools.

Second, the scalability inherent in Azure makes it easy to allocate hardware as needed. If the VM's hard disk isn't large enough to hold the files that need to be identified, a larger disk can easily be attached. Or if one VM is inadequate, duplicate VMs can be easily created. Those are manual steps, but scaling can also be done dynamically in response to changing demands by utilizing Azure's auto-scaling features. This is all much easier than trying to scale up physical hardware in an on-site environment.

6 Information Extraction on Text and Web Corpora

This section describes the work in the fields of text and Web (HTML) page characterization carried out in year 3 of SCAPE. The goal was to extract information relevant to the domain of digital content preservation by using Information Extraction technologies on large collections of natural language text crawled from the Web. The previous deliverable presented an overview of the Information Extraction system developed within SCAPE and discussed its capabilities to discover and extract information without human supervision, i.e. fully automatically. In addition, we described an

application of the system to a use case from the National Library of the Netherlands in which we automatically identified publishers and publications (such as journals) from the Web in order to aid preservation watch (Faria et al. 2013).
In this year, we have greatly extended the system to allow for more direct human specification of an information need. This allows users to specify the type of structured information they seek to find, prompting the system to semi-automatically create the necessary information extractors. The system is designed to accommodate novice users that initially operate with a broad or fuzzy definition of their problem. The users are exposed to an exploratory guidance that minimizes the necessary user interactions and thus the effort required to create custom relation extractors. Our system implements a novel paradigm that we have called "*Exploratory Relation Extraction*" (ERE). A full paper publication introducing the paradigm and the system has been accepted to the 25th International Conference on Computational Linguistics (COLING) (Maqsud et al. 2014), and a fully functional Web toolkit has been made available online for users to try at lucene.textmining.tu-berlin.de.
In the following, we introduce both the system and the demonstrator, and give an overview over challenges we have addressed.

6.1 Exploratory Relation Extraction

In this section, we present our approach for Exploratory Relation Extraction. We provide details on how we define extraction patterns and how we enable exploratory analysis of structured information contained in text by pre-emptively extracting all sub trees in dependency trees from a given text corpus. We then outline a data-guided incremental workflow to explore the indexed data for relations and illustrate this with an exemplary execution.

6.1.1 Human-Readable Relation Extraction Patterns

We based our work on the Propminer (Akbik, Konomi, and Melnijkov 2013) and KrakeN (Akbik and Löser 2012) systems we presented in the preceding deliverables. Much like in these works, we defined relation extraction patterns as sub trees in dependency-parsed sentences. While in our previous work we designed both an unsupervised system that automatically groups these patterns to discover structure, as well as a supervised system in which a user manually crafts extraction patterns using examples, for our final system we have chosen a middle way: We aim to combine the explicit *control* given by the supervised approach with the *comfort* of the unsupervised approach, thus giving a best-of-both-worlds approach to Information Extraction.
In our work, we followed the idea of *Pre-emptive Information Extraction* in which every possible relation for a given text corpus is pre-emptively generated in advance. Applied to our problem this meant that we generated all possible dependency sub trees found in a text, arguing that depending on the user's information need, any such pattern may be valuable. Since we are interested in binary relations, we generated only those sub trees that span two named entities in a sentence. In addition, we also determined the fine-grained entity types for named entities in order to allow users to optionally restrict patterns to match only entities of certain types. Our previous work (Akbik et al. 2013) has shown the benefit of including fine-grained type restrictions into patterns.
Please refer to Figure 6 for an example of the pre-emptive sub tree generation process. Here, we have an example sentence for which we determine all sub trees that span the indicated entity pair. In the sub trees, we replace the entity tokens with the placeholders "X" and "Y", where the former is the placeholder for the X-entity and the latter the placeholder for the Y-entity. For better human-readability, we lexicalize the patterns by lemmatizing the words and discarding information on typed dependencies. We also link the entities in the sentence to entries in the FREEBASE knowledge base, allowing us to retrieve their fine-grained entity types.

We then index the information on lexicalized patterns, the entities they span and their types, as well as the sentences in which the patterns were found (Figure 6D). This allows users to query for any combinations of patterns and entity type restrictions and retrieve matching entity pairs and sentences from the index. For instance, a user may query for all entity pairs that match the "at young age X enter Y" pattern, and optionally restrict the Y-entity to be only of type ORGANIZATION, or more specific types such as CHURCH or UNIVERSITY. We argue that because patterns are lexicalized variants of dependency sub trees and entity type restrictions can have human readable names, such queries are intuitive to users even without a background in Natural Language Processing. The use and pre-emptive indexing of human-readable patterns decreases the entry barriers into the ERE process, as this enables users to query parsed text corpora.

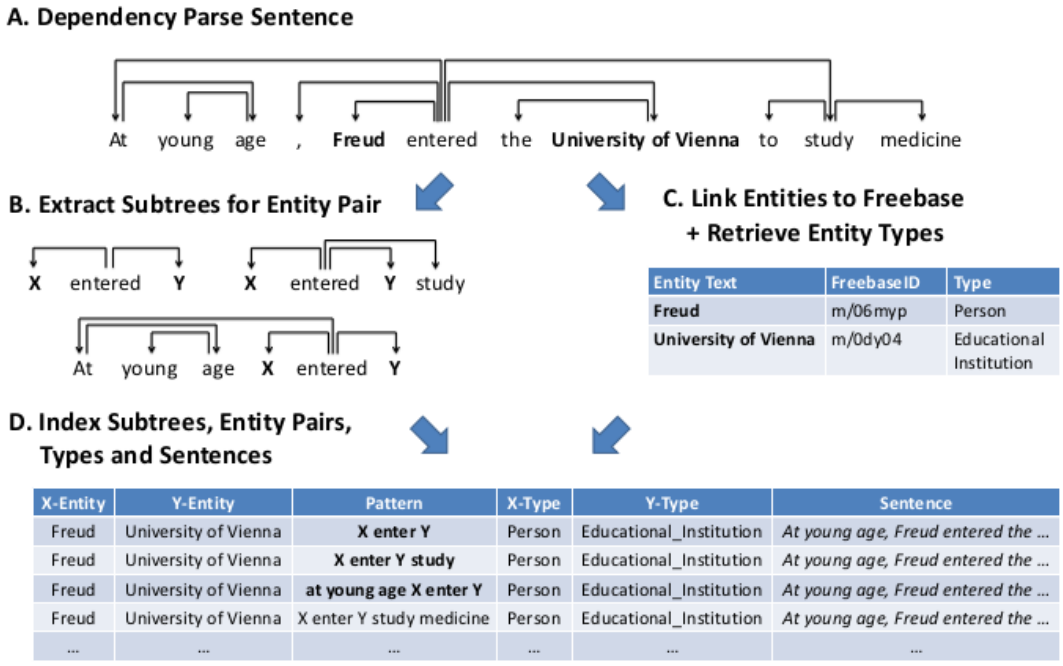

Figure 6 Illustration of the sub tree generation process.

6.1.2 Exploration Workflow

A second key component of our approach is the provision of guidance in the exploration process by computing suggestions for patterns from user input and enabling an interactive workflow that allows users to work with available data. Such guidance is needed for two reasons: first, though much effort is invested in human-readable extraction patterns, users may need support in formulating patterns and choosing entity type restrictions. This is especially the case when users are non-experts in the domain of interest and they strive to identify a range of appropriate patterns. Second, users may be uncertain of the information content of a given text corpus. By providing guidance through automatic pattern suggestions that reflect available information, we help users find patterns for their information need.

Users formulate an *entry point* to launch the exploration process, either by providing entity types, patterns or both. We guide the formulation of this initial query through autocomplete options. If the user enters only types for the entities, the system offers the most common patterns that are observed between entities of these types. The user can also search for patterns that contain a certain keyword.

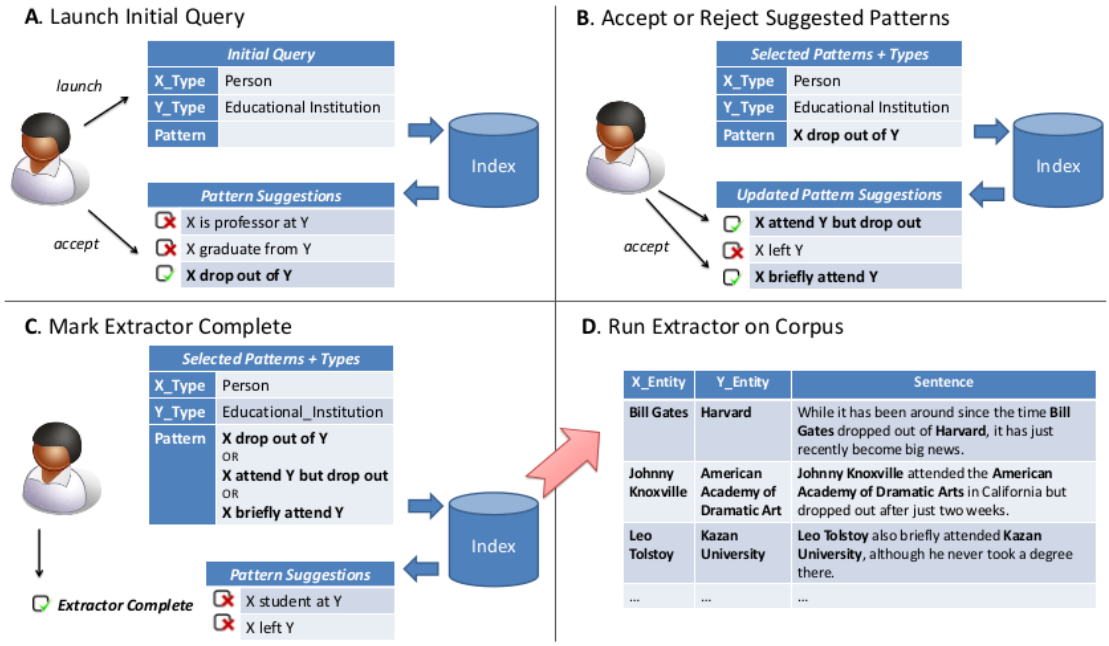

Figure 7 Illustration of the exploratory relation extraction process.

Suppose we have a user who is given a large text corpus and is asked to link persons to their respective educational institutions, but is unsure of what type of relevant information may be found in the corpus. Knowing only that relations should hold between entities of type PERSON and entities of type EDUCATIONAL INSTITUTION, the user starts an exploration process by providing only these entity type restrictions. This is illustrated in Figure 7A.

A query is run against the index that identifies common patterns that hold between entities of such types, including “X be professor at Y”, “X study at Y” and “X drop out from Y”. Recall that each pattern is a human-readable version of a sub tree in a dependency tree with two placeholders for entities, namely “X” and “Y”. These placeholders may match named entities of any type, or can be restricted to matching only entities of certain types such as persons, organizations or locations. By clicking on a pattern, the user retrieves entity pairs and sentences in which a pattern matches; For example, the user is informed that the pattern “X study at Y” finds the relation instance <Bill Gates, Harvard University> in the sentence “Bill Gates briefly studied at Harvard University.”

Intrigued by the pattern “X drop out from Y”, the user affirms this pattern and rejects all other suggestions. This causes a new query to be run against the parsed data, this time consisting of the entity restrictions as well as the pattern. As the query is now more concrete, the pattern suggestions are updated to reflect this new information. The user is presented with similar patterns such as “Y dropout X” and “X attend Y but drop out”. This is illustrated in Figure 7B.

The user repeats this, selecting or de-selecting patterns (Figure 7B). At each interaction, suggestions are updated to reflect the current selection. When the user is satisfied with the identified relation, the selected set of patterns and restrictions is saved as an extractor (Figure 7C) and executed against the entire text corpus (Figure 7D). This returns lists of matching relation instances and sentences. So, in this example, the user has started with an imprecise information need and has identified a relation of interest in a given text corpus, namely a relation for persons that attended an educational institution but did not graduate. This workflow is possible with minimal effort and yields custom relation extractors.

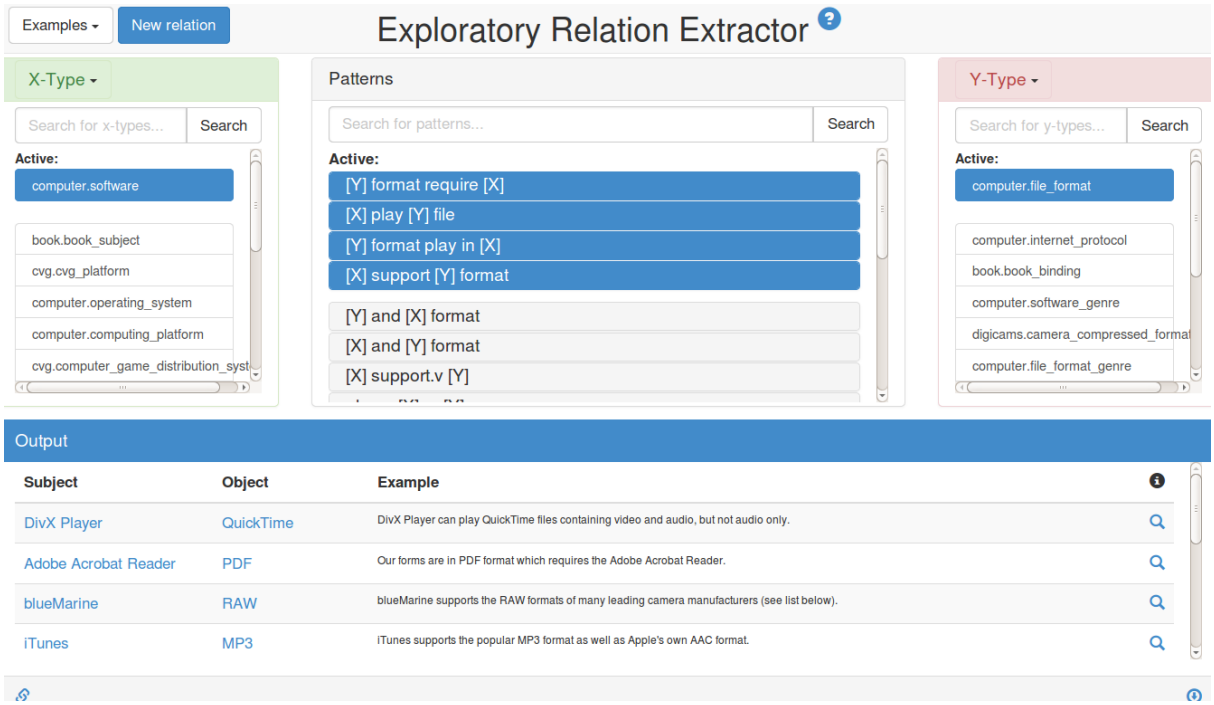

Figure 8 Screenshot of the main screen of the Web Toolkit

6.1.3 Online Web Toolkit

In order to enable the discussion of our approach with the digital preservation community, we have made a fully functional Web toolkit available online at lucene.textmining.tu-berlin.de. The toolkit enables users to execute the workflow we described in this deliverable and to export the identified structured information in the form of lists. A screenshot of the demonstrator is illustrated in Figure 8. A few example relation extractors are available through the examples function in the top left of the main screen. One of them mirrors the extractor we created for the National Library of the Netherlands, an extractor that finds pairs of publishers and publications in Web text. Another example extractor finds information on file types and the software that supports these file types. By clicking the "download" bottom at the lower right corner of the main screen, the extraction results can be downloaded as a text file for local consumption. Extractors can also be saved and be shared as links between different users by clicking the "generate link" button at the lower left corner of the main screen.

The system currently runs on a large portion of the ClueWeb09 corpus, a very large reference corpus for large scale Web analytics. As indicated in the previous deliverable, we have achieved the scale-up of the system by porting the feature extraction steps to Apache Pig.

We have used the Web toolkit in dissemination activities to discuss the scope of problems within automated preservation watch that Information Extraction approaches can address. One item of particular interest in the community is to use our method to create an ontology of file formats, licences, compatible software and supporting organizations with the intent of showing that our method can support collaborative efforts like PRONOM or Open Document[6] in building up repositories of machine-readable data.

[6] http://fileformats.archiveteam.org/wiki/OpenDocument

6.2 Conclusions on Information Extraction on Text and Web Corpora

We have systematically investigated the viability of the use of Information Extraction methods from Web text to assist automated preservation watch and other challenges within the SCAPE project. We have researched, designed and implemented a number of Information Extraction systems focusing on *a)* lowering the entry barriers for users without a background in text mining technologies, *b)* maximising automation – or minimising the required effort – through the use of unsupervised machine learning techniques, while *c)* finding a way for users to input information needs to direct the system to explore and discover structured information of interest. Our research has led to the formulation of the *Exploratory Relation Extraction* paradigm, as well as design of the fully functional *Propminer* system, which now is publicly available through a Web toolkit.

Our system can be used to create custom ontologies on arbitrary topics. Users may use it to freely define extractors to identify machine-readable data in text to assist preservation watch or other information needs within preservation science. We believe our system to be a significant step towards preservation watch using Information Extraction from the Web.

7 End notes

We have in this report shown great progress toward the scalability goals of SCAPE. We have taken three major tools for identification and feature extraction and shown how to employ them on a scale-out infrastructure, both traditional Linux servers and the Microsoft Azure platform.

We have showed how to enhance the data quality created by the existing characterisation tools and how that data can be used for policy driven validation.

We have also explored how information available on the World Wide Web can benefit the community of digital preservation using semantic information extraction. We have created a web application that supports an explorative and iterative workflow, making this information available to curators and researches in the fields of digital preservation as well as in general research in digital humanities.

8 Glossary

Apache Tika™	Toolkit from the Apache Software Foundation that detects and extracts metadata and text content from various document formats. Apache Tika™ is an unregistered trademark of the Apache Software Foundation. http://tika.apache.org
C3PO	'Clever, Crafty, Content Profiling of Objects' is a software tool, which uses metadata extracted from files of a digital collection as input to generate a profile of the content set. http://openplanets.github.io/c3po/
Characterisation	The concept of identification, feature extraction, and validation
DROID	A tool created by The National Archives that can automatically profile a wide range of file formats. http://www.nationalarchives.gov.uk/information-management/manage-information/policy-process/digital-continuity/file-profiling-tool-droid/
Drools	A Business Rules Management System (BRMS) solution. Used by the conflict reduction plugin for C3PO. http://www.drools.org
Feature extraction	The process of reporting the intrinsic properties of a digital object that are preservation planning and action
FITS	The File Information Tool Set (FITS) identifies, validates, and extracts technical metadata for various file formats. It wraps several third-party open source tools, normalizes and consolidates their output, and reports any errors. The Harvard University Library Office created the tool set for Information Systems. http://projects.iq.harvard.edu/fits
Hadoop, Hadoop Framework	Framework for processing large data sets on a computer cluster. http://hadoop.apache.org
Hawarp	Hadoop-based Web Archive Record Processing. This project consists of a collection of tools to process web archive records using the Hadoop framework. https://github.com/openplanets/hawarp
HBase	The Hadoop database, a distributed, scalable, big data store. http://hbase.apache.org
HDFS	The Hadoop Distributed File System is the underlying abstract file system required by the Hadoop platform.
Hive	A data warehouse software that facilitates querying and managing large datasets residing in distributed storage. https://hive.apache.org
Identification	The process of determining the presumptive format of a digital object
Jpylyzer	A validator and feature extractor for *JP2* images. *JP2* is the still image format that is defined by Part 1 of the JPEG 2000 image compression standard (ISO/IEC 15444-1). http://openplanets.github.io/jpylyzer/
KRakeN	An OIE system specifically designed to capture N-ary facts, as well as the results of an experimental study on extracting facts from Web text in which we examine the issue of fact completeness (Akbik and Löser 2012).
MapReduce	A programming model using a map phase for transforming input data, one unit at a time, and a reduce phase for collating the transformed data. See http://en.wikipedia.org/wiki/Mapreduce
MongoDB	An open-source document database, and one of the leading NoSQL databases. Used as a database backend in C3PO
MyExperiment	A collaborative environment where scientists can safely publish their workflows and *in silico* experiments, share them with groups and find those of others. Used by SCAPE Project for publishing preservation components and workflows. http://www.myexperiment.org
Nanite	The Nanite project builds on DROID and Apache Tika to provide a rich format identification and characterization system. Created by the British Library. http://openplanets.github.io/nanite/
Plato	A web-based tool that creates a Preservation Plan and provides a user interface for viewing, managing and updating that plan. The plan itself is stored in the Plan Management Service after creation. http://www.ifs.tuwien.ac.at/dp/plato/intro/
PRONOM	PRONOM is an information system about data file formats and their supporting software products. See https://www.nationalarchives.gov.uk/PRONOM
Propminer	A Workflow for Interactive Information Extraction and Exploration using Dependency Trees (Akbik, Konomi, and Melnijkov 2013)
PW Ontology, SCAPE Ontology	The SCAPE Ontology is an OWL ontology that formally defines the terms used by computing systems in SCAPE. (Sierman, Jones, and Elstrøm 2014) and

	https://github.com/openplanets/policies/
SCAPE Execution Platform	An infrastructure that provides the computational resources to enact a Preservation workflow and execute Preservation actions. Abstracted into three layers: the Execution Environment; the Job Execution Service and the Job Submission Service API (May and Wilson 2014)
Scout	A preservation watch system being developed within the SCAPE project. It provides an ontological knowledge base to centralize all necessary information to detect preservation risks and opportunities. http://openplanets.github.io/scout/
Target platform	Ubuntu 12.04
ToMaR	Provides a simple and flexible solution to run preservation tools on a Hadoop MapReduce cluster in a scalable fashion. http://openplanets.github.io/ToMaR/
Validation	The process of determining the level of conformance of a digital object to the normative syntactic and semantic rules defined by the authoritative specification of the object's format
Vocabulyzer	A tool to align XMLs to a given vocabulary. https://github.com/openplanets/vocabulyzer

9 References

Akbik, Alan, Oresti Konomi, and Michail Melnijkov. 2013. “Propminer: A Workflow for Interactive Information Extraction and Exploration Using Dependency Trees.” In *The 51st Annual Meeting of the Association for Computational Linguistics, ACL*.

Akbik, Alan, and Alexander Löser. 2012. “KrakeN: N-Ary Facts in Open Information Extraction.” In *The Knowledge Extraction Workshop at NAACL-HLT*.

Akbik, Alan, Larysa Visengeriyeva, Johannes Kirschnick, and Alexander Löser. 2013. “Effective Selectional Restrictions for Unsupervised Relation Extraction.” In *6th International Joint Conference on Natural Language Processing*.

“Drools Documentation.” 2014. Accessed August 13. http://docs.jboss.org/drools/release/6.1.0.Final/drools-docs/html_single/index.html.

Faria, Luis, Alan Akbik, Barbara Sierman, Marcel Ras, Miguel Ferreira, and Jose Carlos Ramalho. 2013. “Automatic Preservation Watch Using Information Extraction on the Web.” In *10th International Conference on Preservation of Digital Objects, iPres*.

“Hadoop - Apache Hadoop 2.4.1.” 2014. Accessed August 13. http://hadoop.apache.org/docs/current/.

Jurik, Bolette Ammitzbøll, Asger Askov Blekinge, Rune Bruun Ferneke-Nielsen, and Per Møldrup-Dalum. 2014. “Bridging the Gap Between Real World Repositories and Scalable Preservation Environments.” In *In Preprint*. Vol. 2014. London.

Maqsud, Umar, Sebastian Arnold, Michael Hülfenhaus, and Alan Akbik. 2014. “Nerdle: Topic-Specific Question Answering Using Wikia Seeds.” In *25th International Conference on Computational Linguistics, COLING*.

May, Peter, and Carl Wilson. 2014. *SCAPE | D2.3 Technical Architecture Report v2*. SCAPE Deliverable D2.3. SCAPE Project. http://www.scape-project.eu/deliverable/d2-3-technical-architecture-report-v2.

Møldrup-Dalum, Per. 2014. “A Weekend With Nanite.” *Open Planets Foundation*. Accessed August 6. http://openplanetsfoundation.org/blogs/2014-05-28-weekend-nanite.

Møldrup-Dalum, Per, Lynn Marwood, Sven Schlarb, Alan Akbik, Ivan Vujic, and Carl Wilson. 2013. *SCAPE | D9.2 Characterisation Technology, Release 2 + Release Report*. SCAPE Deliverable D9.2. SCAPE Project. http://www.scape-project.eu/deliverable/d9-2-characterisation-technology-release-2-release-report.

“Openplanets/policies.” 2014. Accessed August 6. https://github.com/openplanets/policies/.

Palmer, William. 2014. “A Tika to Ride; Characterising Web Content with Nanite.” *Open Planets Foundation*. Accessed August 6. http://www.openplanetsfoundation.org/blogs/2014-03-21-tika-ride-characterising-web-content-nanite.

Raditsch, Markus, Peter May, Asger Askov Blekinge, and Per Møldrup-Dalum. 2012. *SCAPE | D9.1 Characterisation Technology, Release 1 & Release Report*. SCAPE Deliverable D9.1. SCAPE Project. http://www.scape-project.eu/deliverable/d9-1_characterisation_technology_release1__release.

Rötzer, Lukas, and Peter Schmidt. 2013. *Conflict Reduction by Rulebased Postprocessing in C3PO*. Vienna: Vienna University of Technology.

"Scalable Preservation Environments, Annex I - 'Description of Work.'" 2013.

Schlarb, Sven. 2014. "Droid File Format Identification Using Hadoop." *Open Planets Foundation*. Accessed August 8. http://openplanetsfoundation.org/blogs/2013-05-24-droid-file-format-identification-using-hadoop.

Schlarb, Sven, William Palmer, and Leila Medjkoune. 2014. *SCAPE | D15.2 Web Content Executable Workflows for Large-Scale Execution DRAFT*. SCAPE Deliverable D15.2. SCAPE Project. http://www.scape-project.eu/deliverable/d15-2-web-content-executable-workflows-for-large-scale-execution.

Sierman, Barbara. 2014. "SCAPE Policy Framework - SCAPE - Confluence." *SCAPE Wiki*. Accessed August 6. http://wiki.opf-labs.org/display/SP/SCAPE+Policy+Framework.

Sierman, Barbara, Catherine Jones, and Gry Elstrøm. 2014. *SCAPE | D13.2 Catalogue of Preservation Policy Elements*. SCAPE Deliverable D13.2. SCAPE Project. http://www.scape-project.eu/deliverable/d13-2-catalogue-of-preservation-policy-elements.

"ToMaR." 2014. Accessed August 6. http://openplanets.github.io/ToMaR/.

Van der Knijff, Johan, and Carl Wilson. 2011. *Evaluation of Characterisation Tools Part 1: Identification*. SCAPE Internal checkpoint CP066. SCAPE Project.

Vujic, Ivan. 2014a. "SCAPE Azure Platform - SCAPE - Confluence." *SCAPE Wiki*. Accessed August 6. http://wiki.opf-labs.org/display/SP/SCAPE+Azure+Platform.

———. 2014b. "Characterisation and Identification on SCAPE Azure Platform - SCAPE - Confluence." *SCAPE Wiki*. Accessed August 6. http://wiki.opf-labs.org/display/SP/Characterisation+and+Identification+on+SCAPE+Azure+Platform.

"Web Archive FITS Characterisation Using ToMaR at ONB - SCAPE - Confluence." 2014. Accessed August 8. http://wiki.opf-labs.org/display/SP/Web+Archive+FITS+Characterisation+using+ToMaR+at+ONB.

White, Tom. 2012. *Hadoop: The Definitive Guide*. 3rd ed. 1 vols. O'Reilly.