Opening Up the Natural History Heritage for Europeana

COMPONENT REPORT

Project Acronym: OpenUp!

Grant Agreement No: 270890

Project Title: Opening up the Natural History Heritage for Europeana

---

C2.5.0 Availability Checker

Revision: 2a [Final]

Authors:

Simon Kennedy NHM London

Gavin Malarky NHM London

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	x
C	Confidential, only for members of the consortium and the Commission Services	

0 REVISION AND DISTRIBUTION HISTORY AND STATEMENT OF ORIGINALITY

Revision History

Revision	Date	Author	Organisation	Description
V1	2012-05-27	S. Kennedy	NHM	1st Draft
V2	2012-05-31	G. Malarky	NHM	Final draft
V2a	2012-06-06	Coordination Team	BGBM	Minor editing
N.B.: The concept and initial details were discussed at the 4th TMG meeting in Tervuren, December 14th 2011, with representatives from AIT, BGBM, MRAC, NHM, NHMW, RBGK, UH. Subsequently the concept was further discussed and details agreed at the 5th TMG meeting in Paris, March 19th 2012, with representatives from AIT, BGBM, ETI, GBIF, IBSAS, MFN, MRAC, NHM, NHMW, NM, RBGK, UH.				

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Distribution

Recipient	Date	Version	Accepted YES/NO
TMG Coordinator (A. Güntsch, BGBM)	2012-05-31	V2	YES
Work Package Leader (G. Malarky, NHM)	2012-05-31	V2	YES
Project Coordinator (W. Berendsohn, BGBM)	2012-06-08	V2a	YES

TABLE OF CONTENTS

1 PURPOSE

OpenUp! is aggregating and supplying URLs for specimen pages and adequate resolution images for thumbnail creation to Europeana.

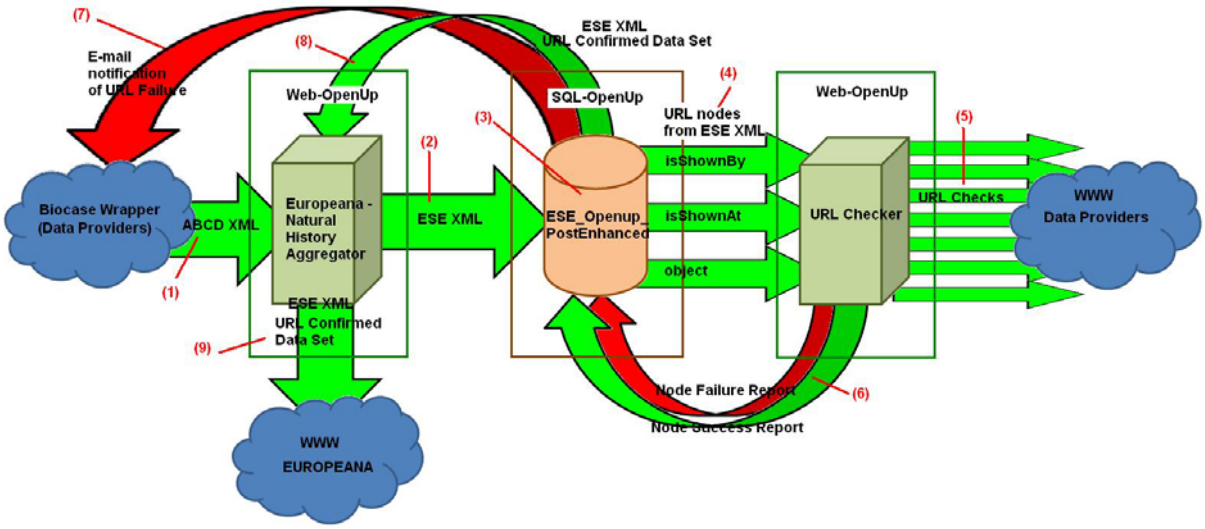
To reduce the likelihood of broken links for OpenUp! records in Europeana we have as part of WP2 implemented an availability checker which will provide a warning flag at metadata record level if any URLs in a record fail. Institute contacts at the corresponding provider will be informed of failures by email. Records which fail on a number of occasions will not be submitted to Europeana.

2 HISTORY

The concept and initial details were discussed at the 4th TMG meeting in Tervuren, December 14th 2011, and subsequently the concept was further discussed and details agreed at the 5th TMG meeting in Paris, March 19th 2012.

3 WORKFLOW

The following figure describes how the availability checker fits with the harvesting and delivery of records to Europeana.



(1) ABCD XML data is harvested by Europeana- Natural History Aggregator from data providers through Biocase wrapper

(2) Pentaho process in aggregator transforms ABCD XML to ESE XML and presents it to ESE_OpenUp_Postenhanced Database

(3) ESE XML is shredded to extract the isShownBy, isShownAt, and Object URL's

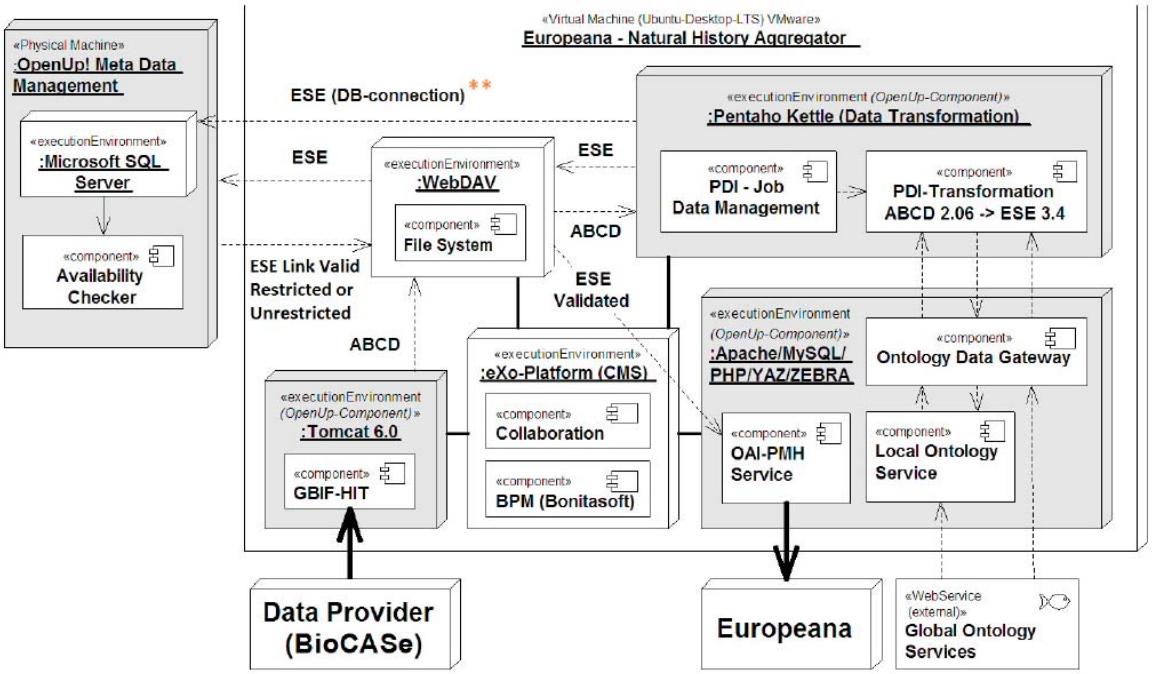
(4) URL's are passed to URL checker

(5) URL checker validates URL against web
(6) URL checker returns succeeded and failed URLs
- Records in PostEnhanced database are updated and failures added to pool to recheck

(7) Content providers of failure URLs are notified by email
(8) ESE XML /URL confirmed data* passed back to aggregator
(9) ESE XML /URL confirmed data* set sent to Europeana

* confirmed data is not just URL validated but either restricted or unrestricted ESE set as requested by provider.

Expanded diagram showing interactions of Europeana - Natural History Aggregator



4 URL CHECKING CODE (CODE WRITTEN BY STEFF WATKINS AT NHM)

A simple script that requests header info for URLs contained in ESE records in the meta-database. The code has been designed to perform multiple simultaneous connections to various differing URLs. The code then processes the list by waiting for a response. On getting a response, the code marks the connection as a success or failure. A "completed" connection is then replaced on the list by the next unused URL from the input list.

```
#!/usr/bin/perl

# url-check (V0.5) - Script to check a large list of URLs for
availability
# Script requests header info for URLs contained in ESE records in the meta-
# database.
# The code has been designed to perform multiple simultaneous connections to
# various differing URLs. The code then processes the list by waiting for a
# response. On getting a response, the code marks the connection as a success
# or failure. A "completed" connection is then replaced on the list by the
# next unused URL from the input list.


# Load the required Perl modules.
use HTTP::Request;
use HTTP::Async;

# As some URLs could take a long time (10 seconds+) to respond, the call
queue
# is to be handled asynchronously/in parallel. That way, a "slow
response"
# will not hold up other calls.

# Make a pool of async call handles
my $async = HTTP::Async->new(timeout=>60,slots=>100);

# Create two files to hold the results of the tests. There are only two
# response states: 'good' responses and 'bad' responses.
open(FO1,">good-urls.txt");
open(FO2,">bad-urls.txt");

# Make up an array of human readable month names for the timestamp
@abbr = qw( Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec );

# Open the list of URLs to be checked.
open(FOO,"urls.txt");

# While there are still URLs in the file to be checked...
while ($line=<FOO>) {
```

```
# Get rid of any cruft at the EOLto tidy it up
chomp($line);

# The input data is in the format "ID,URL". The next lines split them up.
$line =~ /\,/;
$URLid=$`;
$url=$';

# The 'id' here is the 'call ID' and NOT the 'URL id'. This id is
returned by
# the routine that adds requests to the asynchronous queue. The id is
'unique'
# within the scope of the queue and so can be used as the 'key' to other
# arrays that hold info on the various requets.
# Ideally this would be done with an associative array.

# Add a call to the request queue and capture the id of that call
#$id=$async->add( HTTP::Request->new( GET => "$url" ) );
$id=$async->add( HTTP::Request->new( HEAD => "$url" ) );
# Store the URL and supplied id in memory and link them to the freshly-
made
# request
$ReqURL[$id] = $url;
$ReqID[$id] = $URLid;

# Run through the queue unless it is empty
while ( $async->not_empty ) {

# If there is a response to a request then process it.
   if ( my ($response,$id) = $async->next_response ) {

# Trap the status returned by the webserver.
     $status=$response->status_line;

# Make up the timestamp.
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =  localtime(time);
$year = $year+1900;
$Tstamp=$year.'-'.$mon.'-'.$wday.'-'.$hour.':'.$min.':'.$sec;

# The next line is purely for debugging. Uncomment as necessary
#     print("$ReqID[$id] , $ReqURL[$id] , $status\n");

# If the call returns '200' then record it in the "successful" file.
# Should probably add other status codes but these seemed to work in
testing
    if ($status == 200)
    {
     print(FO1 "$ReqID[$id] , $ReqURL[$id] , $Tstamp, $status\n");
    }
    else
# ..else record it in the "not successful" file, complete with status
line
```

```
# so the process can be refined by what responses are received.
    {
     print(FO2 "$ReqID[$id] , $ReqURL[$id] , $Tstamp, $status\n");
    }

# Another debug line  to show response times in a test-run.
#    $old_value = $async->in_progress_count;
#    print("Number of open requests: $old_value \n");
   }
  }
}

close(FO0);
close(FO1);
close(FO2);
```

5 FUTURE WORK

- Investigate whether we can validate that URLs which should return an image do in fact return an image. It is unlikely we can do this for all URLs but might be useful for randon samples from each provider.
- Develop rules for how often we re-check failing URLs before flagging as not valid for Europeana upload.
- Develop email notification to providers of failures to be more helpful.
- Differentiate email rules to be specific to certain cases.
- Integrate results of checks into Central Provider Database so status of providers can include quality of links.