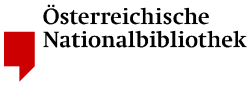
D5.4.1 Europeana Resolution Service

Documentation and final prototype

co-funded by the European Union

The project is co-funded by the European Union, through the ***e*Content*plus*** programme

http://ec.europa.eu/econtentplus



EuropeanaConnect is coordinated by the Austrian National Library

ECP-2008-DILI-528001

EuropeanaConnect

Europeana Resolution Discovery Service

Deliverable number/name	*D 5.4.1*
Dissemination level	
Delivery date	*31/07/2010*
Status	*v.1.0*
Author(s)	*DNB, UW, ONB*

e*Content*plus

This project is funded under the *e*Content*plus* programme,
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

Österreichische Nationalbibliothek

EuropeanaConnect is coordinated by the Austrian National Library

Distribution

Version	Date of sending	Addressee	Role in project
0.1	July 2, 2010	Bernhard Haslhofer, Elaheh Momeni, Georg Petz, Nuno Freire, Kadir Karaca Kocer	Task partners
0.1	July 9 2010	Lars Svensson, Joachim Korb	Project coordinator (DNB), WP5 Leader
0.2	July 15 2010	EuropeanaConnect	EuropeanaConnect Management
0.2	July 15 2010	Jan Molendijk, Martin Gordon	Technical Lead, WP6 lead
1.0	August 18 2010	Max Kaiser	PC

Approval

Version	Date of approval	Name	Role in project
0.3	July 21 2010	Martin Gordon	WP6 lead
0.3	August 8 2010	Jan Molendijk	Technical Lead
1.0	August 20 2010	Max Kaiser	PC

Revisions

Version	Status	Author	Date	Changes
0.1	Draft	Kirubel Legasion	July 8, 2010	Some changes and comments done by the task partners
0.2	Draft	Kirubel Legasion	July 14, 2010	Some changes by Joachim Korb and Lars Svensson
0.3	Draft	Kirubel Legasion	August 18 2010	Changes according to review comments
1.0	Final	VPZ	August 18 2010	Some layout changes

Executive Summary

Task 5.4 in EuropeanaConnect is to build a Resolution Discovery Service for Persistent Identifiers. The Europeana Resolution Discovery Service (ERDS) resolves URN:NBN, DOI, ARK, HANDLE and all HTTP based persistent identifiers. Currently, the ERDS prototype is running on the Europeana Sandbox for testing purposes. [Sandbox]

Persistent Identifiers are very important to enable digital objects to have a stable access point over the Internet. Currently, URLs are used both as an address and as an identifier for referencing the digital objects. If the URL is changed, however, it will no longer be possible to locate the referenced digital object. Hence, the tasks of identifying and accessing the digital objects must be distinguished from each other. In order to distinguish these tasks, an identifier is required that is not permanently bound to a particular address, but which can be resolved at any time to the valid address of the digital object. This problem is solved by using Persistent Identifiers.

Unit/Functional testing and Stress testing have been conducted on the ERDS prototype. All tests passed successfully; the ERDS prototype performs well under normal circumstances.

DNB has led the task of developing the ERDS together with BNP, ONB and UW.

Table of Contents

1. Introduction

The European Resolution Discovery Service (ERDS) is a meta-resolver that interfaces different resolution services in order to allow identification of an object in the World Wide Web to be independent from the object's actual physical location. Implementing resolution services assures persistent accessibility of the digital objects and the ERDS provides this accessibility across the different resolution scenarios for those objects in the Europeana portal that have persistent identifiers.

The ERDS keeps track of namespaces of persistent identifiers and of the institutions that are responsible for those namespaces. With the help of this information, the ERDS can forward resolving requests to the right local resolver and present the returned link.

The following diagram visualizes this principle:

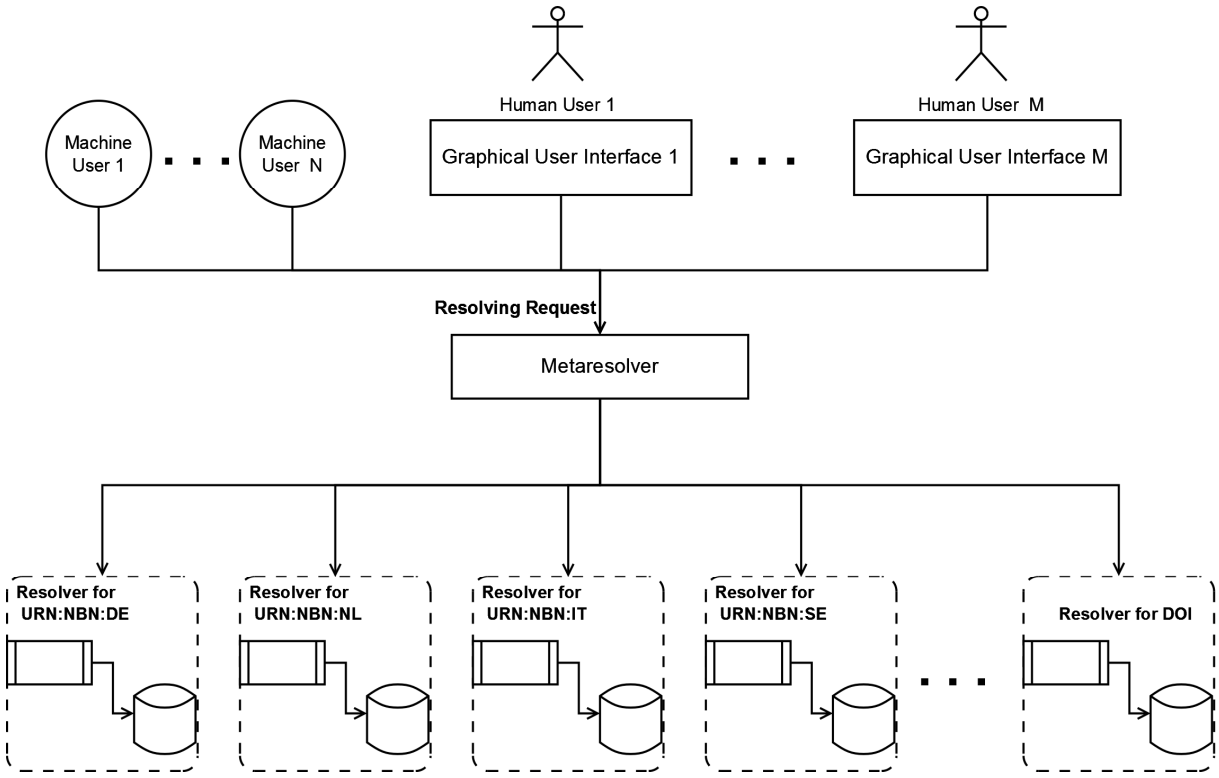


Fig.1: Basic functionality principle of the ERDS

The ERDS is currently running in a Europeana Sandbox. Please refer to [Sandbox] to have access to the Europeana Sandbox where the ERDS is hosted.

The internal work flow between T5.4 partners and the EDL is documented in a EuropeanaLabs ticket. [EuropeanaLabs #931]

The ERDS is a back-end service and users don't interact with it directly. But for the sake of testing purpose in the Europeana Sandbox, a simple graphic user interface (GUI) has been implemented as shown in Fig.2 below. Hence, in the Sandbox, the ERDS takes persistent identifiers as input and returns the resolved URL to redirect the user to the actual location of the digital object.

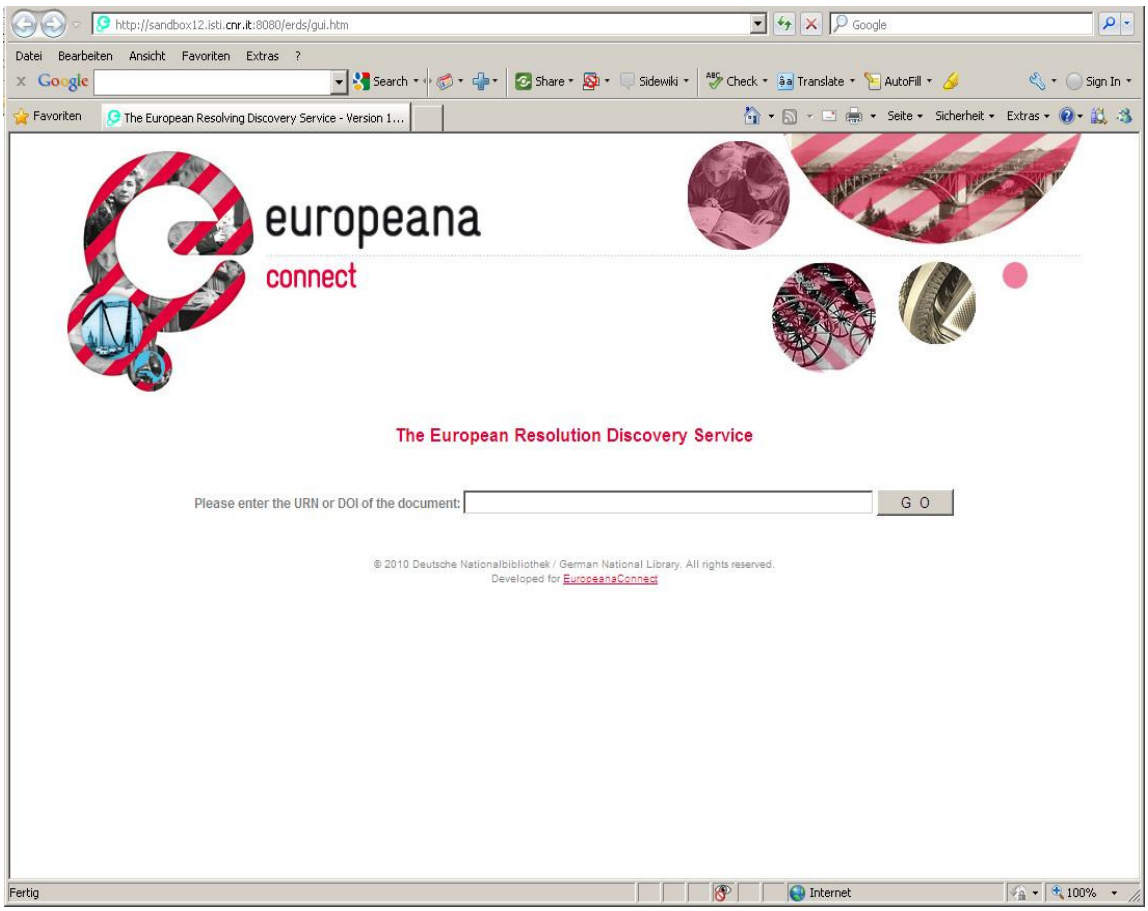


Fig.2: Screenshot of the ERDS on the Europeana Sandbox

2. Functional requirements to develop the ERDS Service

The ERDS service:

1. must resolve all persistent identifiers that are in format urn:nbn:[1]. Resolving means to return the URL with the highest priority pointing to the desired digital object or its identical copies (see section Remote Interfaces)
2. must redirect all persistent identifiers that are in format DOI to the official resolver of The International DOI Foundation[2]
3. must redirect all PURL[3], ARK[4], OpenURL[5] and Handle[6] requests
4. must be easily extensible for other types of persistent identifier types
5. must return clear, well defined error codes/messages in case it can not resolve the request
6. may have a GUI, but it is not required
7. doesn't host data by itself. It rather traffics the data provided by national resolvers
8. caching at the meta-resolver level will not be implemented
9. must handle errors thrown by the local resolvers
10. must offer an easy way to register additional resolvers

2.1 Obligations of the ERDS Service

1. The ERDS service is responsible neither for the correctness of the links that are delivered, nor the authenticity or validity of the linked digital item. This will be the responsibility of the local resolvers.
2. The ERDS service is not responsible for the content it resolves. It simply resolves an identifier to link to the digital object, irrespective of the content of the digital object
3. The ERDS service manages neither relationships between digital entities nor part-of /whole /fragment-of relations inside the same object.
4. The ERDS service does not guarantee that the returned URL is free to access. There may still be legal, commercial or other issues that restrict access to an object to a certain user community

[1] RFC 2141 and RFC 3188
[2] http://doi.org/
[3] http://purl.org/
[4] The ARK Identifier Scheme
[5] OpenURL – ANSI/NISO Z 39.88
[6] The Handle System

2.2 ERDS Service Process Flow-Diagram

1. The user sends a PI-request to the ERDS service (meta-resolver).
2. Upon receiving the PI-request, the ERDS service will check the look-up table which resolver is responsible to resolve this specific request.
3. The ERDS service then forwards the request to the local resolver. The local resolver will return (HTTP 303) a valid URL link that refers to the digital object back to the ERDS service.
4. The ERDS service will redirect (HTTP 303) the valid URL link to the user.
5. The user can click on the provided valid URL to access the digital object directly from the archival entity.

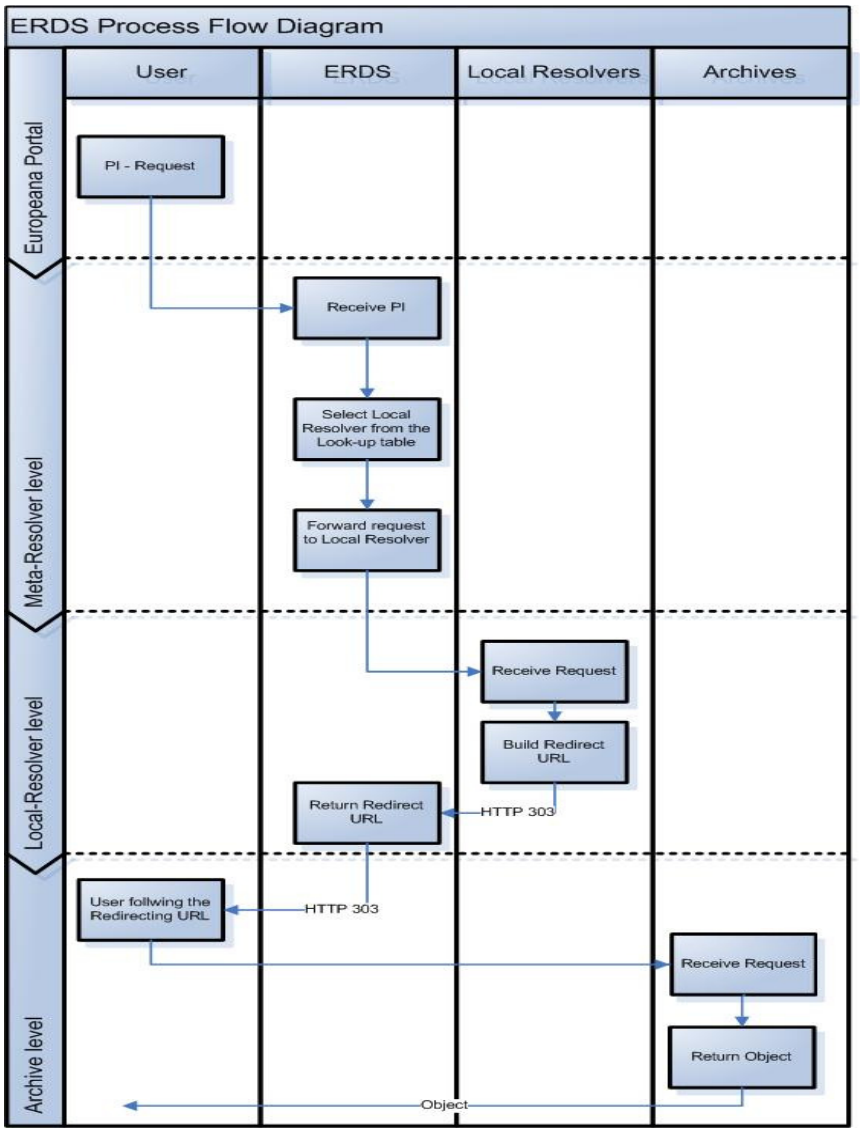


Fig.3: ERDS Process Flow Diagram

3. ERDS Architecture and Code-Descriptions

ERDS is designed as a Web application. It is fully implemented in Java and must be deployed on a Tomcat Servlet container. It is developed and tested on Java 6.0.20 and Tomcat 6.0.26 but should run on any other compatible version.

3.1 ERDS Architecture

ERDS uses Dependency Injection and ModelAndView Patterns built using the Spring Framework. [Spring Framework]

The internal structure of the software can be seen in Class Diagram below:

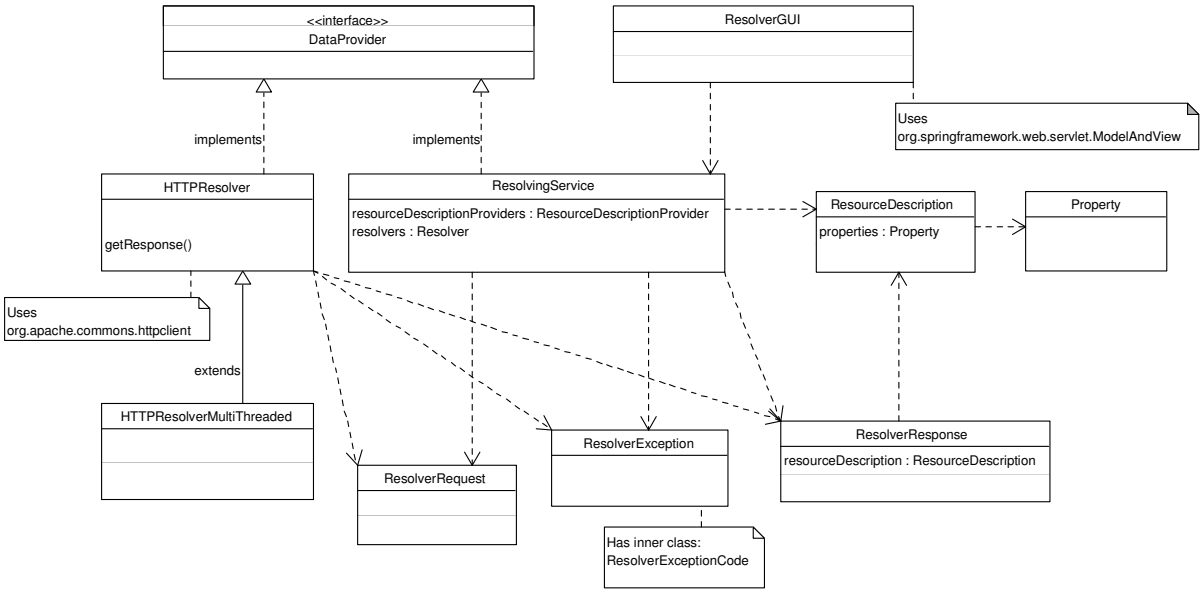


Fig.4: Internal Structure and Architecture of ERDS

3.2 ERDS Code-Description

DataProvider: refers to an abstract expression for all classes that provide some kind of data to the user or a partner system. All implementing classes shall resolve requests for a well defined pattern and return an implementation of the class ResolverResponse.

HTTPResolver: Simple Resolver that queries a remote server via HTTP. It fetches the URL by reading the header of the redirect-response of the remote resolver. HTTPClient from Apache Project is used for this functionality in a single threaded manner.

HTTPResolverMultiThreaded: A Resolver that queries a remote server via HTTP. Like HTTPResolver (which it extends), it fetches the URL by reading the header of the redirect-response of the remote resolver. The difference is that HTTPResolverMultiThreaded supports multiple simultaneous connections and request retries on error.

Property: A simple helper class to manage properties.

ResolverException: Exception type representing an error state during the process of resolving a persistent identifier.

ResolverGUI: Spring controller to provide web access to the Resolvers. A simple graphical user interface is also provided for manual testing.

ResolverRequest: Class representing a request for resolving a persistent identifier.

ResolverResponse: Class representing a response to a resolving request.

ResolvingService: Main service for resolving the URL for a given persistent identifier.
This class manages a list of known resolvers and iterates them until the given identifier matches a registered pattern. A ResolverException will be thrown if none of the registered resolvers matches the pattern or can resolve the id.

ResourceDescription: Simple utility class representing the description of a resource.
More information about all interfaces, classes, methods and parameters can be found in JavaDoc.

3.3 The ERDS and HTTP Client Configuration

I. All the configuration of ERDS can be changed directly in the appropriate XML files:
   - web.xml: Servlet, filters, mappings, time-outs …
   - applicationContext.xml: Application context definition
   - velocity-toolbox.xml: Configuration of the used Apache Velocity Engine. For more information, please refer to [Velocity].
   - MetaResolver-servlet.xml: All servlet specific configurations, especially controller mappings
   - metaResolver.xml: List of all registered local resolvers with their descriptions, supported namespaces and patterns to match.

II. Default ERDS logs everything – log level DEBUG!
The Log4J configuration can be changed in appropriate Log4J configuration file. For more information, please refer to [Log4J]

III. The performance of the ERDS relies to a great degree on the performance of its HTTP communication with the other resolvers. The ERDS deploys an HTTP client implementation based on the HTTP Components project from the Apache Software Foundation – the Jakarta Commons HTTP Client. By default the Jakarta HTTP Client is configured to provide maximum reliability and standards compliance rather than raw performance. So a few configuration options and optimization techniques where implemented to improve the performance of HTTP Client. The following points describe these optimizations:

- ***Reuse of the HttpClient instance across requests to the ERDS:*** This is a general recommendation from the HTTP Client documentation. An application with much HTTP communication, such as the ERDS, should have a single instance of HttpClient.
- ***Concurrent execution of HTTP methods:*** The ERDS application logic allows for execution of multiple HTTP requests concurrently (e.g. multiple requests against various sites, or multiple requests representing different user identities), the use of a dedicated thread per HTTP session can result in a significant performance gain.

  To allow this, the MultiThreadedHttpConnectionManager was used to make the HttpClient fully thread-safe. Each thread of execution has a local instance of its Http-Method. The same HttpClient instance and connection manager are shared among all threads for maximum efficiency.
- ***Automatic retries on failure:*** Occasional failures in HTTP requests happen, so the ERDS should cope with them when communicating with the other resolvers. For this reason an HttpRequestRetryHandler was implemented that allows the HTTP client to retry the resolution requests in case of temporary failure.

4. Tests on the ERDS

After the ERDS prototype was developed, different software testing techniques were implemented in order to test quality and performance of the ERDS software.

The following assumptions were taken with respect to testing the ERDS:

- Integration and System Tests are not part of T5.4. These types of tests will be handled by other testing tasks of EuropeanaConnect in cooperation with the Europeana Foundation.
- In the production system, there is no separate GUI for the ERDS. Rather, the ERDS runs as a back-end service. Hence, it is not necessary to apply Acceptance Tests for the ERDS.
- Apart from the intensive expert tests that were applied in the different cycles of the ERDS software development process, Unit/Functional Tests and Stress/Robustness Tests were very important before deploying the software. Hence, T5.4 task partners at the UW have applied Stress/Functional testing techniques and partners at the ONB were responsible to apply the Unit/Functional testing on the ERDS software. The results of these tests is summarized as follows.

4.1 Stress Test on ERDS

The overall objective of this test is to determine the robustness of ERDS by testing beyond the limits of normal operation. The test puts a greater emphasis on robustness, availability, and error handling under a heavy load than on what would be considered correct behaviour under normal circumstances.

4.1.1 Test Scope and Environment

The test has been performed on the first prototype of ERDS, which is online [Sandbox]. All tests were performed on a Mac Book Pro with 2.4 GHz and 2 GB memory. The operating system was Mac OS X, running Java Version 1.6.0_13 with the following heap size settings:- Xms512m - Xmx1024m.

4.1.2 Test Preparation and Execution

A test case was created with:

- 50 different Persistent Identifiers (PIs), retrieved from random search on the web by using Google (Details shown in Appendix 1)
- 20 threads, which were running in parallel. Each individual thread stands for a user issuing a request.

The test was executed using Apache JMeter based on two phases [JMeter]:

1. The test was run locally by sending 20,000 HTTP requests to the service, which performed perfectly without any error.
2. Due to the positive result of the local test the test was run remotely using the sandbox link [Sandbox] by sending 10,000 HTTP requests to the service, which also performed successfully without any error.

4.1.3 Stress Test Results

The results of the tests are summarized as shown in the following three elaborative figures (Fig.5, Fig.6 and Fig.7).

Fig.5 shows a screenshot of the JMeter test monitor. The horizontal vector shows the HTTP request count and the vertical vector shows the time. The black dots represent upcoming HTTP requests, which were running in parallel. The red line shows the deviation[7], which is the variability of the requests. Among 10,070 samples 8,986 samples were different. The variability is due to the various combinations of different threads and PIs. The blue line shows the average response time of each PI, which is almost around 1,018 ms within the test. The green line indicates the throughput[8], which is calculated as requests/minute. It remains nearly constant over time. The purple line is the median, dividing the samples into two equal halves. 50% of the samples are smaller than the median, 50% are larger. The median also remains constant over time.

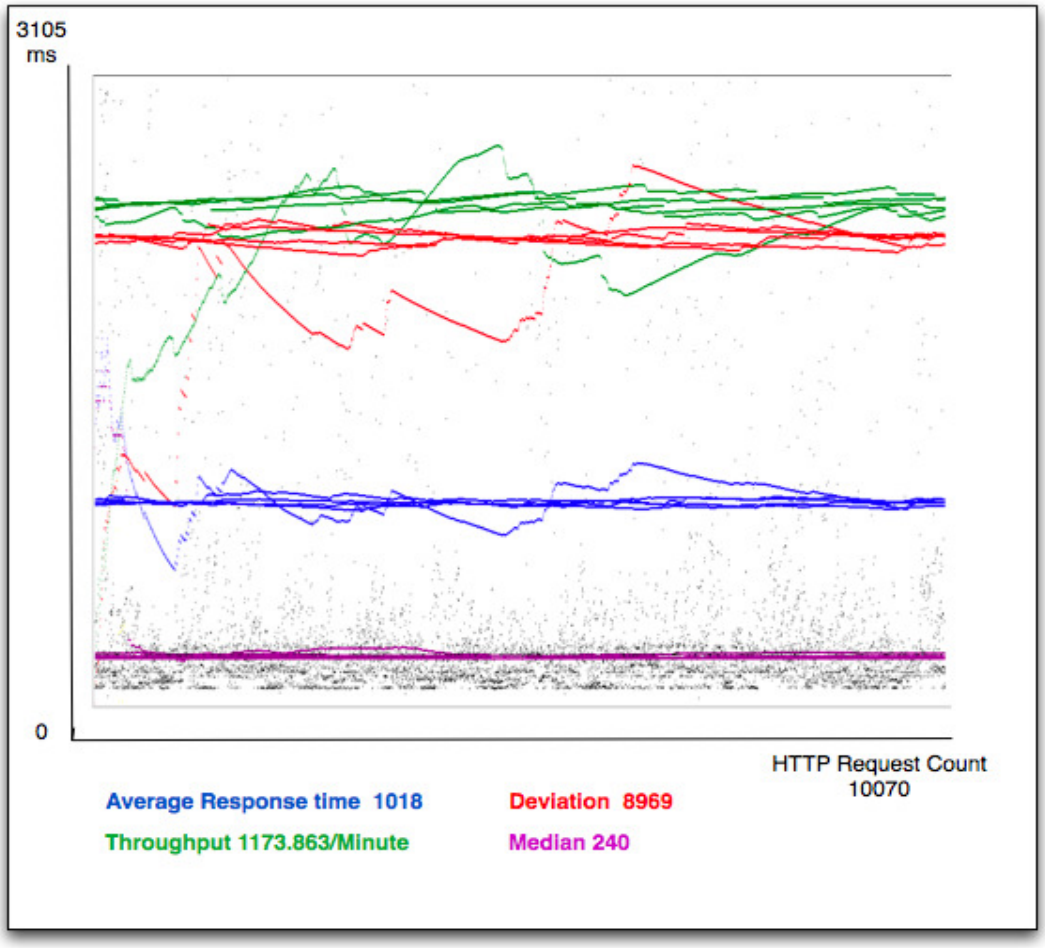


Fig.5: Screenshot of the JMeter test monitor

[7] **Deviation** is a measure of the variability of a data set. This is a standard statistical measure.

[8] **Throughput** is calculated as requests/unit of time. The time is calculated from the start of the first sample to the end of the last sample. This includes any intervals between samples, as it is supposed to represent the load on the server. The formula is: Throughput = (number of requests) / (total time).

Fig.6 shows the average response time of requests for different PIs. Each node on the vertical vector represents a PI and the horizontal vector shows the time. The average response time for the majority of PIs is less than 2 seconds. Longer response times were retrieved only for PIs in the domain of the Dutch PI resolution service. This, however, is not

under the control of the ERDS as it forwards requests to external PI resolvers and depends on their response times.

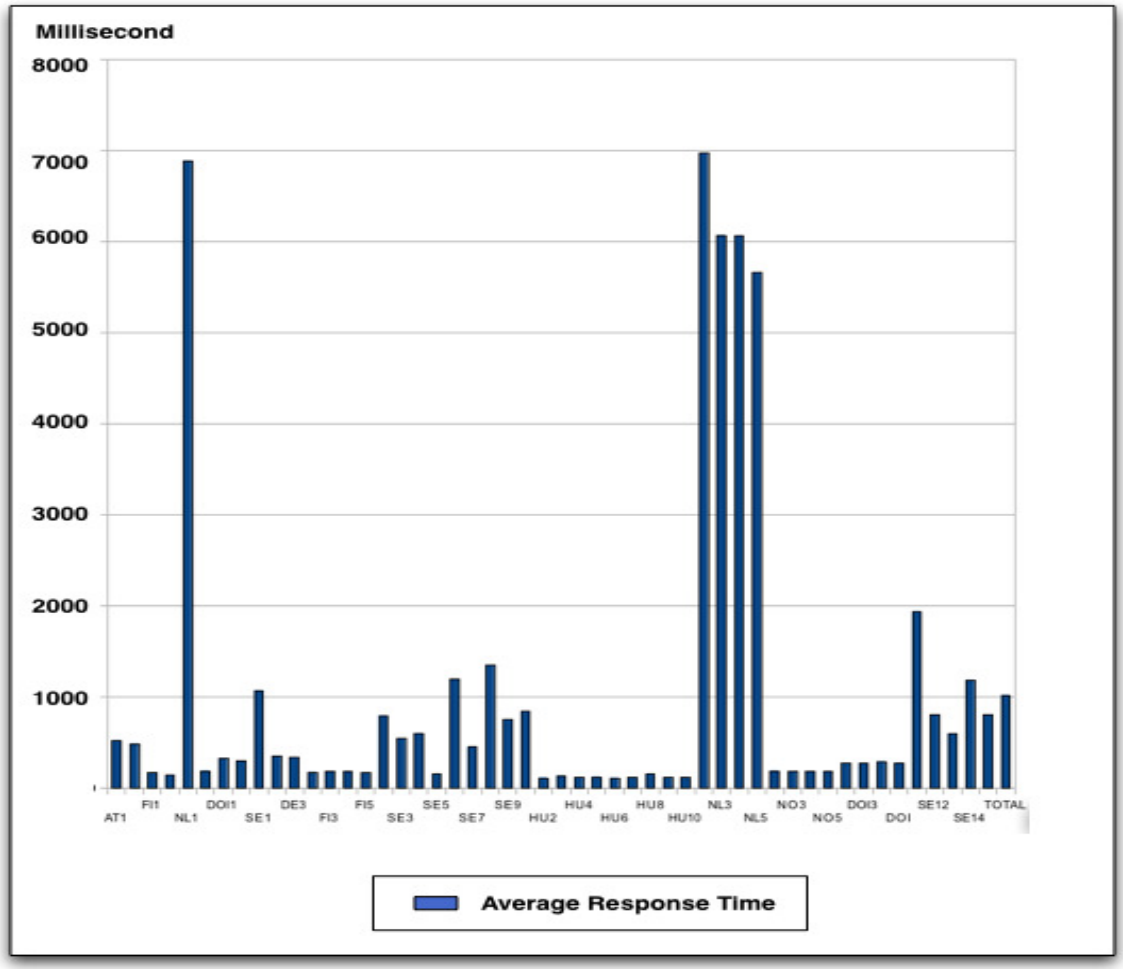


Fig.6: Average response time of requests for different PIs

Fig.7 shows minimum and maximum response time of requests for different PIs. The maximum response time for the majority of PIs is less than 15 seconds.

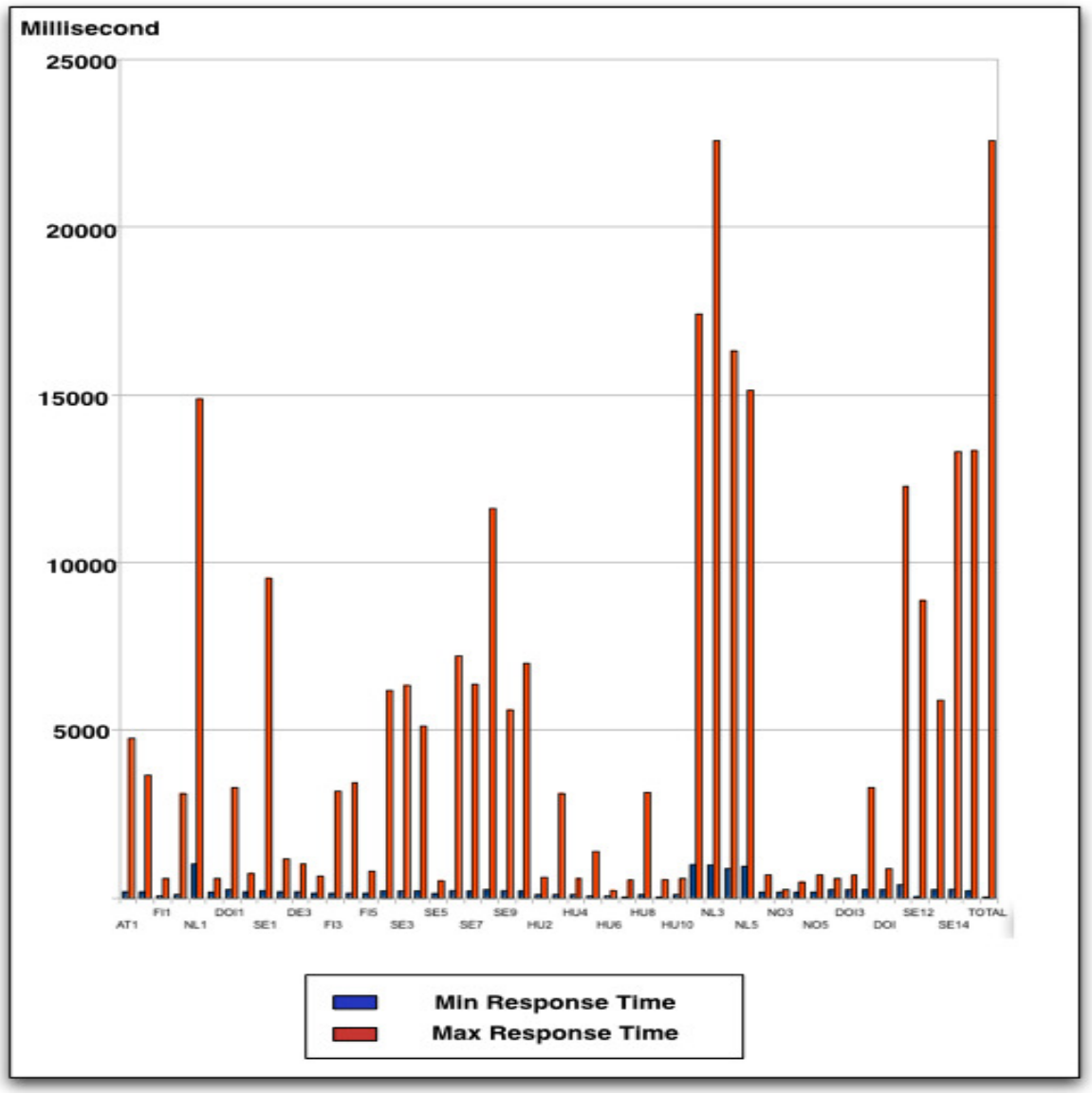


Fig.7: Minimum and maximum response time of requests for different PIs

Table 2 in Appendix 2 shows the number of requests, average reported error, average reported bandwidth, and average reported bytes for different PIs.

4.1.4 Conclusion

The stress test on ERDS with 50 different Persistent Identifiers, as shown in Appendix 1, and 20 threads (totally 10,000 HTTP requests to the service) has been performed successfully. Under a heavy load, the ERDS remained constant over time and the average response time for the majority of PIs was less than 1,018ms.

4.2 Unit/Functional Test on ERDS java classes

ONB has carried out Unit/Functional testing on the ERDS. The classes HTTPResolverMultiThreadedTest.java, HTTPResolverTest.java and ResolvingServiceTest.java were tested with Junit [Junit] with 96.6% statement coverage (measured by CodeCover [CodeCover]).

The following figure, Fig.8 shows the statement coverage of all Junit tests.

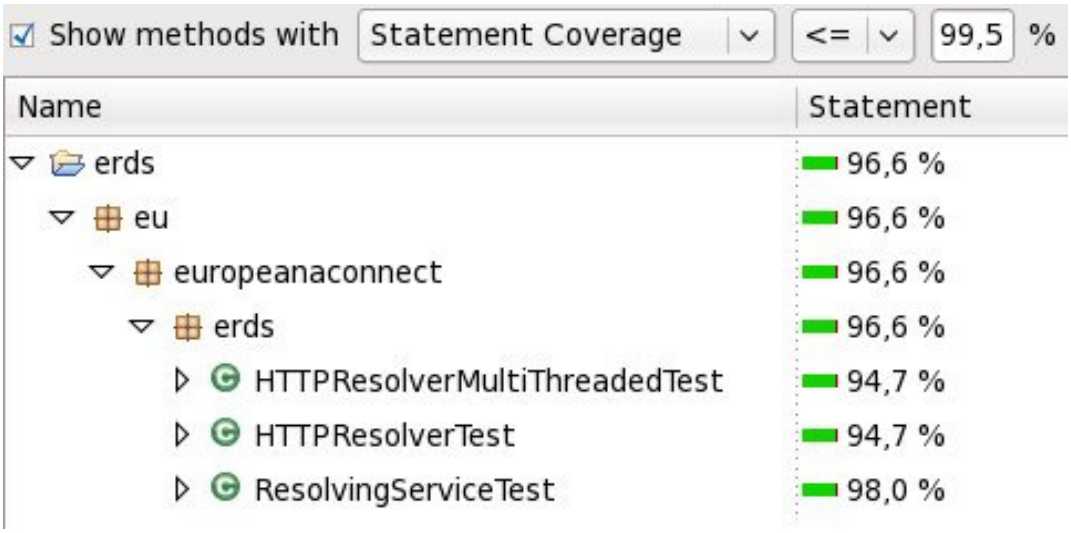


Fig.8: Statement Coverage

HTTPResovlerMultiThreadedTest.java and HTTPResolverTest.java test malformed URNs and resolves **urn:nbn:de**:gbv:089-3321752945 and **doi**:10.1000/182.

ResolvingServiceTest.java resolves **urn:nbn:de**:gbv:089-3321752945, **urn:nbn:at**:0001-03582, **urn:nbn:ch**:bel-9039, **urn:nbn:se**:uu:diva-3475, **urn:nbn:nl**:ui:12-85062 and **urn:nbn:no**-3132.

As a conclusion, all the three tests passed successfully.

5. Conclusion

The ERDS resolves URN:NBN, DOI, ARK, HANDLE and all HTTP based persistent identifiers. The system takes PI requests as an input and returns a valid link to redirect the user to the actual physical location of the requested digital object. This service is intended to run as a back-end service for Europeana.

Users do not necessarily have direct interaction with the ERDS. But upon requesting access to a digital object through the Europeana portal, the ERDS runs behind the scenes and ascertains that the requested object is permanently accessible in the Europeana portal.

The ERDS handles error exceptions in case of failure occurrences from third parties that are providing resolution services such as DOI or ARK.

Unit/Functionality testing has been implemented on the ERDS to make sure of its performance quality. Moreover, in order to determine to what extent the ERDS can handle requests beyond normal circumstances, a robustness/stress test has been implemented. In both test cases, the ERDS has successfully passed the testing criteria.

References

[CodeCover]	Apache JMeter Retrieved on June 15, 2010: http://jakarta.apache.org/jmeter
[EuropeanaLabs #931]	EuropeanaLabs ticket #931 Resolution Service Retrieved on June 30, 2010: https://europeanalabs.eu/ticket/931
[JMeter]	Apache JMeter Retrieved on June 15, 2010: http://jakarta.apache.org/jmeter
[JUnit]	Junit Retrieved on July 07, 2010: http://junit.sourceforge.net/
[Log4J]	Apache Log4J Retrieved on June 30, 2010: http://logging.apache.org/log4j/
[Sandbox]	Europeana Sandbox Retrieved on June 25, 2010: http://sandbox12.isti.cnr.it:8080/erds
[Spring Framework]	Spring Framework Retrieved on June 30, 2010: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/
[Velocity]	Apache Velocity Retrieved on June 28, 2010: http://velocity.apache.org/

Appendix 1

The following table, Table 1, shows details of PIs, which are chosen to carryout stress testing.

Table 1: PIs selected for testing purposes

Persistent Identifier	The name used in report
doi:10.1038/35057062	DOI1
doi:10.1594/WDCC/CCSRNIES_SRES_B2	DOI2
doi:10.1594/PANGAEA.587840	DOI3
doi:10.2312/EGPGV/EGPGV06/027-034	DOI4
doi:10.2314/CERN-THESIS-2007-001	DOI5
urn:nbn:at-123:anl-000001512	AT1
urn:nbn:ch:bel-9478	CH1
urn:nbn:de:swb:90-175200	DE1
urn:nbn:de:gbv:089-3321752945	DE1
urn:nbn:de:bsz:93-opus-59	DE1
URN:NBN:fi:jyu-20094141421	FI1
URN:NBN:fi:jyu-20094141415	FI2
URN:NBN:fi:jyu-20094141416	FI3
URN:NBN:fi:jyu-20094141417	FI4
URN:NBN:fi:jyu-20094141418	FI5
urn:nbn:se:vxu:diva-5859	SE1
urn:nbn:se:vxu:diva-5855	SE2
urn:nbn:se:vxu:diva-5856	SE3
urn:nbn:se:vxu:diva-5857	SE4
urn:nbn:se:vxu:diva-5858	SE5
urn:nbn:se:uu:diva-3475	SE6
urn:nbn:se:uu:diva-3470	SE7
urn:nbn:se:uu:diva-3471	SE8
urn:nbn:se:uu:diva-3472	SE9
urn:nbn:se:uu:diva-3473	SE10
urn:nbn:hu-3952	HU1
urn:nbn:hu-3946	HU2

Persistent Identifier	The name used in report
urn:nbn:hu-3947	HU3
urn:nbn:hu-3948	HU4
urn:nbn:hu-3949	HU5
urn:nbn:nl:ui:12-85062	NL1
urn:nbn:nl:ui:12-85054	NL2
urn:nbn:nl:ui:12-85055	NL3
urn:nbn:nl:ui:12-85057	NL4
urn:nbn:nl:ui:12-85058	NL5
urn:nbn:no-3132	NO1
urn:nbn:no-3131	NO2
urn:nbn:no-3133	NO3
urn:nbn:no-3134	NO4
urn:nbn:no-3135	NO5
urn:nbn:se:liu:diva-17237	SE11
urn:nbn:se:liu:diva-16236	SE12
urn:nbn:se:liu:diva-17236	SE13
durn:nbn:se:liu:diva-17231	SE14
urn:nbn:se:liu:diva-17235	SE15
urn:nbn:hu-3010	HU6
urn:nbn:hu-3008	HU7
urn:nbn:hu-3007	HU8
urn:nbn:hu-3006	HU9
urn:nbn:hu-3005	HU10

Appendix 2

The following table, Table 2, shows the number of requests, average reported error, average reported bandwidth, and average reported bytes for different PIs.

Table 2: Testing data request and response on the ERDS

sampler_label	request count	average_report_error%	average_report_bytes	average_report_bandwidth
AT1	210	0.0	0.4226500155977982	0.04044892727400803
DE1	210	0.0	0.4235467807419329	0.02440357428102934
FI1	210	0.0	0.42422871179921456	0.03964518318599803
HU1	210	0.0	0.42425270915658564	0.013672206447429029
NL1	210	0.0	0.4182750337607706	0.03553703900115922
NO1	210	0.0	0.41978171350897536	0.018447438581937394
DOI1	210	0.0	0.4176735582803783	0.020802101047167277
CH1	210	0.0	0.41772174559950037	0.024883814923407736
SE1	210	0.0	0.4155667340808258	0.025567093991300804
DE2	210	0.0	0.4151731964184392	0.025948324776152453
DE3	210	0.0	0.41494267863282314	0.021881742818527783
FI2	210	0.0	0.41498777762188294	0.03890510415205152
FI3	210	0.0	0.41493775933609955	0.03890041493775934
FI4	210	0.0	0.4149230021457447	0.038899031451163565
FI5	210	0.0	0.4146174364401342	0.03887038466626258
SE2	210	0.009523809523809525	0.41442760640428794	0.03368380629061637
SE3	208	0.02403846153846154	0.41071492040423824	0.04574726173598139
SE4	203	0.0	0.4035159557762415	0.024825688685452357
SE5	203	0.0	0.40360581033310405	0.0126126815729095
SE6	203	0.0	0.40325624451233805	0.024415905429457967
SE7	203	0.0	0.40326105094200193	0.02441619644375402
SE8	203	0.0	0.4030920936365062	0.02440596660689784
SE9	203	0.0	0.40311690787487886	0.02440746903148 6804
SE10	203	0.0	0.40312091045751247	0.0244077113753572
HU2	203	0.0	0.4032506371558713	0.012995381861468508
HU3	203	0.0	0.4029880671858529	0.012986920133919088
HU4	203	0.0	0.4029776674937965	0.012986584987593051
HU5	203	0.004926108374384 2365	0.40290648401572726	0.01734146163258501
HU6	202	0.0	0.4009997200942548	0.012920903921856668
HU7	202	0.004950495049506	0.401083322081821	0.0209996344085808
HU8	201	0.0	0.39919287569809103	0.01286461415823926
HU9	201	0.004975124378109453	0.3992753450454101	0.0182019616884381
HU10	200	0.0	0.40520116211693297	0.012662536316154155
NL2	200	0.0	0.4006739335562416	0.034041633026751
NL3	200	0.01	0.40013764735068863	0.042202017494017946
NL4	198	0.015151515151515152	0.39741320134397917	0.04648953553837446
NL5	195	0.02564102564102564	0.3931515024435878	0.05426506690128066
NO2	190	0.0	0.40617992060251445	0.017849703542102687
NO3	190	0.0	0.4061695009459474	0.017849245647038704
NO4	190	0.0	0.4061668961153343	0.0178491311769434
NO5	190	0.0	0.40616342305982145	0.017848978552433562
DOI2	190	0.0	0.40606793730311047	0.028155101121602386
DOI3	190	0.0	0.40606880515322685	0.018637923674025062
DOI4	190	0.0	0.4057470871630164	0.04160492592980149
SE11	190	0.0	0.4032223835111756	0.024807627110550845
SE12	190	0.0	0.40309748594462713	0.024735715895831123
SE13	190	0.0	0.4031907240668787	0.02480567931270836
SE14	190	0.0	0.4030094198149126	0.02473031179672205

Appendix 3

Acronyms

ARK:	Archival Resources Key
DOI	Digital Object Identifier
DoW	Description of Work – Annex1, ECP 528001
ERDS	European Resolution Discovery Service
NBN	National Bibliography Number
PI	Persistent Identifier
PURL	Persistent Uniform Resource Locator
URL	Uniform Resource Locator
URN	Uniform Resource Name
DNB	German National Library (Deutsche Nationalbibliothek)
ONB	Austrian National Library (Österreichische Nationalbibliothek)
UW	University of Vienna (Universität Wien)