Ensuring the quality and interoperability of open cultural digital content: System architecture and scalability

Haris Georgiadis, Vangelis Banos, Ioanna-Ourania Stathopoulou, Panagiotis Stathopoulos, Nikos Houssos, Evi Sachini
National Documentation Centre / National Hellenic Research Foundation
Athens, Greece
{hgeorgiadis, vbanos, iostath, pstath, nhoussos, esachin}@ekt.gr

***Abstract*—We present an Open Cultural Digital Content Infrastructure, a platform providing a coherent suite of loosely-coupled services that aim to promote metadata quality in repositories and facilitate metadata data and digital content reuse. The key functions of the infrastructure are the aggregation of metadata and digital files and the automatic validation of metadata records and digital material for compliance with desired quality specifications. The system that has recently moved to production, is currently being employed to ensure the quality standards of the output of more than 70 projects that support Greek cultural heritage organisations and are funded by the European Union structural funds. These projects are expected to produce more than 1.5 million digitised and born-digital items accompanied with detailed metadata. The validation is based on a set of quality and interoperability specifications that have been developed for the purpose. In this paper we emphasize on Validator and Aggregator components and present experimental results of their scalability**

Keywords — Metadata aggregation, Metadata quality, Metadata validation, Digital content aggregration, Digital content aggregration validation, Cultural Heritage Infrastructures, OAI-PMH, Interoperability guidelines, Metadata harvesting.

I. INTRODUCTION

The issue of appropriate digitization, documentation and preservation of cultural heritage has been widely recognised for its significance, resulting in a great number of large-scale efforts worldwide. A key issue in achieving the appropriate return from the corresponding investments is ensuring both the quality of the output at the level of both the metadata records and the digital files and their appropriate safe-keeping, dissemination and preservation.

In the past, issues that are hampering the reuse and added value of the documented and digitised cultural heritage items have been observed such as inadequate and non-standards compliant documentation (e.g. use of custom data models instead of established international schemata), poor quality in digitisation and relevant processing (e.g. low image resolution, omission of Optical Character Recognition in scanned texts), non-availability of standard system interfaces for interoperability (lack of OAI-PMH support), failure to secure appropriate safe-keeping of digital files and interruptions (sometimes permanent) in the operation of web application / repositories for the wide dissemination of the material.

To avoid these phenomena in currently running digital cultural heritage funded projects in Greece, a scheme has been created for the development of an infrastructure to aggregate centrally both metadata records and digital files produced in the frame of these projects and automatically validate their conformance with interoperability and quality specifications. A set of such specifications has been developed at the initial stage of the funding programme [1].

The infrastructure that has been built to support this effort is presented in this contribution. It contains a Metadata Aggregator that operates at a national level and provides with a series of added-value services upon them, such as a Search Engine and centralised disposal of content as Linked Data, and a Content Validator that validates the registered repositories against interoperability requirement and the provided content, including both metadata and digital files, against a large and extendible pool of specifications. The Validator is to be used both by the Aggregator personnel to ensure the quality of the metadata and digital files that are to be ingested and published by the Aggregator, and by the content managers and project contractors of the repositories in order to validate their content and to take the necessary steps towards conforming to the specifications. The Validation comprises two components, a front-end and a back-end. A shared, autonomous harvester component supports the operations of both the aggregator and the validator. Communication among the components is performed via REST APIs. The architecture of the infrastructure is depicted in Figure 1.

In this paper, we present the workflow of the infrastructure and emphasize in the Validator and Aggregator components. Specifically, we analyse the architecture of these components in Section 3, we present the compare the infrastructure workflow on Section 4 and display experimental results of the Aggregator and Harvester scalability in Section 5.

II. RELATED WORK

Several repository validator systems are in production operation since a number of years, such as the OpenAIRE validator[1] based on the OpenAIRE guidelines for repositories [2], the ARIADNE validator[2] of learning object repositories [3], the OAI-PMH validator[3] and validator systems produced in the context of Europeana[4] such as the VAMP semantic validation Service for MPEG-7 profile descriptions [5]. Some key features and differentiations of our solution are summarized in the following:

- We validate not only metadata records but also digital files and in addition we check the correct mapping of digital files with the corresponding metadata records.

[1] http://www.openaire.eu:8380/dnet-validator-openaire/

[2] http://ariadne.cs.kuleuven.be/validationService/validateMetadata.jsp

[3] http://validator.oaipmh.com/

[4] http://pro.europeana.eu/web/guest/thoughtlab/improving-metadata-quality

- Certain existing systems limit themselves to only syntactic validation using technologies such as XML Schema and the schematron assertion language [4], while we exercise also semantic validation (as does VAMP [5] and to some extent also OpenAIRE [2]).
- Semantic validation is provided with a great degree of flexibility. For instance, we check whether metadata values of key attributes (e.g. locations, subjects, languages, time periods) belong to formally defined (e.g. with SKOS) controlled vocabularies – the vocabularies can be dynamically configured in the validation rules and do not need to be known a priori to the system.
- Combinations of validation rules are expressed using a domain specific language which is designed to be usable by non-programmers.
- Validation administrative procedures (e.g. connection with repository owners, checking for compliance using rules of a specific funding programme mandate, reporting) are decoupled with the actual validation logic and implemented at a separate component (validator front-end).

On the other hand numerous harvester and aggregator systems have been developed and are in production operation, each attempting to meet specific business needs and tailored to specific domains. The COnnecting REpositories (CORE) aggregator[5] aims to facilitate the access and navigation across relevant scientific papers stored in Open Access UK repositories. Openarchives.gr[6] is the largest portal providing a single point of access to Greek scientific and cultural digital content. National Science Digital Library (NSDL)[7] implemented a digital library based on metadata aggregation of educational content using Dublin Core and OAI-PMH. REPOX[8] is a framework which comprises several channels to import data from data providers and provides services to transform data among schemas and to expose the results to the exterior. It is often used as a harvester component in aggregation infrastructures. European Digital Library (Europeana)[9] has been established through the aggregation of heterogeneous cultural content from multiple content providers, which needs to be delivered reliably and consistently, using a commonly agreed metadata schema. The aggregation/ingestion infrastructure of Europeana is a complicated ecosystem strictly tailored to its business processes that consists of many components including: a metadata harvester (REPOX), a mapping tool (MINT[10]), a CRM system, a Metadata Storage and Indexing system, namely Colelib, and an ingestion framework that orchestrates all these components together.

The aggregator of our infrastructure stores metadata internally in the EDM [6] metadata format, which is the latest recommendation of Europeana for structuring cultural metadata. EDM is RDF-based providing with an embedded contextualization mechanism. The Aggregator uses the Europeanan Corlib as an efficient EDM Storage and Indexing backend - thought without being strictly tied on it – which, being integrated with Apache Solr[11], inherently, features efficient distributed full-text search on ingested EDM metadata records. Unlike most aggregator systems which support only flat metadata formats, the Aggregator component of our infrastructure, due to storing metadata internally as EDM, natively supports sophisticated contextual-based searching and content disposal as Linked Data. The autonomous Harvester component of our infrastructure that supports the operations of both the Aggregator and the Validator, unlike other harvester systems, harvests and stores not only metadata but also digital files.

III. Infrastructure Architecture, Design and Implementation

A. Metadata and Digital File Harvester

An autonomous harvester system is built to harvest and store both metadata and digital files. In order to improve the efficiency and the throughput of the harvester, the execution is implemented in a pipelined workflow scheduling [7] which adopts the pull data workflow model, needing no materialization of intermediate results. The OAI-PMH XML responses are parsed on-the-fly using the Streaming API for XML and as soon as the parsing is finished, they are being forwarded to dependent tasks in the workflow which are being executed simultaneously. Digital files are harvested either from the metadata, using XPath expressions to specify the metadata fields containing the URIs to the digital files, or by providing/uploading the digital files directly to the system. Apart from a web GUI, a REST API has been also implemented that allows external software components (such as the Validator or the Aggregator) to trigger and manage harvests and to obtain harvested metadata and digital files.

B. Automatic Validation of Metadata and Digital Files

The Validator is created with the aim to implement a validation model which operates in many levels: repository interoperability, metadata and digital file validation. We present the system architecture, the Validation Domain-Specific Language to express validation logic and some implementation details we consider valuable for consideration.

The Validator consists of two autonomous systems, the back-end and the front-end. The back-end works closely with the Harvester to retrieve content which is then validated using complex validation business rules at multiple levels (repository, metadata, digital files). Highly granular results are recorded and are made available through the user interface of the validator front-end. Some key system architecture points of the entire system are summarised in the following:

- **Interfaces:** Both systems have a web GUI for controlling every aspect of the validation processes. Furthermore, the Validator back-end is featuring a REST API that allows external software components to trigger and manage validation processes.

[5] http://core.kmi.open.ac.uk/search

[6] http://www.openarchives.gr/

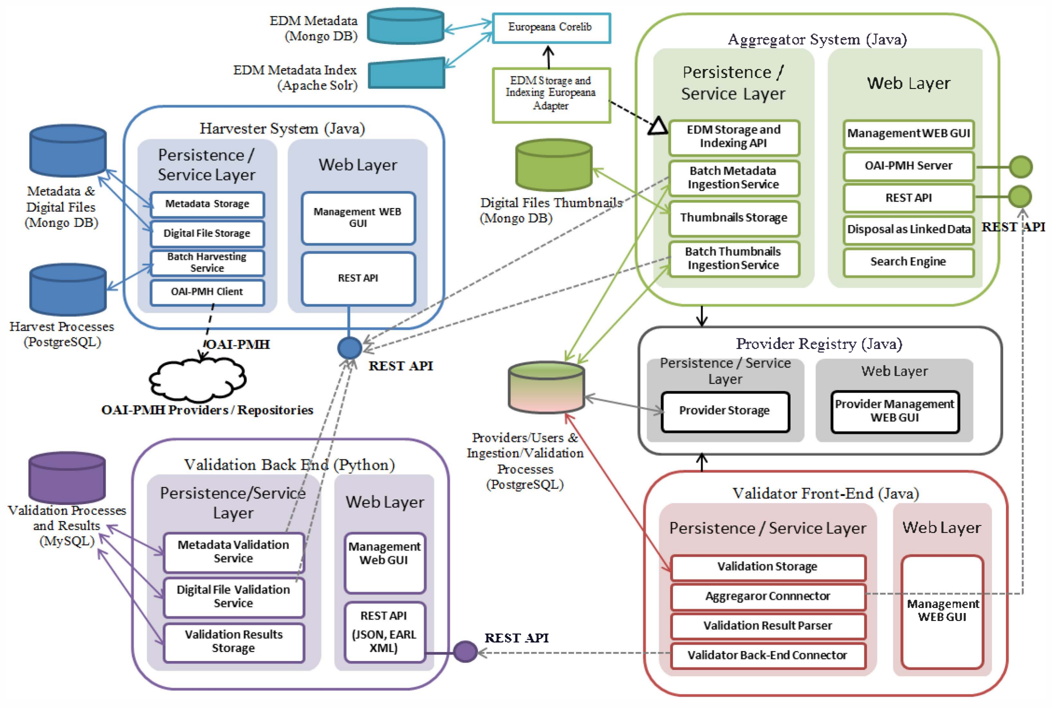
[7] http://nsdl.org/

[8] http://repox.ist.utl.pt/

[9] http://www.europeana.eu/portal/

[10] Mint (Metadata Interoperability Services): http://mint.image.ece.ntua.gr

[11] Apache Solr: https://lucene.apache.org/solr/



EDM Metadata (Mongo DB)
EDM Metadata Index (Apache Solr)
Europeana Corelib
EDM Storage and Indexing Europeana Adapter
Aggregator System (Java)
Persistence / Service Layer
Web Layer
EDM Storage and Indexing API
Batch Metadata Ingestion Service
Thumbnails Storage
Batch Thumbnails Ingestion Service
Management WEB GUI
OAI-PMH Server
REST API
Disposal as Linked Data
Search Engine
REST API
Digital Files Thumbnails (Mongo DB)
Harvester System (Java)
Persistence / Service Layer
Web Layer
Metadata Storage
Digital File Storage
Batch Harvesting Service
OAI-PMH Client
Management WEB GUI
REST API
Metadata & Digital Files (Mongo DB)
Harvest Processes (PostgreSQL)
OAI-PMH
OAI-PMH Providers / Repositories
REST API
Provider Registry (Java)
Persistence / Service Layer
Web Layer
Provider Storage
Provider Management WEB GUI
Providers/Users & Ingestion/Validation Processes (PostgreSQL)
Validation Processes and Results (MySQL)
Validation Back End (Python)
Persistence/Service Layer
Web Layer
Metadata Validation Service
Digital File Validation Service
Validation Results Storage
Management Web GUI
REST API (JSON, EARL XML)
REST API
Validator Front-End (Java)
Persistence / Service Layer
Web Layer
Validation Storage
Aggregaror Connector
Validation Result Parser
Validator Back-End Connector
Management WEB GUI

Figure 1 The Architecture of the infrastructure and service components

- **Input:** The input (metadata records or digital files) comes from the Harvester. A validation process may trigger a new harvest process in the harvester or utilize metadata/digital files that have been already harvested by the harvester.
- **Output:** The validation results are stored permanently and can be served anytime both in analytical (per metadata record/digital file) or aggregative form, via the REST API and the web GUI

1. Validation Domain-Specific Language (VDSL)

An important design choice considered the representation of the validation rules and the introduction of validation logic into the system. The approach of hard-coding the validation logic directly into the application code was rejected from the start, since it would result in a platform which would be hard to maintain and modify in the future, especially in view of the expected continuous evolution of validation requirements and rules. Instead, a dynamic platform was developed to support the definition of arbitrary validation models outside the application code with the use of a novel Validation Domain-Specific Language (VDSL). A domain-specific language (DSL) provides a notation tailored towards an application and is based on the relevant concepts and features of that domain [12]. We believe that the most elegant, efficient and extensible way to express complex digital repository, metadata and digital file validation logic is by creating a Validation Domain-Specific Language tailored to our needs. There are many advantages in creating special DSL for validation purposes:

- Business logic is directly converted to validation models without the need of software implementation.
- It is possible for non-programmers to define and update validation models according to business logic.
- It is possible to create an infinite number of validation models with very high flexibility, adapting to evolving requirements and external changes.
- The software core of the VDSL remains compact and maintainable regardless from business rules development.

The VDSL consists of the following building blocks: a) Metadata element validation rule constructs, b) metadata record validation rule constructs, c) digital file validation rule constructs, d) boolean operators, e) control flow operators. VDSL files are encoded in JSON and are stored in the validator Back-End configuration. Users may choose to apply any rule set to a digital repository of their choice.

Metadata element validation rule constructs are used to evaluate the value of specific XML elements. For instance, the following rule evaluates if <dc:language> elements follow the ISO 639 Standard for Language Codes [13].

```
"dc:language": {"language.iso639": {"spec_id": "repository13"}}
```

The notation indicates that all <dc:language> elements must be checked with the function iso639 of the python module language and the business rule which dictates this is identified by "repository13".

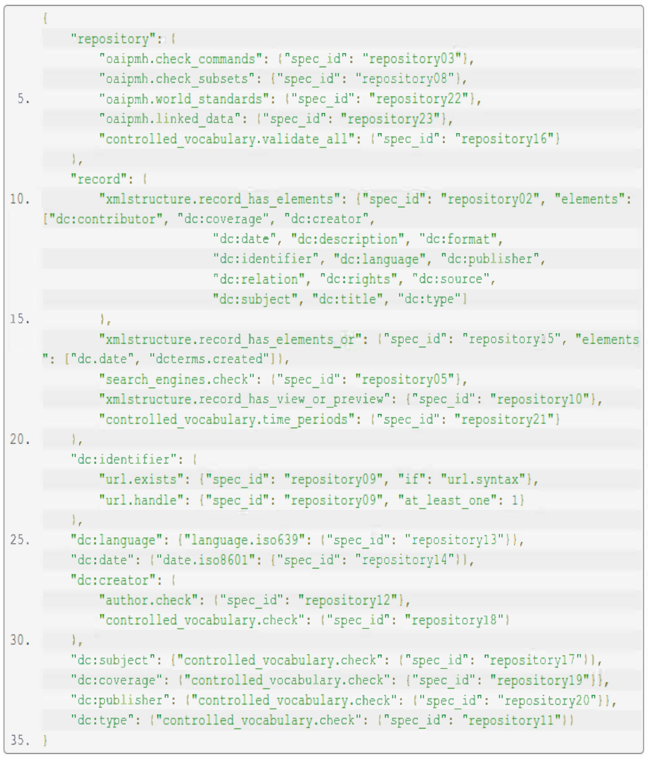
Metadata record validation rule constructs are used to evaluate the structure of whole XML metadata records. For instance, the following rule evaluates if records follow the Europeana Semantic Elements XML Schema.

```
"record": {
"xmlstructure.validate_with_xsd": {"spec_id": "repository07",
"xsd_url": http://www.europeana.eu/schemas/ese/ESE-V3.4.xsd}}
```

Digital file validation rule constructs are destined to evaluate digital files. For instance, we can evaluate the dimensions or color depth of images against some thesholds.

```
{ "files": {
"image.resolution": {"spec_id": "image01", "min_width": 800, "min_height": 600},
"image.colordepth": {"spec_id": "image01", "min_depth": 1}
} }
```

The use of boolean operators (AND, OR, etc) and Control flow operators (IF, ELSE, etc) is also critical to express complex rule sets. Figure 2 presents a complete validation rule set example.



```
{
    "repository": {
        "oaipmh.check_commands": {"spec_id": "repository03"},
        "oaipmh.check_subsets": {"spec_id": "repository08"},
        "oaipmh.world_standards": {"spec_id": "repository22"},
        "oaipmh.linked_data": {"spec_id": "repository23"},
        "controlled_vocabulary.validate_all": {"spec_id": "repository16"}
    },
    "record": {
        "xmlstructure.record_has_elements": {"spec_id": "repository02", "elements":
["dc:contributor", "dc:coverage", "dc:creator",
                        "dc:date", "dc:description", "dc:format",
                        "dc:identifier", "dc:language", "dc:publisher",
                        "dc:relation", "dc:rights", "dc:source",
                        "dc:subject", "dc:title", "dc:type"]
        },
        "xmlstructure.record_has_elements_or": {"spec_id": "repository15", "elements
": ["dc.date", "dcterms.created"]},
        "search_engines.check": {"spec_id": "repository05"},
        "xmlstructure.record_has_view_or_preview": {"spec_id": "repository10"},
        "controlled_vocabulary.time_periods": {"spec_id": "repository21"}
    },
    "dc:identifier": {
        "url.exists": {"spec_id": "repository09", "if": "url.syntax"},
        "url.handle": {"spec_id": "repository09", "at_least_one": 1}
    },
    "dc:language": {"language.iso639": {"spec_id": "repository13"}},
    "dc:date": {"date.iso8601": {"spec_id": "repository14"}},
    "dc:creator": {
        "author.check": {"spec_id": "repository12"},
        "controlled_vocabulary.check": {"spec_id": "repository18"}
    },
    "dc:subject": {"controlled_vocabulary.check": {"spec_id": "repository17"}},
    "dc:coverage": {"controlled_vocabulary.check": {"spec_id": "repository19"}},
    "dc:publisher": {"controlled_vocabulary.check": {"spec_id": "repository20"}},
    "dc:type": {"controlled_vocabulary.check": {"spec_id": "repository11"}}
}
```

Figure 2 Dublin Core Validation Ruleset Example

2. *Technical implementation considerations*

One of the greatest challenges in implementing the Validator Back-End is performance and responsiveness. The platform must be able to evaluate large datasets of metadata and digital files while maintaining a responsive Web UI and REST API for the Validator Front-End or any other system which needs establish communication and exchange information in parallel. To achieve this goal, we handle validation tasks using asynchronous job queues. The web application server maps a validation process into multiple individual atomic subtasks which are inserted in the asynchronous job queue of the system, stored in a Redis List[12]. Background workers, whose number equals the number of server CPU cores, are constantly monitoring the job queue for new tasks. As soon as they identify them, they begin processing them one by one and store the results in a Mysql database. What is more, the platform is highly scalable as it is possible for the asynchronous job queues to scale not only vertically depending on the number of available server CPU cores, but also horizontally, as multiple servers can be configured to share the same asynchronous job queue and mysql database.

C. *Aggregator*

The Aggregator in our solution is the system that aggregates metadata from registered content providers, creates and stores thumbnails for digital files and provides with a series of added-value services, such as a metadata search engine and the disposal of the ingested metadata as Linked Data and for harvesting via OAI-PMH. The metadata records and the digital files from which thumbnails are created derive from the Harvester. The aggregator stores metadata internally in the Europeana Data Model (EDM) [6], which is the new proposal of Europeana for structuring cultural metadata. The Aggregator supports an extensible pool of transformations from well-known metadata formats to EDM.

The Aggregator uses the Europeana EDM Storage component, named Europeana Corlib, as an EDM Storage and Indexing backend. The integration with the Europeana Corelib is done through a generic EDM Storage and Indexing API which is agnostic to the storage backend that is actually used and can cover any storage system able to persist and index EDM structures. The Europeana Corelib stores EDM metadata records in Mongo DB and indexes them using the Apache Solr.

We designed and implemented a solid and extensible ingestion workflow that starts from the retrieval of metadata/digital files from the Harvester (using its REST API), provided that are already validated by the Validator, the enforcement of the appropriate transformations, including metadata format transformations and URI conversions, and ends up in the persistence of the EDM metadata records in the EDM Storage and Indexing System and of the thumbnails derived from the digital files in a noSQL database (Mongo DB: GridFS). The workflow, which is illustrated in Figure 3 is implemented in a pipe-lined fashion, consisting of a series of modular operators and adopts the pull data-flow pattern, needing no materialization of intermediate results.

IV. Validation And Ingestion Workflow

The full workflow of the platform is shown in Figure 3. The validation is triggered from the Validator Front-End by an authorized user, who after selecting the provider under validation and the validation ruleset and setting the harvest parameters (metadata sets, date from-until, etc) (Figure 3: 1), starts a new validation process. Then, a REST call 'Create Validation request' is sent to the Validator Back-End containing the harvest parameters (Figure 3: 2, 3) which, in turn, sends a REST call 'Start harvest process' to the Harvester (Figure 3: 4). The Harvester starts the harvesting process and sends to the Validator Back-End the id of the process (Figure 3: 5, 6, 7).

[12] http://redis.io/topics/data-types

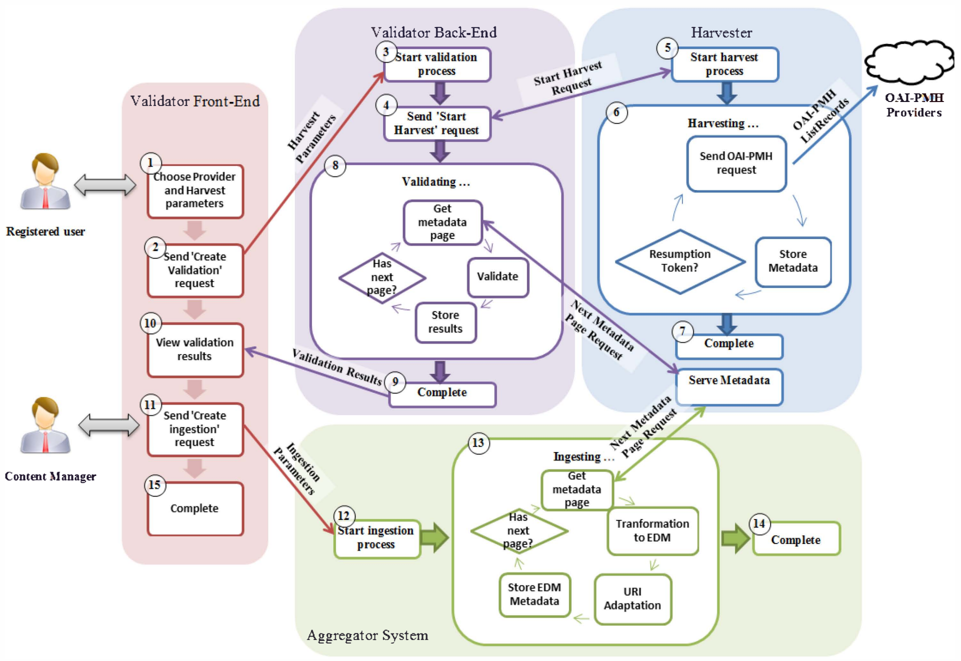


Figure 3 Validation and Ingestion workflows

Both ids of the harvesting and the validation process are sent to the Validator Front-End. In the meantime the Validator Back-End requests the content from the Harvester and, as soon as this is available, it begins the validating procedure (Figure 3: 8). The user can monitor the process from the Validator Front-End and as soon as the validation procedure is finished, the results are available (Figure 3: 9, 10). If the validation is successful, the Content Manager can authorize the ingestion process (Figure 3: 11). In this case, a REST call 'Start Ingestion process' is send to the Aggregator, containing the id of the harvest process (Figure 3: 11, 12). The Aggregator requests the data from the Harvester and starts the Ingestion process (Figure 3: 13).

V. Experimental Results

Experiments were conducted using three OAI-PMH datasets, dataset-1 of 82,000 metadata records, dataset-2 of 205,000 metadata records and dataset-3 of 410,000 metadata records. The datasets originated from harvesting the Hispana organization[13] in ESE[14] metadata formal. All experiments were run on an Intel i5 2.80GHz PC with 4GB of RAM, running MS Windows 7. We used local single-node/non-clustered installations of Mongo DB, Apache Solr and PostgreSQL. A small portion of the harvested records were deleted records, thus lacking metadata content, as illustrated in Table 1.

A. *Harvester Scaling Evaluation*

In an effort to illustrate the harvester scalability, we started a harvest process using as input dataset-3. The harvest took place off-line; we had pre-downloaded the OAI-PMH responses and fed the Harvester with these stored XML files, in order to exclude network and repository response delays from our measurements.

Table 1. Datasets

	Total records	"deleted" records	records with metadata	size in MB
dataset-1	82,004	4	82,000	138 MB
dataset-2	205,920	920	205,000	344 MB
dataset-3	432,435	22,435	410,000	679 MB

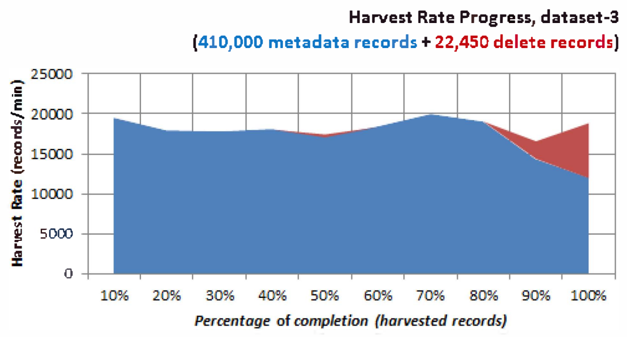


Figure 4. Harvest rate scaling during dataset-3 harvesting process

Figure 4, illustrates the average harvest rate per chunk of 10% (43,243 records, respectively) towards completion. The results depicted almost constant harvest rate throughout the entire process, averaging at 18,344.41 records per minute. The last 20% of the records included an increasing number of deleted (empty) records, accelerating the process, as shown in the graph.

B. *Aggregator Scaling Evaluation*

We conducted similar experiments in order to evaluate whether the Aggregator also scales well for increasing amount of metadata. We excluded the "deleted" records (Table 1, 2nd Column) from this experiment and used only those which included metadata content (Table 1, 3rd Column). The

[13] http://hispana.mcu.es

[14] http://pro.europeana.eu/ese-documentation/

ingestion process is a pipelined process that includes fetching metadata records from the Harvester, the transformation of each record from ESE XML format to a corresponding EDM graph with valid and accessible URIs and the storage of this graph in the EDM Storage and Indexing System, namely, Europeana Corelib, which implies storing the graph in Mongo DB structures and fully indexing each document value by an Apache Sorl installation.

Figure 5 illustrates the average ingestion rate for dataset-1 (82,000 metadata records) and dataset-3 (410,000 metadata records), respectively. The ingestion rate is illustrated per chunk of 10% (8,200 records for dataset-1 and 41,000 records for dataset-3, respectively). The results depicted almost constant ingestion rate throughout the entire process, averaging at 2236.77 records/minute for dataset-1 and at 2,127.06 records/minute for dataset-3. The average ingestion rates for the three datasets are shown in Figure 6.

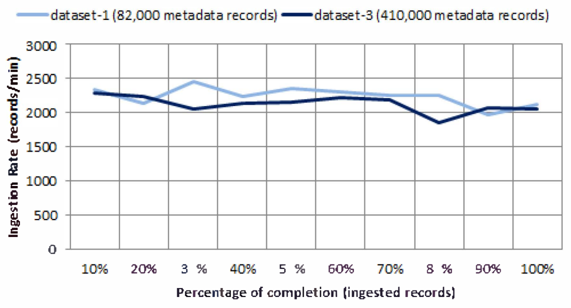


Figure 5. Ingestion rate scaling during ingestion pocesses of dataset-1 and dataset-3 (ignoring delere records).

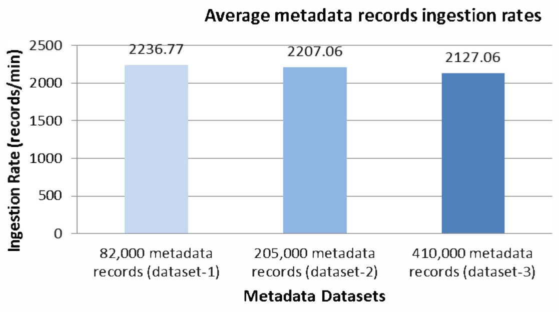


Figure 6 Average ingestion rates per dataset

C. *Validator Back-End Scaling Evaluation*

Finally, we evaluated whether the Validator Back-End is capable to cope with the rest of the infrastructure in terms of performance. Using the datasets presented in Table 1, we performed full evaluations using a complete VSDL Ruleset such as the one presented in Figure 2, with only a minor modification: we removed the validation rules which required network connections with external systems in order to avoid delays induced by 3rd parties. Figure 7 illustrates the average validation rate for all datasets. The average validation rates vary from 1822.491 to 1845.12 records/minute, which are on par with the performance of the rest of the system components as presented in Figures Figure 4, Figure 5 and Figure 6.

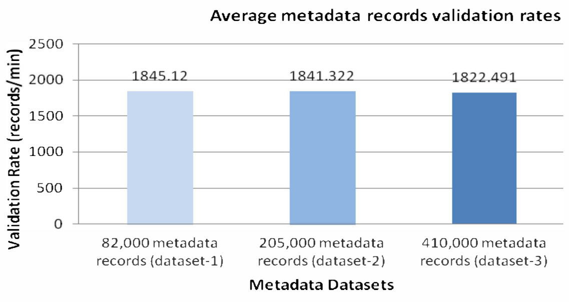


Figure 7 Average validation rates per dataset

VI. CONCLUSIONS-FUTURE WORK

In this paper, we presented an Open Cultural Digital Content Infrastructure, which consists of (1) a Harvester, (2) a Validator and (3) an Aggregator component, which support metadata and digital file validation and ingestion. We presented the workflow of the infrastructure and emphasized on the key aspects of the architecture. The Validator was developed to support the definition of arbitrary validation models outside the application code with the use of a novel Validation Domain-Specific Language (VDSL). Finally, we conducted an experimental study which demonstrated that all system components scale well.

ACKNOWLEDGMENT

The work presented in this article has been partly supported by the project "Platform for provision of services for deposit, management and dissemination of Open Public Data and Digital Content" (Ref No 327378) which is co-funded by Greece and the European Union-European Regional Development Fund through the Operational Programme "Digital Convergence" (NSFR).

REFERENCES

[1] Stathopoulos et al. 2013. Specifications and features for the interoperability of open digital content. http://helios-eie.ekt.gr/EIE/handle/10442/8887

[2] Schirrwagen, J., Manghi, P., Manola, N., Bolikowski, L., Rettberg, N., & Schmidt, B. (2013). Data Curation in the OpenAIRE Scholarly Communication Infrastructure. Information Standards Quarterly, Fall, 25(3), 13-19.

[3] Klerkx J, Vandeputte B, Parra G, Santos JL, Van Assche F, Duval E (2010) How to share and reuse learning resources: the ARIADNE experience. Sustaining TEL: from innovation to learning and practice. Lect Notes Comput Sci 6383/2010:183–196.

[4] Jelliffe, Rick. "The schematron assertion language 1.5." Academia Sinica Computing Center (2000).

[5] Troncy, R., Bailer, W., Höffernig, M., Hausenblas, M. (2010). VAMP: a service for validating MPEG-7 descriptions wrt to formal profile definitions. Multimedia Tools and Applications, 46(2-3), 307-329.

[6] Europeana Data Model Primer, available at http://pro.europeana.eu/edm-documentation

[7] Anne Benoit, Ümit V. Çatalyürek, Yves Robert, and Erik Saule. 2013. A survey of pipelined workflow scheduling: Models and algorithms. ACM Comput. Surv. 45, 4, Article 50 (August 2013)

[8] Van Deursen, Arie, and Paul Klint. "Domain-specific language design requires feature descriptions." CIT. Journal of computing and information technology10.1 (2002): 1-17

[9] ISO, John D. Byrum. "639-1 and ISO 639-2: International Standards for Language Codes. ISO 15924: International Standard for names of scripts." 65th IFLA Council and General Conference Bangkok, Thailand. 1999.