Distrib Parallel Databases (2018) 36:153–194
https://doi.org/10.1007/s10619-017-7210-4

MetaStore: an adaptive metadata management framework for heterogeneous metadata models

Ajinkya Prabhune[1] · **Rainer Stotzka**[1] · **Vaibhav Sakharkar**[1] · **Jürgen Hesser**[2] · **Michael Gertz**[3]

Published online: 4 October 2017

Abstract In this paper, we present MetaStore, a metadata management framework for scientific data repositories. Scientific experiments are generating a deluge of data, and the handling of associated metadata is critical, as it enables discovering, analyzing, reusing, and sharing of scientific data. Moreover, metadata produced by scientific experiments are heterogeneous and subject to frequent changes, demanding a flexible data model. Existing metadata management systems provide a broad range of features for handling scientific metadata. However, the principal limitation of these systems is their architecture design that is restricted towards either a single or at the most a few standard metadata models. Support for handling different types of metadata models, i.e., administrative, descriptive, structural, and provenance metadata, and including community-specific metadata models is not possible with these systems. To address this challenge, we present MetaStore, an adaptive metadata management framework based on a NoSQL database and an RDF triple store. MetaStore provides a set of core functionalities to handle heterogeneous metadata models by automatically generating the necessary software code (services) and on-the-fly extends the functionality of the framework. To handle dynamic metadata and to control metadata quality, MetaStore also provides an extended set of functionalities such as enabling annotation of images and text by integrating the Web Annotation Data Model, allowing communities to define discipline-specific vocabularies using Simple Knowledge Organization System, and providing advanced search and analytical capabilities by integrating the ElasticSearch. To maximize the utilization of the data models supported by NoSQL

✉ Ajinkya Prabhune
ajinkya.prabhune@kit.edu

[1] Karlsruhe Institute of Technology, Eggenstein-Leopoldshafen, Germany

[2] Department of Radiation Oncology, Heidelberg University, Heidelberg, Germany

[3] Institute of Computer Science, Heidelberg University, Heidelberg, Germany

databases, MetaStore automatically segregates the different categories of metadata in their corresponding data models. Complex provenance graphs and dynamic metadata are modeled and stored in an RDF triple store, whereas the static metadata is stored in a NoSQL database. For enabling large-scale harvesting (sharing) of metadata using the METS standard over the OAI-PMH protocol, MetaStore is designed OAI-compliant. Finally, to show the practical usability of the MetaStore framework and that the requirements from the research communities have been realized, we describe our experience in the adoption of MetaStore for three communities.

Keywords MetaStore · NoSQL database · Automated code generation · Annotations

1 Introduction

With the advent of novel data acquisition systems and the ability to process enormous amounts of data, there is an exponential growth in the volume of data and metadata that are generated by scientific experiments [1,2]. To systematically describe and execute various data processing steps, research communities are rapidly adopting scientific workflows [3–7]. In the past, researchers have heavily relied on ad hoc techniques for managing metadata. For example, metadata was stored in lab journals, workflows were described using non-standard scripting languages, and data description were captured using obscure file or folder names. However, these approaches for handling metadata are inefficient and hinder research. Various aspects of scientific research such as discovering, reusing, sharing, and analyzing data are entirely dependent on the metadata. Metadata is crucial for managing the complete life-cycle of scientific data, for example, automating scientific analysis workflows, enabling data access, enhancing data interpretation by visual exploration and creating metadata-aware tools [8].

Metadata can be broadly classified into static metadata and dynamic metadata [9]. Static metadata describing the data is not subject to any changes over the complete life of the data, even if the data evolves. For example, the metadata describing the preparation of a specimen and the optical configuration [10] of the microscope for a nanoscopy investigation is static metadata that is immutable and prohibited to changes, while any modifications to this metadata will invalidate the investigation and the subsequent results. Contrarily, dynamic metadata is subject to change, i.e., as the data evolves the metadata may also change.

Based on the applications of metadata, static metadata is categorized into three main types: descriptive metadata, structural metadata, and administrative metadata [11]. Descriptive metadata describes the details of the resource (data) and is used for searching and identifying the resource. Additionally, in scientific research, any dynamic description can also be tagged to a given resource through annotations. For example, in medical research, it is a common practice for the medical experts (physicians) to attach additional information on the patient's medical images in the form of annotations [12,13]. Also, in the research areas of the humanities, digitized medieval manuscripts are enriched by humanities scholar with additional description in the form of annotations [14]. Structural metadata describes the internal structure of compound resources and the relationship among its elements. For example, the structural information describing the relations between the pages of a medieval manuscript can be described with structural metadata. Administrative metadata describes the details necessary to manage the resource, such as access and rights management and resource preservation information. An additional type of metadata is provenance metadata, which is necessary for capturing the lineage of how a resource was generated. To model these different types of metadata, various metadata standards (models) are available. These standards are typically serialized in XML with a corresponding XML Schema Definition (XSD) [15].

For modeling the metadata, each scientific community either uses an existing metadata schema or defines a community-specific metadata schema. Moreover, to enable storing, querying, retrieving, analyzing and sharing of the metadata, these communities develop restricted metadata management frameworks that are capable of handling only specific metadata models (schemas) [16–19]. These solutions are not generic and cannot be applied to other metadata models. When introducing a new metadata model in the existing solution, the following software engineering tasks necessary for a database schema migration have to be performed: (1) The existing database schema needs to be updated with the new metadata model. If the current schema contains metadata records, then an additional step of data migration needs to be performed. (2) The source code, i.e., the software and services that implement the functionality across the schema need to be updated, tested, compiled and redeployed. Thus, these time and resource intensive software engineering tasks need to be performed each time when a new metadata model is introduced in the system.

To address the limitations of existing metadata management systems and to fulfill the requirements from the nanoscopy, eCodicology, and the Corpus Vitrearum Deutschland (CVD) research communities, we coarsely identify the requirements necessary for designing a generic reusable metadata management system, and in Sect. 2, we extend these requirements and provide a detailed description for each: (1) an automatically adapting metadata management framework for handling heterogeneous metadata schemas, (2) a flexible multi-model data storage (database) system for efficiently storing and retrieving both static and dynamic metadata, (3) an integration of a standard annotation data model for handling dynamic metadata in the form of annotations, (4) support for scientific communities to define or integrate discipline-specific vocabularies, and (5) conformance to a standard metadata harvesting protocol with an extensible metadata model for enabling sharing of metadata.

Contributions In the previous version of MetaStore, we established the core functionality of the framework required for handling static metadata [20]. Following features were realized in the first version of MetaStore:

– A dedicated metadata registry for registering metadata schemas, necessary for tracking schema evolution and validating ingested metadata.
– An automated code (service) generator for dynamically generating the necessary functionality for handling registered metadata schemas.
– An automated extraction of index terms from metadata schemas with the creation of indexes in NoSQL document databases for enabling full-text search.

- Automated modeling of workflow provenance in the ProvONE [21] provenance model and allowing provenance interoperability between ProvONE and PREservation Metadata: Implementation Strategies (PREMIS) [22] provenance models.
- Support towards metadata harvesting standards, namely the OpenArchives Initiative Protocol for Metadata Harvesting (OAI-PMH) [23] with the extension of Metadata Encoding and Transmission Standard (METS) [24] for enabling comprehensive metadata sharing.

With the core functionality for handling static metadata established, we extended MetaStore with the advanced requirements put forth by the research communities. Hence, in this version of MetaStore, the following new features particularly for handling dynamic metadata are introduced.

- Realization of a WADM annotation framework for handling automatically generated annotations from scientific workflows and semi-automatically created annotations from discipline-specific tools.
- Extended metadata quality control with modeling of discipline-specific vocabularies based on the SKOS [25] data model, as well as support for registering external vocabulary providers.
- A systematic separation of static metadata models from dynamics models with the adoption of a standard RDF data model with SPARQL querying capabilities.
- Interface for extending MetaStore schema registry with an RDA metadata directory registry for supporting existing metadata models.
- Migration of provenance from a non-standard graph model to an RDF data model, with realization of SPARQL queries for analyzing provenance graphs and annotations.
- Integration of ElasticSearch [26] for allowing in-depth faceted and analytical search capabilities.

The implementation of the MetaStore framework presented in this paper is built on ArangoDB,[1] a native multi-model NoSQL database, and Apache Jena, a framework for building semantic web and linked data applications [27]. The prime reason for choosing ArangoDB is due to its compactness to store key-value pairs, documents and graph data in a single database.

With the possibility to serialize the descriptive, administrative and structural metadata schemas in XML, the document data model of ArangoDB is an appropriate choice to store static metadata, whereas an RDF database like Apache Jena is suitable to store dynamic metadata in the form of annotations. As no complex analytical queries are expected over the registered schemas, the key-value model is adequate for registering metadata schemas with their namespace and version as simple key-value pairs. For modeling workflow definition and its associated provenance, the RDF data model is the most appropriate one, because it not only allows to store provenance information as a graph but also complex pattern matching queries using SPARQL can be efficiently designed and executed. For automating the modeling of workflows in ProvONE, we use the Prov2ONE algorithm [28]. However, the MetaStore framework is database agnostic and can also be realized using a combination of the key-value store RedisDB

[1] https://www.arangodb.com/.

[29], the document store MongoDB [30] and the graph database management system Neo4j [31].

Organization of the paper The remainder of the paper is structured as follows. In Sect. 2, we present a detailed description of the requirements that motivate the design and features of MetaStore. Section 3 presents the complete multilayered architecture of the MetaStore framework, with a detailed description of the functionality provided by each layer. In Sect. 4, we present a brief overview of the features available in existing metadata management systems and frameworks. Section 5 presents feature-based and performance comparison of MetaStore with the existing metadata systems presented in Sect. 4. To illustrate the generic applicability of MetaStore, in Sect. 6, we describe the adoption of MetaStore for three scientific use cases. In Sect. 7, we describe the rationale behind the architecture design decisions that led to the implementation of MetaStore. Finally, Sect. 8 concludes the paper with a brief outline of our ongoing work.

2 Requirements for the MetaStore framework

The primary aim of developing the MetaStore framework is to provide a generic and reusable metadata management system that can be adopted by multiple scientific communities with different needs. This aim is further strengthened by the requirements put forth by the nanoscopy, eCodicology, and the CVD research communities. In the following, we summarize these requirements:

Metadata schema support Research in several disciplines is rapidly changing, especially the nanoscopy and eCodicology research areas are continuously evolving and their metadata schemas are frequently modified. In nanoscopy research, a community-specific metadata schema is created to model the experiment description,[2] and workflow provenance is described using ProvONE. Similarly, the results of eCodicology workflows are modeled using the standard PAGE XML schema.[3] Hence, the requirement is to design MetaStore to support both, standard metadata schemas as well as community-specific metadata schemas. Additionally, to systematically handle the evolution of metadata schemas in order to mitigate redundant modifications and enable tracing of different schema versions, MetaStore should provide the registration of each metadata schema with its version. Moreover, nanoscopy and eCodicology research communities have adopted METS to systematically organize multiple metadata schemas that are required by the community to describe their data comprehensively. Hence, the requirement for MetaStore is to handle community-specific METS-profiles,[4,5]

Full-text search For the majority of the communities, the data is huge and serialized in machine-readable file formats, and frequently accessing the data is not efficient due to hardware and network limitations. In nanoscopy and eCodicology research,

[2] http://datamanager.kit.edu/masi/localizationmicroscopy/2016-03/LocalizationMicroscopy.xsd.

[3] http://www.primaresearch.org/schema/PAGE/gts/pagecontent/2017-07-15/pagecontent.xsd.

[4] http://datamanager.kit.edu/masi/localizationmicroscopy/mets/nanoscopy-METS-profile.xml.

[5] http://zimks68.uni-trier.de/stmatthias/T1108/T1108-digitalisat.xml.

raw data acquired either from high-resolution microscopes or scanners is ingested in a data repository. For enabling further reuse of this data by either data-processing workflows or domain-specific applications, the data needs to be made discoverable. For this, metadata is critical for searching, identifying, and retrieving required data. A standard approach is to provide a set of queries for each metadata schema. However, this is not a sustainable approach, because for frequently changing metadata schemas, the implemented queries will quickly become obsolete and return either incomplete or incorrect results. Moreover, implementing a set of queries for each metadata schema is an arduous and resource intensive task. Thus, the requirement from the communities is to have a full-text search over all the metadata.

Metadata quality control The next requirement is to provide automated metadata quality control for verifying the well-formedness of metadata, schema conformance, and content verification for communities possessing controlled vocabularies. Metadata can be automatically generated by a data acquisition system or during the execution of workflows, or metadata in the form of annotations can be manually created and tagged with the data by domain experts. Thus, irrespective of the source from which the metadata is generated, for maintaining a basic level of metadata quality, the MetaStore framework should provide different levels of automated quality control checks.

Metadata harvesting The metadata should be provided for allowing harvesting, i.e., from the collected metadata, the MetaStore framework should provide either partial metadata harvesting, in the case of sharing only a specific part of metadata to an external system or complete metadata harvesting, in the event of data migration from one data repository to another. For enabling seamless integration of MetaStore with existing systems and compliance with existing metadata harvesting standards, the MetaStore framework should support the OAI-PMH specification. In the eCodicology project, the processing of digitized manuscripts results in large volumes of multi-dimensional metadata. As this metadata needs to be visualized using the CodiViz [32] tool, selective metadata harvesting is requested by the humanities scholars. Thus, a common requirement from the research communities is to allow large-scale metadata sharing.

Provenance support The nanoscopy data processing workflows are described using the BPEL language, while the eCodicology workflows are described in SCUFL. During the execution of these workflows, it is necessary to capture and model the comprehensive workflow provenance. For both research communities, provenance is critical metadata that assists in validating the quality of results, enables data reproducibility, provides systematic tracking of workflow evolution, and allows execution of graph pattern queries for analyzing provenance traces. Moreover, as MetaStore can also be coupled with a scientific data repository, it is necessary to support the PREMIS provenance model used in long-term data and metadata preservation. Hence, the requirement is to not only provide automated workflow provenance handling capabilities using a comprehensive provenance model like ProvONE but also support the data preservation in the PREMIS model.

Automatic/Semi-automatic annotations Currently, in the field of digital humanities, research communities often apply image processing workflows that generate metadata at each step. For the eCodicology community, the SWATI workflow [32] is employed to extract Optical Layout Recognition (OLR) features from the digitized medieval

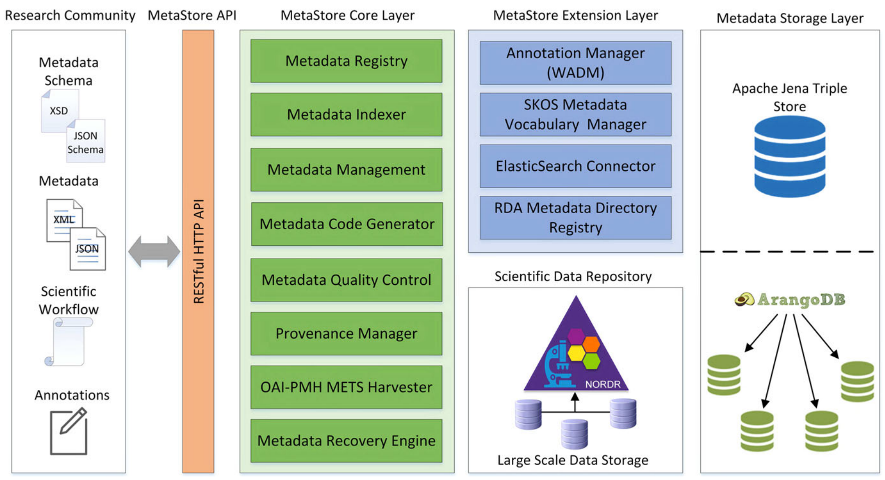


Fig. 1 Multilayered architecture of MetaStore

manuscripts. As the SWATI workflow is continuously evolving with improved layout recognition algorithms, the extracted OLR features (metadata) are also frequently changing. Hence, the first requirement for MetaStore is to store this metadata in the form of annotations and make it available for analysis. Furthermore, these annotations are subject to modification by research experts through the CodiLab toolset,[6] which is an annotation framework providing visual interpretation of annotations by overlaying them over digitized manuscripts. Thus, the second requirement is to allow semi-automated annotations of images and documents, with an extension for controlled vocabularies.

Miscellaneous In the following, we list the additional requirements requested by the research communities: logically linking the metadata with the data through Persistence Identifier (PID) generated from a PID management systems like the Handle System,[7] high-performance metadata access and ingest, scalable storage for handling increasing metadata volume, and a REST API for integrating MetaStore with existing systems.

3 MetaStore architecture

In this section, we present the multilayered architecture of the MetaStore framework (see Fig. 1). The architectural design of MetaStore is based on the following design principles.

Modular design Each functionality in MetaStore is available as a separate component (module), so that any enhancement or technology change can be systematically handled by updating or replacing that specific component, without affecting the other components in the framework.

6 https://github.com/JochenGraf/CodiLab.

7 http://www.handle.net/index.html.

Adaptive feature enhancement MetaStore is designed as an entirely adaptive framework that modifies itself in handling ad hoc metadata models. For this, we designed MetaStore based on the principle of Compositional Adaption [33]. Compositional Adaption can be realized with the following technologies: Separation of Concerns (SoC), Computational Reflection, and Component-based design. Out of these technologies, we adopt the Component-based design architecture. The Component-based design architecture supports two types of compositions; static and dynamic composition. In static composition, the program is composed at compile time and cannot be changed without recoding, whereas in a dynamic composition, components can be added, removed, modified, or reconfigured at runtime. To design MetaStore as a dynamically adapting framework, we realized the dynamic composition.

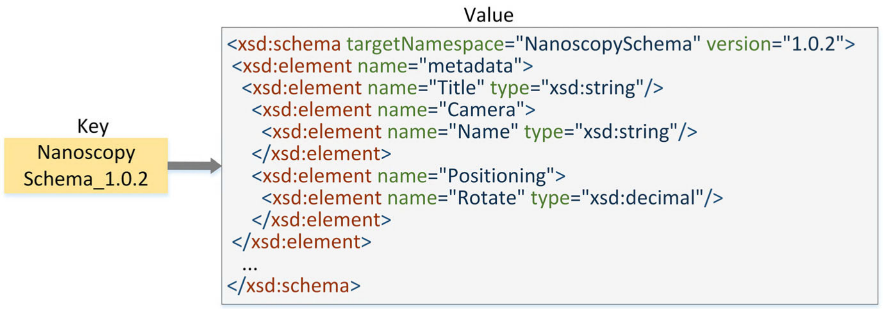
The complete functionality of MetaStore is divided into two layers: (1) The **MetaStore Core Layer** provides the basic functionalities necessary for handling the static metadata, and is designed independent of any scientific data repository system or metadata storage technology. (2) The **MetaStore Extension Layer** allows integration of third-party libraries and tools necessary for handling the dynamic metadata and supports handling of SKOS modeled controlled vocabularies that are specific to scientific disciplines. In the following subsections, the description of each layer and the realization of the features through the different MetaStore components are presented.

3.1 Research community

A research community performs metadata related interactions through the REST API exposed by the MetaStore framework. A valid metadata schema defined either as XSD or JSON Schema is submitted by the research community for registering it in MetaStore. Only metadata complying to a registered schema can be inserted in MetaStore. Similarly, for modeling provenance information in the ProvONE model, scientific workflows defined by a research community are submitted to MetaStore. Currently, workflows defined in the Business Process Execution Language (BPEL) [34], Simple Conceptual Unified Flow Language (SCUFL) [35], and Modeling Markup Language (MoML) [36] are supported by MetaStore. The semi-automated annotations provided by domain experts are captured in the WADM and SPARQL [37] end-points are exposed for searching and retrieving the annotations. An OAI-PMH metadata harvester exposed as a REST interface is provisioned to the research community for sharing their metadata.

3.2 MetaStore core layer

The MetaStore Core Layer consists of various task-specific components that collectively build the functionality of MetaStore. Each component or group of components in the MetaStore Service Layer follows a well-defined workflow to accomplish a given task. Through these components, we have implemented the features corresponding to the requirements from Sect. 2.



```
<xsd:schema targetNamespace="NanoscopySchema" version="1.0.2">
 <xsd:element name="metadata">
  <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="Camera">
    <xsd:element name="Name" type="xsd:string"/>
   </xsd:element>
   <xsd:element name="Positioning">
    <xsd:element name="Rotate" type="xsd:decimal"/>
   </xsd:element>
 </xsd:element>
  ...
</xsd:schema>
```

Fig. 2 Metadata schema registration as key-value

3.2.1 Metadata registry and indexer

For supporting systematic handling of heterogeneous metadata schemas, we designed the Metadata Registry component based on the key-value data model. Registering a metadata schema with its namespace and version allows efficient schema validation checks during the metadata quality control process.

The Metadata Registry component allows the registration of either a single metadata schema that exclusively describes administrative, descriptive, structural metadata or a community defined METS XML that may comprise multiple metadata schemas. The Metadata Schema Registry allows research communities to maintain multiple metadata schemas with different versions. In the case of a single metadata schema, the schema is registered as a key-value pair, in the key-value data model of ArangoDB, where the key is the combination of schema namespace and version and the value is the complete schema. In the case of a METS XML, the metadata schemas used in constructing the various sections of the METS-profile are extracted and individually registered. Moreover, a relation map of the metadata schemas and the corresponding METS-profile is created. This relation map is necessary for segregating the metadata during the ingest stage, and for reconstructing the metadata during the harvesting stage. A reduced version of the nanoscopy descriptive metadata schema registered as a key-value pair is shown in Fig. 2.

Typically, in NoSQL databases, indexes have to be created manually, which is an arduous and time-consuming task, especially when dealing with complex scientific metadata schemas. For enabling full-text search over metadata, we coupled the Metadata Indexer component with the Metadata Registry component. The rationale behind this design decision was to enable automated creation of indexes during the registration of a metadata schema in MetaStore. With immediate indexing of a registered metadata schema, we guarantee that full-text search is enabled from the first insertion of metadata in MetaStore. Moreover, an update to a schema does not require manual interference from the user (system administrator), because the index terms are extracted and reapplied on the database collection, thus, updating the index list.

The Metadata Indexer component implements the IndexTermExtractor algorithm illustrated in Algorithm 1. The algorithm is similar to the Depth First Search (DFS) algorithm. For an input XSD or XML (in the case of METS), the algorithm recursively iterates the entire depth of the XSD or XML, and during each iteration, a unique index term path is added to the list L. In the first step, the VERIFYSCHEMA method determines if the input schema is a simple schema or a composite schema like METS. In the case of a simple schema, the EXTRACTTERMS method is invoked. For METS files the algorithm first decomposes all the metadata sections (DECOMPOSEMETS method), and for each section, the EXTRACTTERMS method is invoked. The EXTRACTTERMS method recursively constructs a unique index path for all the elements at a depth of $n - 1$ and adds it to the list L. After completion of each recursion, the path P has to be reset to the completed element x_i. For this, the SUBSTRING method is recursively invoked for removing the sub-path that has already been added to the list L.

In ArangoDB, as the indexing of the leaf elements is contained by the element at a depth of $n - 1$ (parent element), the algorithm only needs to construct the unique path for a depth of $n - 1$ for XSD or XML with a nested depth of n. The algorithm terminates when all the elements x_i for the depth of $n - 1$ are added to the list L. After the termination of the algorithm, the index terms from list L are applied to the respective metadata collection.

Algorithm 1 IndexTermExtractor

```
    Input: Metadata schema $X$ serialized in XSD or XML
                          ▷ $X = \{x_1, x_2, ...x_n\}$ is a vector of nested XML child elements
    Output: List $L$ containing index terms
 1: List S := ∅                              ▷ list containing the various sections in METS
 2: String P := ∅                            ▷ fully qualified path of the index term
 3: List L := ∅                              ▷ list containing unique index terms
 4: function VERIFYSCHEMA($X$)
 5:     if $X.namespace \in METS$ then
 6:         S = DECOMPOSEMETS($X$)
 7:                  ▷ the $\langle dmdSec\rangle$,$\langle amdSec\rangle$,$\langle structMap\rangle$, ... sections are extracted by this method
 8:         for $s_i \in S$ do
 9:             P = $s_i.rootElement$;
10:             EXTRACTTERMS($s_i$)
11:                          ▷ for each section in METS, the method extractterms is called
12:         end for
13:     else
14:         P = $X.rootElement$;             ▷ for flat XML structure assign rootElement to P
15:         EXTRACTTERMS($X$)                ▷ for a simple metadata schema
16:     end if
17: end function
18: function EXTRACTTERMS(X)
19:     for $x_i \in X$ do
20:         if $x_i.children \neq \emptyset$ then
21:             P = P ∪ $x_i$
22:             EXTRACTTERMS($x_i$)
23:         else
24:             L = L ∪ P                    ▷ Add all the unique index term paths to list L
25:             P = $X.rootElement$          ▷ Reset the path to the root element
26:         end if
27:         P = P.SUBSTRING($x_i$)           ▷ Reset P to the current path of the element $x_i$
28:     end for
29: end function
```

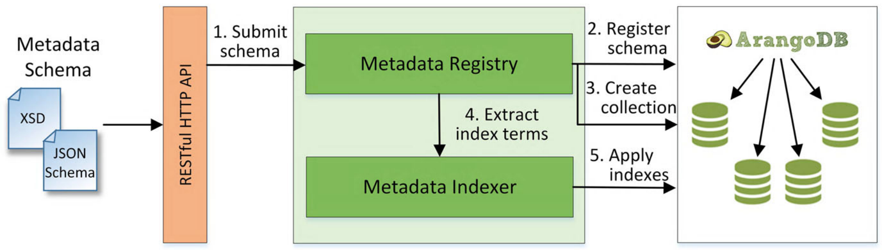


Fig. 3 Metadata schema registration and indexing

For the default configuration of MetaStore, when ElasticSearch indexing is disabled, the metadata schema registration and indexing process illustrated in Fig. 3 is followed. This process consists of the following five steps: (1) The scientific community submits their metadata schema through the MetaStore REST API, which is verified for the schema name and the version. (2) If the submitted schema with a given version does not exist in the schema registry, it is added to the schema registry. (3) For each unique schema registered, a corresponding collection is created in the document store of ArangoDB. This collection is required for storing the succeeding metadata, thus allowing a clear separation of community-specific metadata storage. (4) Once the metadata schema is successfully registered, the Metadata Indexer component analyzes the registered metadata schema to generate a list of index terms. (5) These index terms are applied to the respective collection for enabling full-text search. If a new version of an existing metadata schema is registered, the corresponding indexes are updated.

3.2.2 Metadata code generator

The principle aim of the Metadata Code Generator is to allow scientific communities to handle heterogeneous metadata schemas without the need to write any software code (services). To realize the dynamic composition of the services, the MetaStore Code Generator follows a two-step process. In the first step, the MetaStore Code Generator component automatically adapts the existing MetaStore installation by creating the services that are necessary for handling the registered metadata. Once the services are generated, they are dynamically added to the existing pool of services in the Metadata Management component. However, to expose these new services through the REST interface, the Metadata Code Generator invokes the entire compilation and redeployment of MetaStore. In the second step, to prevent the disruption of an active instance of MetaStore, the recompiled version of MetaStore is deployed on an auxiliary web-server. This adaptive and automated redeployment enables a scientific community to continue an uninterrupted usage of MetaStore. For enabling an uninterrupted update of MetaStore, we use a load balancer with an auxiliary instance of a web-server that is only used during the update process. The automated code generation and deployment process for a single active instance of MetaStore is described in Fig. 4. (1) After successful registration of the metadata schema, it is forwarded to the Metadata Code Generator component. (2) For the given metadata schema (namespace) and version, the Metadata Code Generator component automatically generates the software code

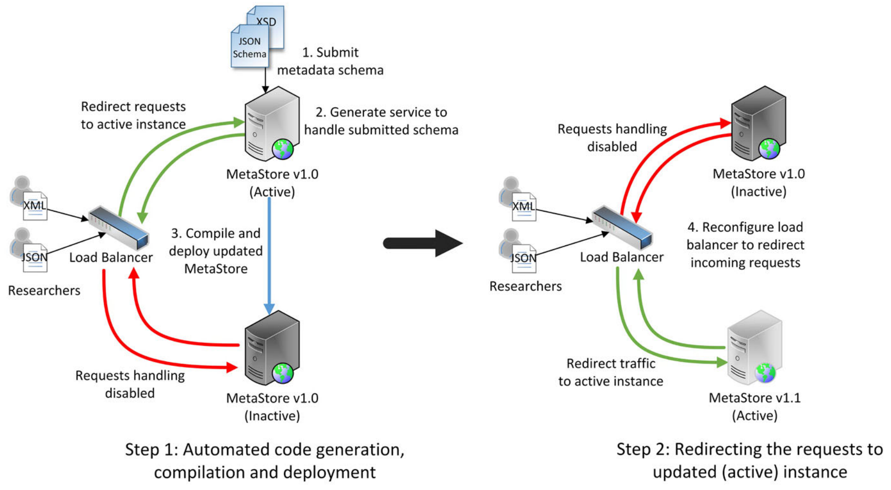

XSD
JSON Schema
1. Submit metadata schema
Redirect requests to active instance
2. Generate service to handle submitted schema
MetaStore v1.0 (Active)
XML
JSON
Load Balancer
Researchers
3. Compile and deploy updated MetaStore
Requests handling disabled
MetaStore v1.0 (Inactive)
Step 1: Automated code generation, compilation and deployment
Requests handling disabled
MetaStore v1.0 (Inactive)
4. Reconfigure load balancer to redirect incoming requests
XML
JSON
Load Balancer
Researchers
Redirect traffic to active instance
MetaStore v1.1 (Active)
Step 2: Redirecting the requests to updated (active) instance

Fig. 4 Automated code generation and deployment

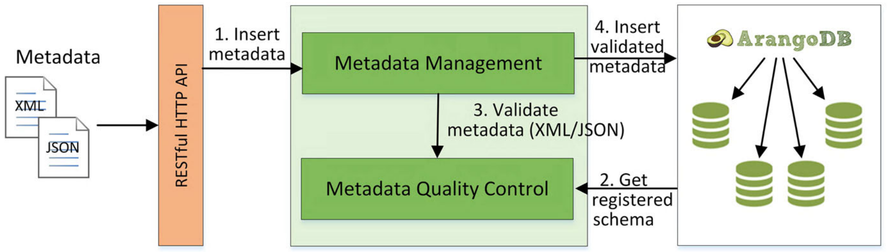


Fig. 5 Metadata management and quality control

(service) necessary for handling the metadata registered in the Metadata Registry. (3) The entire MetaStore with this new service is compiled and deployed on the auxiliary web-server. (4) The load balancer is configured to redirect incoming requests to the updated instance of MetaStore, simultaneously the old version of MetaStore is disabled after successful completion of the active requests.

3.2.3 Metadata management and quality control

The Metadata Management component exposes various CRUD (create, retrieve, update, delete) operations that are specific to the given NoSQL database (in this case ArangoDB and SPARQL queries for Apache Jena). The Metadata Management component adds the dynamically created services by the Metadata Code Generator to the pool of existing services. This component also acts as a bridge between the native ArangoDB libraries and the MetaStore REST services.[8] Additionally, the metadata is modeled in a community specified METS-profile, and stored with the data in a

[8] http://datamanager.kit.edu/masi/localizationmicroscopy/swagger-ui/.

Table 1 Vocabulary mapping between PREMIS and ProvONE

Rule. #	PREMIS	ProvONE	SKOS mapping
1	Event	ProcessExec	relatedMatch
2	Object	Data	narrowMatch
3	Agent	User	broadMatch
4	relatedObjectIdentification	wasDerivedFrom or hadMember	relatedMatch
5	linkingObjectIdentifier	used	relatedMatch
6	linkingAgentIdentifier	wasAttributedTo	broadMatch
7	relatedEventIdentification	wasGeneratedBy or used	broadMatch

scientific data repository. To fulfill the quality control requirement from the research communities, MetaStore supports two stages of quality control: (1) schema validation and well-formedness and (2) vocabulary based metadata control. The vocabulary based metadata quality control is explained in Sect. 3.3.2.

Figure 5 shows the default schema validation and well-formedness quality control process that is followed for each insertion of metadata. (1) Metadata is submitted by the scientific community in either XML/JSON format, or extracted from data and inserted through the REST API of MetaStore. (2) The Metadata Management Component analyzes the metadata to determine its corresponding schema and version from the key-value store. (3) Based on the corresponding version of the available schema, the Metadata Quality Control component performs a schema conformance and well-formedness check of the metadata. (4) The validated metadata is converted into JSON format and inserted into the designated collection.

3.2.4 Provenance manager

Provenance has a wide-range of applications. On the one hand, provenance can be used for determining the quality of the results and for analyzing and improving the quality of workflows, and on the other hand, provenance is critical for the long-term preservation of data. For efficiently handling provenance, as per its application, it is necessary to support the appropriate provenance model where required.

The Provenance Manager supports the handling of scientific workflow provenance based on the ProvONE model and provenance for long-term preservation of a digital resource using the PREMIS model. Comprehensive workflow provenance information consists of two parts, the workflow definition (prospective provenance) and the execution details (retrospective provenance) [38]. ProvONE is a provenance model that is capable of modeling both the prospective and retrospective provenance. For efficient storage and querying of the provenance information in the ProvONE model, the ProvONE provenance graphs are stored in the Apache Jena TDB. Various query patterns for retrieving the ProvONE provenance information are implemented and exposed as REST services.[8]

However, for allowing interoperability between these models, an exclusive mapping between ProvONE and PREMIS terms is unfeasible, because the ProvONE model is designed to support both prospective provenance and retrospective provenance, whereas the PREMIS standard is intended to model only retrospective provenance. Thus, to enable retrospective provenance interoperability between these models, the vocabulary mapping rules between the ProvONE (retrospective provenance) and the PREMIS model are described using the W3C SKOS mapping vocabulary specification. This mapping is an extension to an existing vocabulary mapping between the Open Provenance Model (OPM) [39] and PREMIS [40]. The mapping rules are presented in Table 1, and a brief explanation of each rule is as follows.

Rule 1 A PREMIS Event is mapped to ProvONE ProcessExec using the SKOS relatedMatch. Both the PREMIS Event and the ProvONE ProcessExec (linked to a Process class) represents an action performed on a data represented by the ProvONE Data or Collection or a PREMIS Object respectively.

Rule 2 A PREMIS Object is mapped to ProvONE Data using the SKOS narrowMatch because a PREMIS Object can only be of the type bitstream, file, or an aggregation, while ProvONE Data can be of any type.

Rule 3 A PREMIS Agent can be a either *software, person, or an organization*, and is mapped using SKOS broadMatch to ProvONE User that can represent only a *person*.

Rule 4 A PREMIS relatedObjectIdentification is mapped using the SKOS relatedMatch to ProvONE hadMember (structural relation) or ProvONE wasDerivedFrom (derivation relation).

Rule 5 A PREMIS linkingObjectIdentifier is mapped to ProvONE used class using the SKOS relatedMatch, as it represents a relation between a PREMIS Event and PREMIS Object, and ProvONE ProcessExec and ProvONE Data, respectively.

Rule 6 A PREMIS linkingAgentIdentifier is mapped using the SKOS broadMatch to ProvONE wasAttributedTo. In PREMIS a linkingAgentIdentifier represents a relationship with any of the Agents, whereas in ProvONE, the wasAttributedTo represents a relation only between a User and a Process that is associated with a ProcessExec.

Rule 7 A PREMIS relatedEventIdentification is mapped using SKOS broadMatch either to ProvONE wasGeneratedBy or used class, based on the PREMIS relationshipSubType.

3.2.5 OAI-PMH METS provider and harvester

A common requirement from the research communities is to have customizable metadata harvesting for sharing partial or entire collections of metadata. For this, we explicitly implemented the six verbs recommended by the OAI-PMH. OAI-PMH is the *de facto* standard for exporting metadata across scientific data repositories. By design, the basic interoperability using Dublin Core metadata standard [41] is supported by MetaStore. However, due to the limited expressiveness of the Dublin Core standard, MetaStore also supports a comprehensive metadata standard like METS. METS is a metadata container format comprising different sections that allow encoding of administrative ⟨amdSec⟩, structural ⟨fileSec⟩, ⟨structMap⟩, ⟨structLink⟩ descriptive

⟨dmdSec⟩ and provenance ⟨digiprovMD⟩ metadata. For example, to support harvesting of nanoscopy metadata, the nanoscopy METS-profile[4] is provided.

The primary design consideration behind this approach is to keep the architecture design of MetaStore simple and avoid any dependency to an external OAI-PMH implementation. Adopting the ArangoDB Query Language (AQL) has the following benefits: (a) It provides the flexibility to define and implement precise queries required for metadata harvesting. These queries are exposed through a dedicated REST interface for enabling seamless integration with other systems. (b) With the metadata harvesting implemented on the primary metadata database, the cost and effort of maintaining and synchronizing an auxiliary OAI data provider server are avoided.

In the following, we briefly describe the metadata harvesting process. The OAI-PMH METS Harvester component retrieves the administrative, descriptive and structural metadata from the document store of ArangoDB and assembles it in the ⟨amdSec⟩, ⟨dmdSec⟩, ⟨structMap⟩, ⟨structLink⟩ section of the community specified METS-profile. The retrospective provenance metadata is queried from the Apache Jena TDB and based on the vocabulary mappings shown in Table 1; the retrospective provenance is translated in the PREMIS standard and assembled in the ⟨digiprovMD⟩ section of the METS-profile. Thus, the entire metadata stored in ArangoDB and Apache Jena TDB acts as the default OAI-PMH data provider for harvesting the complete metadata. The OAI-PMH specified six verbs are implemented as REST services[8] based on the following core AQL queries:

GetRecord This verb retrieves a specific record from the OAI-PMH repository. The required arguments are an *identifier* associated with a record and the *metadataPrefix* specifying the metadata format to be retrieved. The query iterates over all the documents in the *OAIPMH_Repository* and returns a single record based on the filter arguments.

```
FOR Metadata IN OAIPMH_Repository
FILTER Metadata._key=='<identifier>' AND
Metadata.prefix=='<metadataPrefix>'
RETURN Metadata
```

Identify This verb retrieves the administrative information describing the underlying OAI-PMH repository. The query retrieves the administrative information, *AdminInfo* from the *OAIPMH_Repository*.

```
FOR AdminInfo IN OAIPMH_Repository FILTER
AdminInfo._key=='Identify'
RETURN AdminInfo
```

ListIdentifiers This verb returns a list of headers, containing a unique identifier of every record. The required argument is the *metadataPrefix* specifying the metadata format. Optional arguments for selective harvesting based on *datestamp* or *set* membership are also allowed. The query iterates over all the documents in *OAIPMH_Repository* and returns for the given *metadataPrefix* the list of header information of all the records.

```
FOR Metadata IN OAIPMH_Repository FILTER
Metadata.prefix=='<metadataPrefix>'
RETURN {"identifier":Metadata._key}
```

ListMetadataFormats The verb is used to retrieve the metadata formats supported by the OAI-PMH repository. The query fetches the administrative information *AdminInfo* from the *OAIPMH_Repository* to return a list of supported formats.

```
FOR AdminInfo IN OAIPMH_Repository FILTER
Admin_Data._key=='ListFormats'
RETURN AdminInfo
```

ListRecords The verb is used to retrieve all the records from the OAI-PMH repository. The required argument for this verb is *metadataPrefix*. Optional arguments allow selective harvesting based on *set* membership or *datestamp*. The query iterates over all the documents and returns all the records.

```
FOR Metadata IN OAIPMH_Repository FILTER
Metadata.metadataPrefix=='<metadataPrefix>'
RETURN Metadata
```

ListSets This verb is used to retrieve all the *set* structures supported by OAI-PMH repository. This command is useful for selective harvesting.

```
FOR AdminInfo IN OAIPMH_Repository FILTER
AdminInfo._key=='ListSets'
RETURN AdminInfo
```

3.2.6 Metadata recovery engine

The Metadata Recovery Engine performs the restoration of the complete metadata storage in case of a database failure. The Metadata Recovery Engine collects all the METS files from the scientific data repository, where each file is decomposed according to the various METS sections. For example, the descriptive metadata from the ⟨dmdSec⟩ section is extracted and with prior schema registration and validation inserted into the document store. The provenance metadata comprising the workflow definition in XML and the PREMIS retrospective provenance from the ⟨digiprovMD⟩ section is extracted, transformed based on mapping shown in Table 1 into the ProvONE model and stored in the Apache Jena TDB. The metadata recovery process is based on the combination of the processes shown in Figs. 3 and 5.

3.3 MetaStore extension layer

The MetaStore Extension Layer allows the integration of various third-party tools and technologies with the MetaStore Core Layer. The primary aim of the MetaStore Extension Layer is to support reuse of existing tools, software libraries, web-services, or databases in MetaStore. For example, to enable handling of annotations, the Anno4j library [42] providing an implementation of WADM specification is integrated through the extension layer of MetaStore.

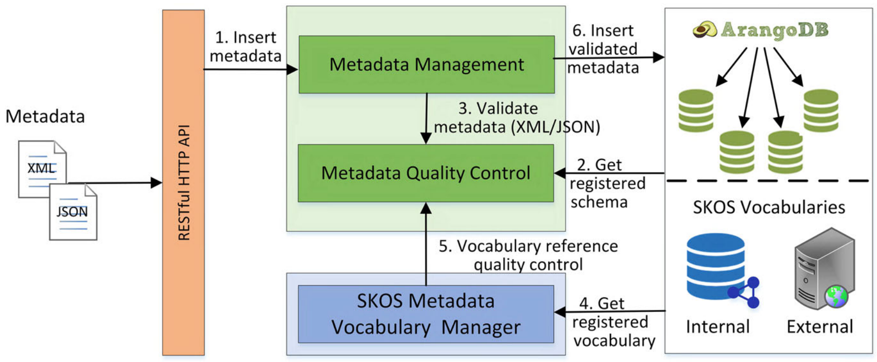

Metadata
XML
JSON
RESTful HTTP API
1. Insert metadata
Metadata Management
3. Validate metadata (XML/JSON)
Metadata Quality Control
5. Vocabulary reference quality control
SKOS Metadata Vocabulary Manager
6. Insert validated metadata
2. Get registered schema
4. Get registered vocabulary
ArangoDB
SKOS Vocabularies
Internal
External

Fig. 6 Vocabulary supported metadata quality control

3.3.1 Annotation manager (WADM)

For enriching the quality of the results and imparting additional knowledge about the data, it is necessary for domain experts to associate additional descriptions in the form of annotations with the datasets. Moreover, when the datasets evolve, there are new insights about the data, which require the annotations to be updated. For example, the *text segmentation* step of the eCodicology *Layout feature extraction* workflow (see Fig. 14) currently implements the Trainable Weka Segmentation.[9] However, with the modification of the workflow to support projection-based segmentation, the annotations (layout features) generated for the same input dataset will be different. As these annotations are subject to frequent changes, we consider them as dynamic metadata, and to model this dynamic metadata we adopt the WADM. For enabling annotations of images and text, we integrate two annotation frameworks namely CodiLab for annotating medieval manuscripts from humanities projects and Annotorious JavaScript plugin of the Annotator library for the nanoscopy images.

Currently, Annotorious JavaScript API provides limited functionality, as it does not support the modeling of the annotations in the W3C recommended WADM. Moreover, the storage systems supported by the Annotorious JavaScript API are ElasticSearch or Parse Storage. Thus, we extend the modeling and storing of annotations by integration the Anno4j library that provides an implementation of WADM, and integrate the Apache Jena framework for persisting the annotations and allowing querying using SPARQL. The annotation manager module of the MetaStore is available in GitHub.[10]

3.3.2 SKOS metadata vocabulary manager

For enabling an in-depth vocabulary-based metadata quality control, research communities can extend MetaStore with the SKOS Metadata Vocabulary Manager. Currently, MetaStore offers two approaches for integrating SKOS-based controlled vocabular-

9 http://imagej.net/Trainable_Weka_Segmentation.

10 https://github.com/svaibhav19/Thesis/tree/master/.

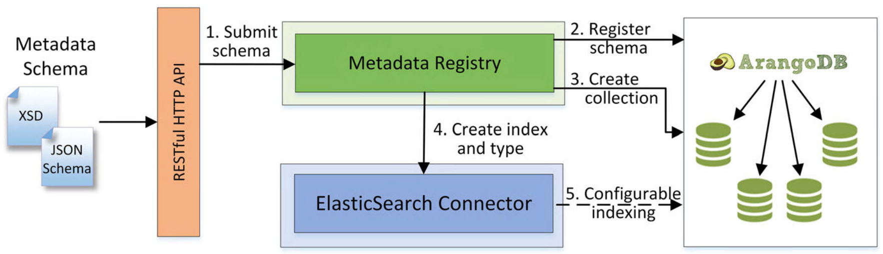


Fig. 7 Metadata registration and ElasticSearch indexing

ies. In the first approach, research communities can import their existing vocabularies in the Apache Jena TDB, which are modeled as SKOS vocabularies and maintained by MetaStore. During the metadata registration these vocabularies can be linked to their corresponding metadata schemas. This linking is necessary for enabling automated vocabulary-based quality control. The access to these imported vocabularies is provisioned through the integration of the web-based Skosmos tool [43] that offers a REST API.[11] Skosmos exposes a generic REST API for retrieving the vocabularies. In the second approach, MetaStore allows configuring the available metadata vocabulary providers (externally available vocabularies). The REST endpoints for accessing the external vocabularies are linked to their corresponding metadata schemas for enabling vocabulary-based metadata quality control.

For example, in the Medieval Stained Glass Church Windows use case, i.e., the “Corpus Vitrearum Deutschland” project [44,45], the controlled-vocabularies are modeled in the ICONCLASS [46] standard and exposed through the Skosmos API. For this use case, we follow the first approach described above, i.e., we have imported the ICONCLASS controlled-vocabularies from their vocabulary server into the MetaStore Apache Jena TDB. These vocabularies are linked to the different fields in their metadata schema, and during the metadata quality control process the metadata is validated against these controlled-vocabularies.

The automated vocabulary-based metadata quality control process, as shown in Fig. 6, is an extension to the default metadata quality control process explained in Sect. 3.2.3. The default quality control process is extended with two additional steps that are introduced after step (3) Validate metadata (XML/JSON). Following are the additional steps: (4) For validating the metadata, each element is verified with its corresponding vocabulary. The vocabularies are retrieved either from MetaStore’s Apache Jena exposing the Skosmos REST-API or from the configured external services that expose their discipline-specific vocabularies. (5) Based on the retrieved sets of vocabularies, the Metadata Quality Control component validates the metadata. Finally, as explained in step (4) of Fig. 5, the metadata is converted to JSON format and inserted into the designated collection.

[11] http://api.finto.fi/doc/.

3.3.3 ElasticSearch connector

As a core functionality, MetaStore automatically indexes the complete schema in the database for enabling full-text search over the metadata. However, for allowing a richer full-text search with the possibility of performing faceted and fuzzy search, MetaStore also provides integration with ElasticSearch. Research communities can configure MetaStore to index the complete metadata or a part of the metadata in ElasticSearch. However, enabling this configuration will change the default behavior of MetaStore, i.e., the automated indexing of metadata in the primary database (ArangoDB) will be disabled, and the full-text search queries will be redirected to the metadata indexed in ElasticSearch. Ideally, the configuration should be done during the registration of the metadata schema or the community defined METS-profile. For example, in the case of a nanoscopy METS file, the descriptive metadata contains the information necessary for performing full-text search, thus, only the metadata defined in the ⟨dmdSec⟩ section of METS is indexed in ElasticSearch.

For an ElasticSearch enabled configuration of MetaStore, the metadata schema registration is followed by the index and type creation in ElasticSearch, as illustrated in Fig. 7. The initial process of metadata schema registration is similar to the one shown in Fig. 3. The process comprises four primary steps and one optional step. Steps (1) Submit schema, (2) Register schema, and (3) Create collection, are the same as described in Sect. 3.2.1. (4) For enabling full-text search based on ElasticSearch, the corresponding index and a type for an individual registered metadata schema are created in ElasticSearch. If a METS-profile is registered, an index is created with multiple types corresponding to the different schemas embedded within METS. It should be noted that with enabling of ElasticSearch in MetaStore, the automated index creation in ArangoDB will be disabled. (5) The configurable indexing is an optional step that allows indexing of the existing metadata from ArangoDB in ElasticSearch at any given point in time. This change does not affect the functioning of MetaStore, as the existing metadata from ArangoDB will be indexed in ElasticSearch, and full-text searches will be redirected from ArangoDB to ElasticSearch.

3.3.4 RDA metadata directory registry

Research Data Alliance (RDA)[12] is a research community that focuses on building social and technical infrastructures for open sharing of data. The working groups within the RDA regularly contribute recommendations and tools for handling the different aspects involved in a life-cycle of scientific data. Two such working groups in RDA are the Metadata Standards Directory Working Group (MSDWG) and the Metadata Standards Catalog Working Group (MSCWG). The MSDWG has implemented a publicly available metadata registry for collecting a wide range of metadata standards[13] from different disciplines [47], and the MSCWG is currently working on

[12] https://www.rd-alliance.org/.

[13] https://github.com/rd-alliance/metadata-directory.

building a metadata catalog[14] based on this metadata registry. The metadata catalog aims at providing an interface for allowing querying and retrieving of the metadata standards.

Currently, the MSDWG registry does not provide the functionalities necessary for managing metadata but contains an exhaustive list of metadata standards. Thus, integrating the MSDWG registry will leverage the metadata models currently supported by our MetaStore framework. Moreover, with the functionalities provided by MetaStore, the handling of the various metadata standards registered in the MSDWG can be completely automated. Each metadata standard in the MSDWG registry is described using a YAML [48] template, and serialized in a ".md" file. The RDA Metadata Directory Registry component in the MetaStore Extension Layer is a bridge between the MSDWG registry and the MetaStore framework. The RDA Metadata Directory Registry component retrieves the metadata standard (.md file) from the MSDWG registry and submits it to the Metadata Registry of MetaStore. Based on this ".md" file the metadata registration, indexing and code generation processes described in Sects. 3.2.1 and 3.2.2 are executed. Thus, the RDA Metadata Directory Registry component enables a seamless integration for handling the metadata standards registered in the MSDWG registry.

3.4 Scientific data repository

In principle, a scientific data repository is a data storage entity that offers various low-level services for storing (long-term archival), converting (format conversion), and transferring scientific data. For example, the Nanoscopy Open Reference Data Repository (NORDR) is a scientific data repository for storing experiment data. Basic administrative metadata for identifying the data is maintained by NORDR. Currently, NORDR provides many low-level services for handling the data in the repository. Following low-level services are available in NORDR: (a) automatically assigning a Persistence identifier (PID) to a dataset, (b) generating preservation metadata necessary for long-term archival, and (c) implementing the different data transfer protocols (GridFTP [49] and WebDAV [50]) for allowing transfer of data between NORDR and a high-performance computing cluster. Moreover, the various data processing services necessary for composing a workflow are registered in NORDR and deployed on a high-performance computing cluster.

4 Related work

Research in metadata has advanced multifold, especially in various disciplines under the broad research area of e-Science. In this section, we describe the available metadata management systems by categorizing them under the following four categories:

[14] https://github.com/rd-alliance/metadata-catalog-dev.

Metadata management in scientific data repositories

A critical aspect of a Scientific Data Repository (SDR) is its ability to handle the metadata that is associated with the data stored in the repository. The majority of the operations within a SDR are directly dependent on the metadata, for example, dataset registration (administrative metadata), long-term data preservation (preservation metadata), data discovery (descriptive metadata), user-rights and access control (rights metadata), data views and organization (structural metadata), and quality and lineage (provenance) are the operations that are entirely controlled by metadata.

The overall research area of SDRs can be categorized in two types: (a) community-specific or community-driven SDRs and (b) generalist SDRs. The community-specific SDRs are not considered in this paper due to the following two reasons. First, the community-specific SDRs are designed for handling only specific metadata standards that are necessary for a given research discipline, whereas the core focus of this paper is to design an entirely generic metadata management system that can be reused by multiple research communities. Second, currently, there are around hundred community-specific SDRs, and there already exist in-depth surveys elaborating on the metadata management capabilities of these SDRs [51–53].

DSpace is an open source digital repository system for preserving digital data (image, video and text datasets) [54]. By design, DSpace supports the Dublin Core standard and allows metadata harvesting through the OAI-PMH. It is also possible to use a few hierarchical metadata standards such as MARC and MODS in DSpace, but this can be achieved only with the help of external tools. DSpace supports simple non-hierarchical metadata standards (i.e., basically a list of key-value pairs) for modeling user-defined metadata. For allowing full-text search over metadata, DSpace provides the possibility to manually select the metadata fields that are to be indexed. For storing the metadata, DSpace can be deployed with either a PostgreSQL or an Oracle database.

Archivematica [55] is a digital repository system that allows archival of and access to digital objects. Archivematica supports the PREMIS and the Dublin Core metadata standards, with integration into Elasticsearch for enabling data search and discovery over metadata. For data preservation purposes, the PREMIS preservation standard within METS is supported.

The ICAT [56] project supports large facility experiment data and is based on the Core Scientific Metadata Model (CSMD) [57] that is realized in a relational database (MySQL). The ICAT server provides various SOAP and RESTful service interfaces to the underlying database. For allowing full-text search, some of the information from the MySQL database is indexed in Apache Lucene.

The European Data Infrastructure (EUDAT) is a Pan-European project that aims at providing common data services and a collaborative research environment for multiple research communities [58]. Of the various common data services, the metadata related services are B2SHARE and B2FIND. The B2SHARE service provides ingesting, storing, preserving and sharing of data. For enabling full-text search over metadata, the metadata fields have to be filled in manually during the ingest process. B2FIND uses SOLR indexing, wherein the metadata is harvested using the OAI-PMH protocol from various metadata providers and indexed in SOLR [59].

EPrints is designed specifically for archiving research papers, theses, and teaching material, however, it is also possible to store data (files) in the EPrints repository

[60]. Similar to DSpace, EPrints provides support for the Dublin Core standard, with the possibility to export metadata in the METS standard. For allowing full-text search over metadata, extra effort is required for integrating the Xapian[15] engine with EPrints. However, the full-text search is only available for a few data formats (PDF, Word, and HTML).

Standalone metadata management systems

Currently, there exist few systems that provide a generic and reusable metadata management solution. In the following, we survey these systems.

The XMC Cat metadata catalog for the LEAD cyberinfrastructure [61] follows a hybrid XML/relational approach that stores XML metadata as a Character Large Object (CLOB) and further shreds XML using inlining [62] and stores it in a relational database schema to enable execution of complex queries.

Metacat is an open source metadata catalog and data repository that aims at catering the metadata needs of the National Center for Ecological Analysis and Synthesis (NCEAS) [63]. The Metacat framework uses a hybrid storage approach wherein it extracts, models, and stores the metadata from XML in a relational database schema that conforms to XML format. For retrieving the metadata, the Metacat provides a set of SQL queries.

The DIstributed MEtadata Server (DIMES) is a flexible metadata management server that is primarily designed for handling the metadata of the Earth Science research community [64]. In principal, the DIMES is based on the concept of representing an XML document containing the metadata in its natural Document Object Model (DOM) tree structure. For querying purposes, an XML query engine based on the XML4J package is integrated into DIMES that provides the traversing over an XML DOM tree structure. For retrieving the metadata, basic queries, nearest neighbor queries, and tree expand queries are supported.

Metadata management in grid infrastructures

In the research area of distributed computing, there exist multiple systems for handling metadata in the Grid environment. In the following, we describe these systems with their features for handling scientific metadata.

The Storage Resource Broker (SRB) of the San Diego Supercomputer Center is a middleware that provides an API for accessing heterogeneous distributed storage resources [65]. For supporting attribute-based access of the data, the SRB employs a Metadata Catalog Service (MCAT) [66]. The MCAT metadata schema for modeling descriptive and system related metadata is similar to the Dublin Core metadata standard. This schema is modeled in a relational database (DB2) and a set of APIs are provided for querying and updating the metadata.

The Metadata Catalog Service (MCS) is a Grid-based metadata service that is designed for handling the metadata generated in the Grid environment [67]. Principally, MCS provides functionalities for storing, accessing, and querying descriptive metadata based on user defined attributes. The MCS schema is an extension of the MCAT schema of the SRB middleware, and is implemented in a relational database (MySQL). For systematically organizing the metadata, the MCS metadata schema is divided in the

[15] https://xapian.org/.

following logical categories: *logical file metadata, collection metadata, view metadata, authorization, user, audit, provenance metadata, and user-defined and annotation metadata*. Out of these logical categories, the *user-defined* and *annotation* metadata allow the MCS schema to be extended beyond its default attributes. As a generic solution for handling community-specific metadata, MCS provides two tables, one that contains the common attributes and an extension table with a set of predefined attribute types (Integer, String, Float, Date, Time, DateTime) for storing additional community-specific attributes. The extension table contains three columns (object id, attribute name, and attribute value). Each entry in this table has a reference to the object id in the common table with an attributed name and value (basically a key-value pair). The annotation service of MCS allows users to create a key-value pair of the string type that can be associated with an object of type *logical file, collection or view*.

The Science Object Linking and Embedding (SOLE) tool is specifically designed for linking research papers with science objects for making research data reproducible [68]. In SOLE, a science object can be *language objects* (*source code*)*, annotated PDFs and datasets, web-services, or virtual images*. For retrieving the data analysis pipelines, SOLE offers users to link the results presented in a research paper with the workflows that were used to derive them. Metadata in the form of tagged annotations from PDFs are automatically extracted and stored in the SOLE database, whereas datasets can be manually tagged with annotations for imparting additional description.

Commercial metadata management systems

With the wide-spread realization of the importance of metadata, a few commercial solutions are available that offer comprehensive metadata management capabilities. In the following, we briefly describe these solutions.

Stardog[16] is an RDF database for handling enterprise data using an enterprise knowledge graph. It provides semantic tools based on OWL 2 ontologies, with SPARQL support. For allowing full-text search, the data is indexed in Apache Lucene [69]. For querying legacy systems through SPARQL, Stardog provides a Virtual Graph that allows mapping between the Stardog graph and external data sources. For enabling data quality control, Stardog uses Integrity Constraint Validation (ICV) that allows an explicit definition of data rules (constraints). Moreover, Stardog supports handling of database revision history using the PROV model [70], and domain-specific controlled vocabularies can be integrated using the SKOS specification.

PoolParty is another commercial semantic technology platform that supports an enterprise in organizing enterprise knowledge with data analytics features [71]. Similar to the Stardog system, the PoolParty platform is natively designed on an RDF database and primarily offers the following features: (a) The Thesaurus Server provides the typical CRUD operations for handling domain-specific taxonomies and thesauri based on SKOS, wherein the SKOS taxonomies can be enriched with ontologies (for example, the Friend of a Friend (FOAF)[17] ontology). (b) Users can create custom schemas from the existing ontologies using the ontology and schema editor service. (c) Metadata vocabulary quality can be measured using the qSKOS specification. (d) The Graph

[16] http://www.stardog.com/.

[17] http://xmlns.com/foaf/spec/.

Search Server collects data from the various PoolParty services and transforms it into RDF, which is indexed in either Apache SOLR or Elastic Search for allowing full-text search. (e) The Extractor service analyses documents and texts, with the metadata schemas that are mapped to predefined SKOS thesauruses, and automatically extracts meaningful phrases, named entities, or other relevant entities.

In Sect. 5.1, we analyze the metadata management systems mentioned above and compare their features with the features of the MetaStore framework.

5 Evaluation

In this section, first, we present a comparison of the features of existing metadata management systems with those of MetaStore. Second, we describe the performance of MetaStore for two NoSQL database systems, followed by read and write performance evaluations of MetaStore with XMC Cat metadata catalog, MCS query interface, and the Metacat catalog system.

5.1 Evaluation of features

In Sect. 4, we surveyed several metadata management systems from diverse areas of research. In this section we present a feature-based comparison of the available metadata systems with MetaStore.

A common architectural design limitation of the metadata management systems in SDRs is that in these systems, the database schema is designed based on a specific metadata model, with interfaces (functionality) that are tightly coupled to this model. For the community, it is an additional effort in either adapting or translating their existing metadata schema to the one supported by the SDR. Such conversions between metadata schemas often lead to a lossy transformation, wherein some of the schema attributes have to be either discarded or semantically modified. To avoid any schema mappings and conversions, MetaStore is designed independent of any specific metadata schema, and there are no limitations on metadata schemas that MetaStore can support, as long as the metadata schema can be serialized in XML or JSON. Furthermore, as the architectural design of these systems is based on relational databases, it inherently limits the extensibility in handling new metadata schemas. This is because, to handle a new metadata schema the relational schema needs to be modified, the services (functionality) build over this schema need to be updated, and the entire system needs to be upgraded considering the backward compatibility (i.e., the existing functionality should also be supported). In MetaStore, we overcome this design limitation by integrating a NoSQL database that allows handling of ad hoc metadata schemas. With the Metadata Code Generator component, the services necessary for handling the metadata schema are created on-the-fly and are added to the architecture without modifying the existing ones. Regarding OAI-compliance, many of the SDRs support metadata harvesting through OAI-PMH. However, the workflow provenance and annotations using interoperable standards (ProvONE and WADM) are not supported, whereas MetaStore is designed OAI-compliant and also supports the ProvONE provenance model and the WADM for provenance and annotation interoperability.

Comparing the features of the standalone metadata management systems and the Grid-based metadata systems with MetaStore, XMC Cat, Metacat, and DIMES are based on an XML database or a hybrid of XML and SQL databases. On the one hand, all XML-based metadata schemas can be supported by these systems, but on the other hand, in terms of database scalability, handling increasing metadata volumes is a major concern. In terms of metadata quality control, these systems do not implement any quality control mechanism. Moreover, integration of controlled vocabularies for verifying the metadata content with automated rectification is not supported by these systems. MetaStore offers default quality checks for schema conformance and well-formedness, and advanced quality control for metadata content validation based on SKOS-based domain-specific controlled vocabularies.

A critical deficiency among these systems is that they are not OAI-compliant, i.e., large-scale metadata harvesting through standard harvesting protocols like OAI-PMH is not supported. MetaStore is designed OAI-compliant, and the OAI-PMH data provided for allowing metadata harvesting is implemented over the primary metadata storage (ArangoDB). In terms of data reproducibility and long-term data preservation, most of these systems do not support handling of provenance metadata. The MCS for a Grid environment provides the *Creation and Transformation history* metadata attribute that allows a textual description of the data transformation process, but this is an MCS-specific attribute that is not compliant with existing provenance models (OPM, PROV, ProvONE). MetaStore supports two provenance models PREMIS and ProvONE, each for a specific purpose. With PREMIS the provenance required for long-term preservation is handled, and as ProvONE allows modeling of a workflow definition and the runtime provenance, the routine activities such as querying, analyzing and improving workflows can be performed. The MCS schema is an extension of the MCAT metadata management system that provides a special extension table (dynamic metadata) with six attribute types for modeling community-defined metadata schemas as non-hierarchical key-value pairs. However, this approach has multiple limitations: (a) hierarchical metadata schemas with embedded data structures cannot be stored in the MCS schema. (b) the MCS schema can not handle complex attribute types such as arrays, geospatial information, and time-series data. (c) as the extension attributes are stored in a single table, it is evident and also mentioned by the authors that with increasing data volumes the query performance is expected to decline. We overcome these limitations by designing MetaStore based on a NoSQL database. This allows us to store any type of metadata model (hierarchical, flat key-value pairs) with a wide-range of data types.

Regarding the features of the commercial metadata management systems Stardog and PoolParty, both systems are primarily designed for handling enterprise knowledge data. As both systems are closed-source, enriching these systems with community driven extensions is a challenging task. On the one hand, as these systems are designed to the RDF specification, it is possible to handle dynamic metadata in the form of annotations, but, on the other hand, annotation interoperability through the W3C recommended WADM is not supported. Also, these systems do not provide large-scale metadata harvesting through the OAI-PMH specification. Stardog provides traces of a database revision history using the PROV model, whereas for scientific communities it is necessary to capture the workflow description and provenance for enabling data reproducibility. Moreover, for long-term preservation of the data the support towards the appropriate metadata standard (PREMIS) is not compatible with these systems.

We also evaluated MetaStore with the Big Data processing frameworks like LOOM [72] and Apache Falcon.[18] In principal, these frameworks focus on enabling efficient data processing in a distributed environment. The LOOM framework aims at providing in-memory optimization of aggregations within a big data analysis framework. Similar to the MapReduce processing data model, LOOM provides two-phased computation. In the first phase, the individual datasets are processed, followed by consolidation of these results in the second phase. Apache Falcon is a feed and process management framework built on Hadoop that primarily abstracts and automates the redundant tasks involved in data processing pipelines. Comparing these systems to MetaStore, the current version of MetaStore focuses on managing heterogeneous metadata models with support for handling large metadata volumes. However, the LOOM and Apache Falcon frameworks are each a potential extension to the MetaStore framework when large-scale metadata processing is required.

Considering the metadata management approach based on the classification of metadata as per its use, Deelman et al. [73] propose the logical organization of metadata in different layers. In their paper, they claim that by organizing the metadata in layers, it is possible to distinguish the metadata and the source or applications for which the metadata is relevant. For example, the primary layer handles the metadata of the raw datasets, and the secondary and tertiary layers are responsible for metadata describing the process of obtaining derived data and the metadata of the derived data. Moreover, the layered metadata organization allows users to expose the appropriate granularity of detail to the user, with the possibility to track layer-based usage patterns. However, irrespective of the layers in which metadata is classified, the fundamental characteristic of metadata is the underlying model or schema to which it conforms. Thus, as MetaStore is entirely driven by the metadata schema, metadata from any layer can be handled in MetaStore. Moreover, for comprehensively aggregating heterogeneous metadata models from different layers, we use the METS model. An overview of the feature comparison between existing metadata management systems and MetaStore is shown in Table 2.

5.2 Performance evaluation

In this section, we present the performance evaluation of MetaStore and compare it with the existing metadata management systems. For demonstrating the extensibility of MetaStore, we integrated MetaStore with both ArangoDB and MongoDB, and the read and write performance are evaluated. MetaStore is not integrated with an SQL database due to the primary requirement of having a flexible data model in modeling heterogeneous metadata models. Moreover, in terms of query performance, there exist few studies that provide an in-depth performance analysis between NoSQL and SQL databases [74,75].

[18] https://falcon.apache.org/.

Table 2 Features comparison of several metadata management systems with MetaStore

Metadata system	Schema flexibility	Supported schemas	Annotation support	Provenance support	Full-text search	OAI support	Open source	Quality check
DSpace	✗	DC, MARC, MODS	✗	✓(DC extension)	Indexing of DC terms	✓	✓	✗
Archivematica	✗	DC, METS, PREMIS	✗	✓(PREMIS)	✓(Elastic Search)	✓	✓	✗
Digital Commons	✗	DC	✗	✗	✓(Lucene)	✓	✗	✗
Hydra	✗	DC, PREMIS, METS	✗	✓(PREMIS)	✓(SOLR)	✓	✓	✗
ICAT Server	✗	CSMD	✗	✗	✓(Lucene)	✓	✓	✗
EUDAT	✗	ISO 19115, MARC, DC, CMDI, DDI	✓	✗	✓	✓	✓	✗
EPrints	✗	DC	✗	✗	Partial (PDF, Word, HTML)	✓	✓	✗
XMC Cat	✓	Any XML schema	✗	✗	✗	✗	✓	✗
Metacat	✓	Any XML schema	✗	✗	Partial	✗	✓	✗
DIMES	✓	Any XML schema	✗	✗	✗	✗	✓	✗
SRB MCAT	✗	Proprietary model	✗	✗	Partial	✓	✓	✗
MCS	✗	Proprietary model	✗	✓(Proprietary model)	Partial	✓	✓	✗
SOLE	✗	Proprietary schema	✓	✗	✗	✗	✓	✗
Stardog	✓	All RDF serialized schema	✗	✓(transaction provenance)	✓(Apache Lucene)	✗	✗	✓
PoolParty	✓	All RDF serialized schema	✗	✗	✓(Apache Lucene, SOLR)	✗	✗	✓
MetaStore	✓	Any XML & WADM	✓	✓(ProvONE)	✓(NoSQL and ElasticSearch)	✓	✓	✓

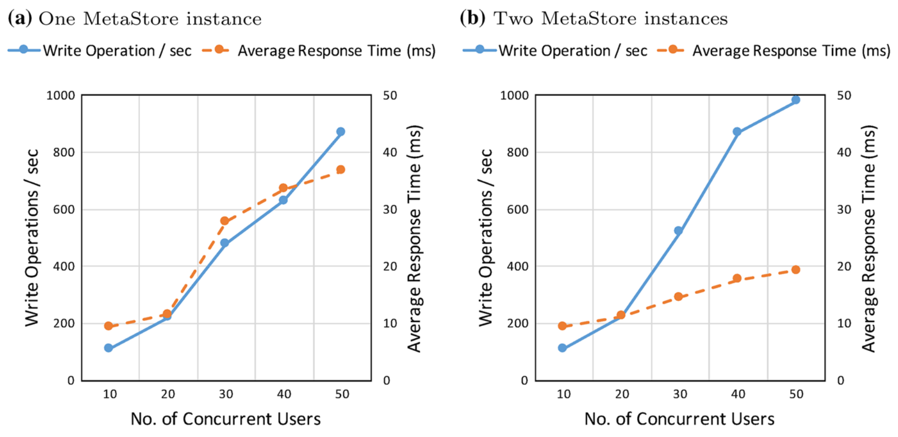


Fig. 8 MetaStore write performance (ArangoDB)

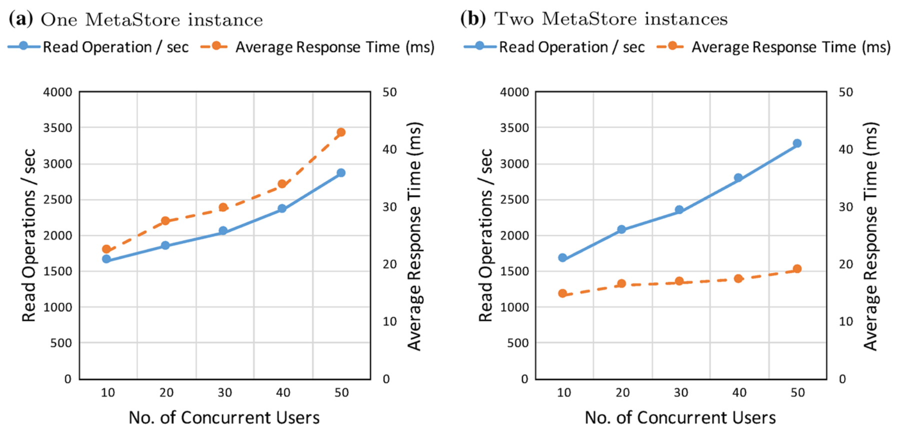


Fig. 9 MetaStore read performance (ArangoDB)

The MetaStore framework is evaluated considering a typical research community usage pattern (more read and fewer write operations) and the performance is measured in terms of the average response time for each operation. For evaluating the performance, various configurations of MetaStore were assessed. Each instance of MetaStore is deployed on a 32 GB RAM server with a 2.66 GHz (8 cores) processor, and each configuration is setup with one Apache2 load balancer on a 32 GB RAM server with a 2.66 GHz (8 cores) processor.

Following cluster configuration is deployed for ArangoDB. A single instance of ArangoDB coordinator server is deployed on a 128 GB RAM server with a 3.5 GHz (6 cores) processor, and four Arango database servers (shards) were deployed on a 16 GB RAM server with a 2.3 GHz (2 cores) processor each. In the case of the MongoDB cluster, three MongoDB query routers and config servers were deployed on a 8 GB RAM server with a 2.3 GHz (2 cores) processor. For storing the data, four MongoDB

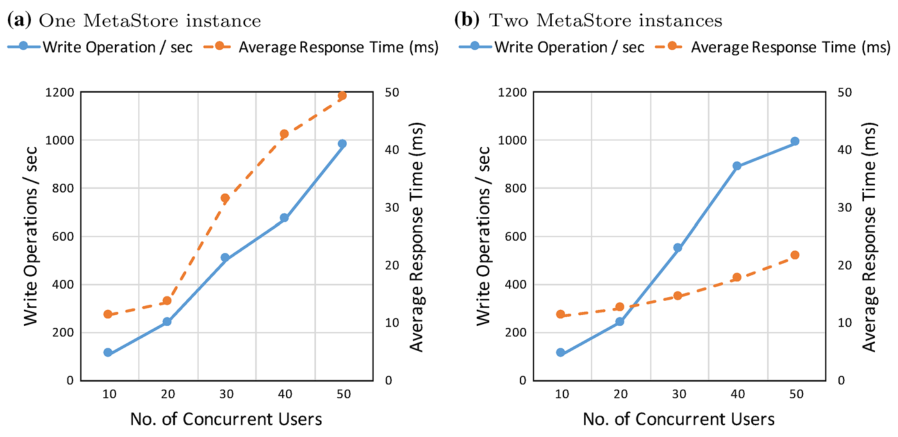


Fig. 10 MetaStore write performance (MongoDB)

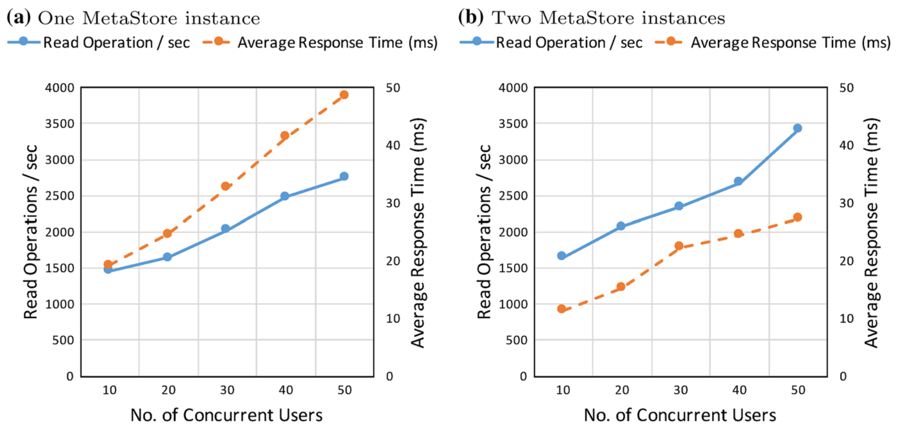


Fig. 11 MetaStore read performance (MongoDB)

data nodes (shards) were deployed on 16 GB servers with a 2.3 GHz (2 cores) processor each.

For efficiently handling large numbers of concurrent requests, we used the Apache server with the *mod_jk* module for load balancing the MetaStore instances deployed on independent tomcat servers, wherein the `maxThreads` parameter in the *AJP 1.3* configuration for each tomcat server is increased to 1024 for handling a larger number of concurrent requests. We also optimized the load balancer to handle a higher number of requests per second by enabling the Apache Multi-Processing Module (MPM)[19] worker. It is possible to further tune the load balancer performance by increasing the `maxRequestWorkers` parameter's default value of 256, but an inappropriately large value might bring the server to a standstill due to an influx of requests. Thus, for a production environment we set this parameter to its default value. Each test is

19 https://httpd.apache.org/docs/2.4/mpm.html.

executed three times. Due to negligible variations in each result, the error bars are not shown, and only the average results are depicted. Figures 8, 9, 10, and 11 show the write and read results.

Write performance Figures 8 and 10 illustrate the results of the write performance for one and two instances of MetaStore integrated with AranogDB and MongoDB, respectively. Each write operation carries a payload ranging between 20 and 30 KB and follows the process described in Fig. 5. For a single instance of MetaStore deployed on ArangoDB and MongoDB, we noticed that with increasing concurrent users, each generating multiple write operations, the average response time increased drastically, and some write operations failed to complete. This increase in average response time is due to the concurrent request handling limitation of each tomcat server. Moreover, each write operation comprises the quality control process, wherein an additional request is sent to retrieve the corresponding schema, followed by well-formedness and schema conformance checks.

With the introduction of a second instance of MetaStore either deployed on ArangoDB or MongoDB, we observe that due to the systematic distribution of the write operations between the two MetaStore instances, the average response time is retained below 22 ms, and not a single write operation failed.

For comparing the write performance of MetaStore with the MCS query interface, first we need to understand how metadata is stored in the MCS systems. Each add operation in the MCS system creates a logical file with multiple user-defined attributes in the database (maximum of ten attributes). The add operation is implemented as a native Java API and as a web service interface. Using the native Java API connected to a MySQL database, a maximum of 360 writes per second was achieved with 25 concurrent threads. For the web interface, a maximum of 70 add operations was possible for 25 concurrent threads. For multiple clients with four concurrent threads each, approximately 380 add operations was achieved, whereas for the web interface, a maximum of 80 add operations was achieved.

The LEAD infrastructure's XMC Cat metadata catalog based on hybrid XML/relational approach was evaluated with a native XML database, Oracle's Berkley DB XML.[20] Jensen et al. [76] argue that the size of a metadata file or the metadata content of an experiment affects the insert time, and the hybrid XML/relational approach performs better than an XML database. The insert performance for a scaled workload ranged between 80 ms and 120 ms.

MetaStore exposes all its functionalities including the metadata insert operation through REST services. Comparing the write performance of MetaStore with the MCS web interface and the XMC Cat metadata catalog, it is clear that MetaStore is easily able to handle more than 800 requests with 50 concurrent users (threads) per second, and the performance improved with the addition of a second MetaStore instance.

Read performance Figures 9 and 11 show the results of the read performance. The number of read operations on the MetaStore framework are expected to be higher than the write operations and hence the reads/sec factor is approximately five times more

[20] https://www.oracle.com/database/berkeley-db/xml.html.

than write/sec. As the documents in ArangoDB and MongoDB are fully indexed, all read operations are arbitrary full-text searches.

In the case of Metacat, the XML tree structure represented by a Document Object Model (DOM) [77] is decomposed into nodes, where the root node is the document entity and the children nodes are elements and attributes. These nodes are stored in a relational database schema, for allowing querying of the metadata at different depths. Nested SQL queries are defined with a specific depth or full-text search queries are defined with depth zero. For evaluating the read performance, the full-text query was executed on a node size of 2.03×10^6 and the nested query on a node size of 23,000/25,000 with a depth equal to five. For full-text search queries, it took 30 s when using indexes and 29 s when using a nested query. For the queries using indexes with a node depth varying between one and six, the query execution time ranged between 1 and 50 s, whereas for SQL nested queries, the execution time was between 1 and 10 s.

The MCS query interface provides simple queries that do a value match for a single static attribute in the file and complex queries that do value matching for all the attributes in the file. For simple queries with a single client executing multiple concurrent threads on a native Java API, a peak of approximately 2300 queries per second was reached, whereas for the web interface approximately 120 queries per second was achieved. For the complex queries, approximately 560 queries per second was achieved and for the web interface, approximately 100 queries per seconds was possible.

Compared to these results the full-text read performance of MetaStore (considering both ArangoDB and MongoDB) varied between 19 and 49 ms for 10 to 50 concurrent users with 1650 to 3200 reads/sec, thus, showing significantly better performance compared to that of Metacat, the LEAD XMC Cat metadata catalog, or the MCS query interface.

Analyzing the performance comparison between MetaStore and existing metadata management systems, it necessary to understand how metadata is modeled, stored and queried in these systems. Following are the reasons why the performance of MetaStore is better than that of the existing systems: (1) In MetaStore the XML containing the metadata is never decomposed into atomic entities and stored in a relational database but transformed to JSON and stored as a JSON file or as a binary-encoded serialization of JSON (BSON[21]) in the NoSQL database. The JSON-based storage of metadata in NoSQL databases avoids the overhead of transforming XML either as a CLOB or shredding it using the inlining approach. (2) As the metadata schema is fully indexed, the I/O intensive nested queries for retrieving specific paths from the XML structure are not required. (3) The overhead of reconstructing XML documents from the relational schema is completely avoided because the queries return JSON objects that are serialized to XML based on the registered metadata schema. (4) In the case of NoSQL databases, the write operations have a better throughput because they are primarily performed in the main memory, instead of writing to the disk, as in the case of an SQL based database. For example, in ArangoDB the memory-mapped

[21] http://bsonspec.org/spec.html.

files are used to handle write operations and are regularly synced to disk via the `fsync` command.

6 Application use cases

In this section, we briefly explain three use cases (one from bio-medical research and two from digital humanities) for which MetaStore is adopted.

Use case 1: nanoscopy

For illustrating the applicability of MetaStore in managing heterogeneous metadata models, first, we consider the light super-resolution microscopy (nanoscopy) use case. Nanoscopy is a novel imaging technique that aims at bridging the resolution gap between conventional light microscopy and electron microscopy [78]. A typical nanoscopy investigation begins with the acquisition of raw image datasets in HDF5, KDF, or TIFF format from a high-resolution microscope. A complete series of measurements for a given specimen can easily amount to 100–150 TB in size. For handling such large volumes of data, NORDR offers efficient handling of massive data volumes collected from various geolocated high-resolution microscopes, backed by a large-scale data storage and high-performance computing cluster [79]. Various nanoscopy scientific workflows are executed for processing these raw datasets. These workflows are described using the Business Process Execution Language (BPEL) [80] and the administrative and descriptive metadata is described based on the community specified metadata schema.

For automating the metadata extraction from raw datasets, we implemented a staging processor that extracts the metadata from HDF5 and TIFF files during the ingestion of the raw datasets in NORDR. This extracted metadata is submitted to MetaStore via the REST interface where it is verified with the registered localization microscopy metadata schema and stored in the document store of ArangoDB. Using the Prov2ONE algorithm, the workflow defined in BPEL is translated into the ProvONE prospective provenance, which is enriched with the retrospective provenance collected during the workflow execution. The entire ProvONE graph is serialized in RDF and stored in Apache Jena. As the functionality (services) for handling the extracted metadata is automatically created during the schema registration stage, we do not need to create any additional services in MetaStore explicitly. Moreover, the extracted metadata is wrapped in the METS format that complies to a registered METS-profile and exposed to the researcher communities for metadata harvesting through OAI-PMH.

Use case 2: Corpus Vitrearum Deutschland

The CVD is a part of the Corpus Vitrearum Medii Aevi (CVMA), which is long-term international research project in the digital humanities. The CVMA aims at digitizing, cataloging and analyzing the medieval stained glasses that are preserved in churches, museums, and galleries all over Europe. Currently, the CVD research community has digitized and published 5086 images of stained glasses, out of which two examples are shown in Fig. 12. On the one hand, the CVD image repository is implemented for storing and accessing these images, but on the contrary discovery and analysis of the images based on the metadata is lacking. Each image is embedded with an extensive set of metadata attributes that conform to the XMP metadata standard. Most of these

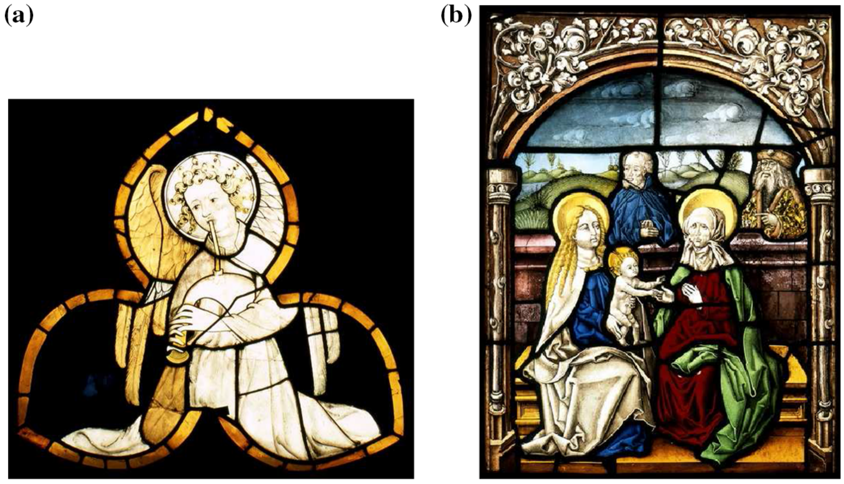


Fig. 12 Digitized medieval stained glasses. **a** An angel with bagpipes. Photo: Andrea Gössel, CVMA Germany/Freiburg, CC BY-NC 4.0, http://id.corpusvitrearum.de/images/4486.html. **b** The holy family. Photo: Andrea Gössel, CVMA Germany / Freiburg, CC BY-NC 4.0, http://id.corpusvitrearum.de/images/3431.html

attributes are filled during the digitization process, while few attributes that describe the painted figures or depicted scenes are referred from a predefined ICONCLASS vocabulary.

Similar to the nanoscopy use case, the first step is to extract the metadata that is embedded in the image (TIFF). For this, we reused the TIFF metadata extractor staging processor of the nanoscopy use case with minor modifications. The extracted metadata is submitted to MetaStore through the REST interface, where the metadata is validated before inserting in the document store of ArangoDB. For automatically verifying the correctness of the terms describing the stained glasses, the corresponding ICONCLASS vocabulary is linked to the XMP metadata schema during the metadata registration stage.

Use case 3: eCodicology

The eCodicology is a research project in the humanities that aims at designing, evaluating and optimizing algorithms for identifying the layout features of medieval manuscripts [81]. First, the manuscripts from the "Virtual Scriptorium St. Matthias" were digitized, and a total of 170,000 images were acquired. An example image is shown in Fig. 13. These images are stored in CodiStore, which is a scientific data repository supporting long-term archival and reuse of digitized medieval manuscripts [81]. Currently, the CodiStore is not capable of handling the metadata embedded within the datasets (static metadata) and the metadata generated during the workflow execution (dynamic metadata). Hence, the entire MetaStore has to be integrated with the CodiStore.

In the following, we describe our experience and the challenges faced during the integration of MetaStore with CodiStore. During the digitization process, each manuscript (each containing a few hundreds of pages) is enriched with bibliographical

Fig. 13 Digitized manuscript page Hs 1108/55 4° 37r from St. Matthias (City Library of Trier)

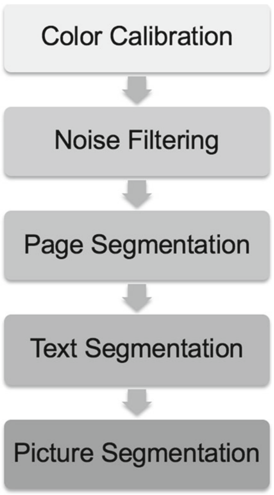


Fig. 14 Layout feature extraction workflow

metadata describing the following attributes: *Signature, Century, Material, Format, Leaf Count, Library* [32]. For systematically structuring the metadata for an entire manuscript, the eCodicology community has adopted the Text Encoding Initiative (TEI) format that is further embedded in a METS file, which conforms to an eCodicology METS-profile. This METS file also contains the administrative metadata, structural map, and structural links that describe the hierarchy of the pages within a manuscript and the logical and physical links between the pages. As the metadata in TEI and METS is not subject to change, we consider it as static metadata. Similar to the previously use cases, during the ingest of data (digitized manuscripts) in the CodiStore data repository, we implemented a staging processor for extracting TEI metadata. The extracted metadata is inserted in the document store of ArangoDB with prior quality control.

The handling of dynamic metadata (i.e., metadata generated during the execution of a workflow) is a challenging task. For each digitized page the Layout Feature Extraction workflow (see Fig. 14) is executed. Similar to the nanoscopy workflow, the provenance for the workflow is captured in ProvONE and stored in Apache Jena. The workflow consists of five steps; *Color Calibration, Noise Filtering, Page Segmentation, Text Segmentation, and Picture Segmentation*. Each step in the workflow generates metadata that describes the output generated by a particular step. For example, the *Color Calibration* generates a color calibration matrix, the output of *Page Segmentation* is the page space measurements (page area, height, width, inclination angle, and left-corner co-ordinates), and the *Text Segmentation* generates the written space measurements (main text area, height, width and inclination angle) and number of lines. This workflow-specific metadata is subject to frequent changes because modifying the workflow parameters will yield different outputs, or changing the algorithms will generate entirely different results. Hence, to handle such dynamic metadata, we had to extend MetaStore with an annotation framework that supports modeling of ad hoc metadata schemas conforming to the W3C recommended WADM. For this, we integrated the Anno4j library with an RDF triple store (Apache Jena).

7 Discussion

In this section, we explain the various architecture design decisions undertaken in the implementation of the MetaStore framework. Considering the architecture design of MetaStore, as a core design principle, we adopted the principle of modular architecture design pattern. Adopting this design pattern has a multitude of advantages. First, each feature of MetaStore is developed as an independent component (high cohesion) with no inter-component dependencies (low coupling). This allows component-specific updating of features with easy maintenance of the entire framework. For example, initially, the Metadata Quality Control component offered only basic schema validation and well-formedness checks. However, with modular architecture design of MetaStore, it is easy to extend this component to support domain-specific vocabulary metadata content verification. Second, it enables the seamless integration of new features (components) while reusing the existing ones. For example, the integration of an RDF database and the support for building and reusing SKOS-based domain-specific vocabularies is seamlessly appended to the core functionality of MetaStore. Moreover, the same RDF database is used to store and query the ProvONE provenance graphs. Third, customizable integration of MetaStore with existing systems kept the complexity of the overall architecture to a minimum. For example, for the nanoscopy use case, as there is no dynamic metadata either from the workflows or the annotations, only the MetaStore core layer is integrated with NORDR. Thus, the modular design of MetaStore allowed integration of selective features from MetaStore with NORDR. However, for handling diverse metadata in eCodicology research, i.e., static metadata encoded in TEI format, dynamic annotations in the WADM, and workflow provenance in ProvONE, the entire MetaStore framework is integrated with CodiStore.

With the MetaStore architecture design based on NoSQL database systems, different types of metadata, such as descriptive, administrative, provenance and structural metadata can be efficiently modeled to maximize the utilization of the data models offered by the ArangoDB and Apache Jena. This decision not only vastly reduced the complexity of the architecture (technology footprint) but also the repetitive software development effort in redesigning, implementing and updating of a relational database schema is avoided. Moreover, with the adoption of existing open-source solutions, the total cost of ownership is minimal. For example, workflow and provenance metadata comprising complex relationships are modeled in the ProvONE provenance model and serialized as RDF triples for allowing execution of graph traversal queries. Descriptive, administrative and structural metadata that can be serialized in XML is modeled in the document data model for enabling full-text search and complex analytical queries. Regarding database scalability in handling increasing data volumes, with ArangoDB it is possible to shard (distribute) large volumes of metadata in multiple database instances.

The MetaStore offers different research communities an entirely automated metadata management system that is capable of handling both community-specific metadata schemas as well as the existing metadata standards. The Metadata Code Generator component automatically generates the necessary code (services) for handling the registered metadata schema and on-the-fly extends the functionality of MetaStore. Hence, significantly reducing the resources and efforts of scientific communities in writing and maintaining the software (services) that are necessary for implementing the metadata schema. For generating the code, we use the JAXB-XJC[22] library, and at runtime, the Metadata Management Component determines the appropriate metadata schema that needs to be created using Java Reflections API [82].

For enabling large-scale metadata harvesting in MetaStore, it is necessary to implement the six verbs of the OAI-PMH specification. Initially, our approach was to integrate the jOAI web application[23] that provides an implementation of the OAI-PMH specification. However, it involves setting up an additional data provider server containing the metadata in XML format that conforms to a set of rules described by the jOAI web application. This would have been an unnecessary overhead, for not only maintaining an additional metadata file-server just for metadata harvesting purpose but also in transforming the metadata stored in ArangoDB to the jOAI-compliant format. Instead, we implemented the six OAI-PMH verbs as AQL queries in MetaStore, allowing better query performance as compared to native file I/O operations in the case of jOAI. Hence, our architectural design approach for implementing the OAI-PMH specification within MetaStore and directly across different NoSQL databases significantly reduced our cost and effort in using and maintaining additional metadata harvesting tools and infrastructures.

With the availability of numerous metadata schemas, each specific for a given application, it is evident that research communities will adopt multiple schemas simultaneously to describe their data. For example, the nanoscopy research community adopted the Core Scientific Metadata (CSMD) model for describing the administrative metadata, a community-specific metadata schema for the descriptive metadata,

[22] https://jaxb.java.net/2.2.4/docs/xjc.html.

[23] https://uc.dls.ucar.edu/joai/.

and the PREMIS and ProvONE for the provenance metadata. On the one hand, supporting heterogeneous metadata schemas is realized by MetaStore, but on the contrary, the aggregate modeling of metadata for allowing comprehensive metadata harvesting required for metadata publishing, or during data and metadata migration is a challenging aspect. For this, we adopted the METS schema and provided a default METS-profile that allows us to aggregate heterogeneous metadata schemas. However, as METS supports only PREMIS for modeling provenance, it is necessary to define the SKOS-based vocabulary mapping between PREMIS and ProvONE to enable provenance interoperability.

Currently, the existing workflow management systems and the provenance interoperability frameworks are based on the PROV model or the OPM [83–85]. On the one hand, OPM and PROV allow provenance interoperability through a standard provenance model, but on the other hand, a core limitation of these models is that they are capable of handling only the retrospective provenance and not the prospective provenance. In scientific research where complex workflows are designed, executed, repeated, and continuously evolve, the prospective provenance is of critical importance. Thus, for handling the entire provenance trace for scientific workflows in a single provenance model, we adopted the ProvONE provenance model. With the availability of workflow provenance in ProvONE, it is possible to design queries that encompass the analysis of the workflow results along with the corresponding workflow definition. In the case of communities where the research is still growing, the availability of provenance in ProvONE is useful in avoiding redundant execution of obsolete workflows, with the possibility to rapidly evolve their workflow to generate better results. Moreover, for long-term sustainability and adoption of a standard querying language (SPARQL), the ProvONE graphs are serialized in the RDF data model.

Based on the software architecture design pattern adopted for implementing MetaStore, and the generic adaptability of MetaStore for diverse use cases, in the following, we summarize the benefits of MetaStore: (a) The modular architecture design pattern implicitly provides a clear separation of concerns, allowing us to realize the requirements as separate modules in the architecture. Moreover, with low inter-module coupling, the maintenance and upgrading of the components are easy. (b) With the integration of open-source solutions and the automatic adaptive architecture design, the overall total cost of ownership is minimal. Adopting the dynamic composition design pattern further eliminates the need in following the routine software development life cycle. (c) By exposing all the features through a well-defined REST interface, a seamless integration of MetaStore with existing systems is possible. For example, NORDR, CodiStore and CVD image repository were easily integrated with MetaStore. (d) Conforming to the existing metadata standards in handling provenance and annotation metadata, and with an implementation of the widely accepted OAI-PMH metadata harvesting specification, we guarantee the long-term sustainability and reuse of the MetaStore framework. (e) The flexible data model of NoSQL and RDF databases enables us not only to handle ad hoc metadata models but also provides the inherent sharding capabilities for managing increasing metadata volumes. Moreover, with the adoption of these databases, it is possible to efficiently store and query heterogeneous metadata schemas in the appropriate data model.

8 Conclusion

In this paper, we presented MetaStore, a novel metadata management framework for comprehensive management of heterogeneous metadata schemas. For systematically realizing the requirements stated by research communities, the core architecture of MetaStore is based on the principle of modular design. On the one hand, the modular design enables the requirements to be realized as task-specific components, but on the other hand, the architecture design is still static, i.e., for handling new metadata schemas, the required functionality has to be manually implemented in MetaStore. To overcome this static nature of MetaStore, the modular design of MetaStore is enhanced with the principle of Compositional Adaption, wherein the new functionality in MetaStore is dynamically generated at runtime based on the dynamic composition design.

For utilizing the features that support a flexible data model, horizontal scaling (sharding) and efficient query performance, the MetaStore framework is intentionally based on a NoSQL database and an RDF triple store. The presented version of the MetaStore framework is designed on ArangoDB and Apache Jena framework, for efficient modeling and querying the metadata in the key-value, document and graph data model. To leverage the functionality offered by NoSQL databases and to completely automate the metadata management in scientific data repositories, MetaStore provides the following features:

– a metadata schema registry for validating metadata, with an extension to integrate discipline-specific SKOS vocabularies for allowing automated metadata quality control,
– on-the-fly generation, compilation and deployment of code for creating the services necessary for handling registered metadata schemas (zero downtime upgrade of MetaStore),
– automated index creation in NoSQL databases for enabling full-text search, with integration of ElasticSearch for allowing execution of complex analytical queries, faceted search, and fuzzy search,
– providing an annotation framework for enabling researchers to add descriptive information (dynamic metadata) in the form of annotations, with support of WADM for enabling annotation interoperability,
– an OAI-PMH metadata harvester implemented on a NoSQL data provider for enabling metadata harvesting (metadata sharing).

Furthermore, for enabling provenance interoperability between the relevant provenance models, the mapping rules between the PREMIS and ProvONE vocabularies were presented. These mapping rules are implemented in the Provenance Manager component of MetaStore. To fulfill the metadata related requirements of the nanoscopy, eCodicology, and CVD research communities and to verify the generic applicability of MetaStore, the MetaStore framework is integrated into NORDR, CodiStore, and CVD image repository. For evaluating the MetaStore framework, first, we presented a feature based comparison of MetaStore with existing metadata management systems. We observe that MetaStore not only fulfills an exhaustive coverage of required functionalities but also adheres to the recommended standards for long-term architecture design sustainability and metadata interoperability. Second, we presented the performance-based evaluation, wherein we highlighted the benefit of designing MetaStore on a NoSQL database over the traditional SQL or XML databases. Furthermore, for evaluating the scalability of MetaStore, various configurations of MetaStore were evaluated with different workloads, and it is observed that the performance of MetaStore is significantly better compared to existing metadata management systems. The various MetaStore configurations serve as a reference to scientific communities when setting up their metadata management system.

In our ongoing work, we are currently integrating the Apache Spark [86] framework with MetaStore for allowing execution of complex analytical queries and enabling graph mining over the annotation and provenance graphs.

Acknowledgements This research is supported by the Portfolio Extension of Helmholtz Association "Large Scale Data Management and Analysis" and DFG (German Research Foundation) MASi Project (STO 397/4-1).

References

1. Hey, T., Trefethen, A.: The Data Deluge: An e-Science Perspective. Wiley and Sons (2003)
2. Gutierrez, D.D.: InsideBIGDATA guide to scientific research. http://insidebigdata.com/2015/12/01/insidebigdata-guide-to-scientific-research/. Accessed 9 June 2017
3. Berry, D., Parastatidis, S.: e-Science workflow services workshop, December 2003. http://www.nesc.ac.uk/esi/events/303/index.html. Accessed 10 June 2017
4. Gannon, D., Fox, G., Farazdel, A., Goble, C., Deelman, E., Berry, D.: Workflow in grid systems workshop, March 2004. http://www.extreme.indiana.edu/groc/Worflow-call.html. Accessed 16 June 2017
5. Jacob, J., Katz, D., Miller, C., et al.: GRIST workshop on service composition for data exploration in the virtual observatory, July 2004. http://www.roe.ac.uk/~rgm/sc4devo/sc4devo1/index.html. Accessed 10 June 2017
6. LINK-Up Workshop on Scientific Workflows, October 2004. http://kbis.sdsc.edu/events/link-up-11-04/. Accessed 16 June 2017
7. Deelman, E., Gil, Y., Zemankova, M.: NSF Workshop on the Challenges of Scientific Workflows, May 2006. https://www.nsf.gov/events/event$_$summ.jsp?cntn$_$id=108411. Accessed 16 June 2017
8. Gray, J., Liu, D.T., Nieto-Santisteban, M., Szalay, A., DeWitt, D.J., Heber, G.: Scientific data management in the coming decade. SIGMOD Rec. **34**(4), 34–41 (2005)
9. Graybeal, J., Miller, S.P., Stocks, K.: The MMI guides: navigating the world of marine metadata. http://uop.whoi.edu/techdocs/presentations/MMI_Guides.pdf (2010). Accessed 15 June 2017
10. Lemmer, P., Gunkel, M., Baddeley, D., Kaufmann, R., Urich, A., Weiland, Y., Reymann, J., Müller, P., Hausmann, M., Cremer, C.: SPDM: light microscopy with single-molecule resolution at the nanoscale. Appl. Phys. B **93**(1), 1 (2008)
11. National Information Standards Organization: Understanding Metadata, NISO Press, Bethesda http://www.niso.org/publications/press/understanding_metadata (2004). Accessed 15 May 2017
12. Dimitrovski, I., Kocev, D., Loskovska, S., Džeroski, S.: Hierarchical annotation of medical images. Patt. Recogn. **44**(1011), 2436–2449 (2011)
13. Hu, B., Dasmahapatra, S., Lewis, P., Shadbolt, N.: Ontology-based medical image annotation with description logics. In: Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence, pp. 77–82 (2003)
14. Blanke, T., Hedges, M., Dunn, S.: Arts and humanities e-science: current practices and future challenges. Fut. Gener. Comput. Syst. **25**(4), 474–480 (2009)
15. Gao, S., Sperberg-McQueen, C.M., Thompson, H.S., Mendelsohn, N., Beech, D., Maloney, M.: W3C XML schema definition language (XSD) 1.1 part 1: structures. W3C Candidate Recommendation **30**(7.2) (2009)
16. Higgins, D., Berkley, C., Jones, M. B.: Managing heterogeneous ecological data using Morpho. In: Proceedings 14th International Conference on Scientific and Statistical Database Management, pp. 69–76 (2002)

17. Frew, J., Bose, R.: Earth system science workbench: a data management infrastructure for earth science products. In: Proceedings Thirteenth International Conference on Scientific and Statistical Database Management. SSDBM, pp. 180–189 (2001)
18. Pancerella, C., Hewson, J., et al: Metadata in the collaboratory for multi-scale chemical science. In: International Conference on Dublin Core and Metadata Applications (2003)
19. Malet, G., Munoz, F., Appleyard, R., Hersh, W.: A model for enhancing internet medical document retrieval with medical core metadata. J. Am. Med. Inf. Assoc. **6**(2), 163 (1999)
20. Prabhune, A., Ansari, H., Keshav, A., Stotzka, R., Gertz, M., Hesser, J.: Metastore: a metadata framework for scientific data repositories. In: IEEE International Conference on Big Data (Big Data), pp. 3026–3035 (2016)
21. Cuevas-Vicenttín, V., Ludäscher, B,. Missier, P., Belhajjame, K., Chirigati, F., Wei, Y., Dey, S., Kianmajd, P., Koop, D., Bowers, S., et al.: ProvONE: a PROV extension data model for scientific workflow provenance (2015)
22. PREMIS Working Group et al.: Data dictionary for preservation metadata: final report of the premis working group. OCLC Online Computer Library Center & Research Libraries Group, Dublin, OH, USA, Final report (2005)
23. Lagoze, C., Van de Sompel, H., Nelson, M., Warner, S.: The open archives initiative protocol for metadata harvesting-version 2.0 (2002)
24. McDonough, J.P.: METS: standardized encoding for digital library objects. Int. J. Digit. Libr. **6**(2), 148–158 (2006)
25. Miles, A., Matthews, B., Wilson, M., Brickley, D.: SKOS core: simple knowledge organisation for the web. In: International Conference on Dublin Core and Metadata Applications, pp. 3–10 (2005)
26. Gormley, C., Tong, Z.: Elasticsearch: The Definitive Guide. O'Reilly Media, Inc., Sebastopol (2015)
27. Apache Jena. A free and open source java framework for building semantic web and linked data applications. https://jena.apache.org. Accessed 15 March 2017
28. Prabhune, A., Zweig, A., Stotzka, R., Gertz, M., Hesser, J.: Prov2ONE: An Algorithm for Automatically Constructing ProvONE Provenance Graphs, pp. 204–208. Springer International Publishing (2016)
29. Carlson, J.L.: Redis in Action. Manning Publications Co., Greenwich (2013)
30. Banker, K.: MongoDB in Action. Manning Publications Co., Greenwich (2011)
31. Vukotic, A., Watt, N., Abedrabbo, T., Fox, D., Partner, J.: Neo4j in Action. Manning Publications Co., Greenwich (2015)
32. Chandna, S., Rindone, F., Dachsbacher, C., Stotzka, R.: Quantitative exploration of large medieval manuscripts data for the codicological research. In: 2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV), pp. 20–28 (2016)
33. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.: Composing adaptive software. Computer **37**(7), 56–64 (2004)
34. OASIS. Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (2007)
35. Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Bhagat, J.: The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucl Acids Res **41**, W557–W561 (2013)
36. Lee, E.A., Neuendorffer, S.: MoML: a modeling markup language in SML: version 0.4. Electronics Research Laboratory, University of California (2000)
37. Prud, E., Seaborne, A., et al.: SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, Accessed 15 March 2017
38. Zhao, Y., Wilde, M., Foster, I.: Applying the Virtual Data Provenance Model. Springer, Berlin (2006)
39. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., et al.: The open provenance model core specification (v1. 1). Fut. Gener. Comput. Syst. **27**(6), 743–756 (2011)
40. Sahoo, S., Groth, P., Hartig, S.M., Miles, S., Gil, Y., Myers, J., Moreau, L., Panzer, M., Zhao, J., Garijo, D.: Provenance Vocabulary Mappings. W3C Provenance Incubator Group (2010)
41. Weibel, S., Kunze, J., Lagoze, C., Wolf, M.: Dublin core metadata for resource discovery. Technical report (1998)
42. Berndl, E., Schlegel, K., Eisenkolb, A., Kosch, H.: Idiomatic persistence and querying for the W3C Web Annotation Data Model. In: Joint Proceedings of the 4th International Workshop on Linked Media and the 3rd Developers Hackshop Co-located with the 13th Extended Semantic Web Conference ESWC (2016)

43. Suominen, O., Ylikotila, H., Pessala, S., Lappalainen, M., Frosterus, M., Tuominen, J., Baker, T., Caracciolo, C., Retterath, A.: Publishing SKOS Vocabularies with Skosmos. Manuscript submitted for review (2015)
44. Scholz, H.: Die mittelalterlichen Glasmalereien in Mittelfranken und Nürnberg: extra muros, vol. 10. Deutscher Verlag für Kunstwissenschaft (2002)
45. Scholz, H.: Die mittelalterlichen Glasmalereien in Nürnberg: Sebalder Stadtseite. Deutscher Verlag für Kunstwissenschaft (2013)
46. Couprie, L.D.: Iconclass: an iconographic classification system. Art Libr. J. **8**(2), 3249 (1983)
47. Ball, A., Chen, S., Greenberg, J., Perez, C., Jeffery, K., Koskela, R.: Building a disciplinary metadata standards directory. Int. J. Digit. Curation **9**(1), 142–151 (2014)
48. Ben-Kiki, O., Evans, C., Ingerson, B.: YAML Ain't Markup Language (YAML) version 1.1. yaml. org, Technical Report (2005)
49. Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., Tuecke, S.: GridFTP: protocol extensions to FTP for the grid. Global Grid Forum GFD-RP **20**, 1–21 (2003)
50. Whitehead, E.J., Wiggins, M.: WebDAV: IEFT standard for collaborative authoring on the web. IEEE Internet Comput. **2**(5), 34–40 (1998)
51. Marcial, L.H., Hemminger, B.M.: Scientific data repositories on the web: an initial survey. J. Am. Soc. Inf. Sci. Technol. **61**(10), 2029–2048 (2010)
52. Woodberry, E., Bailey, C.W.: SPEC Kit 292: Institutional Repositories. Australian Acad. Res. Libr. **39**(2), 129–130 (2008)
53. Lynch, C.A., Lippincott, J.K.: Institutional repository deployment in the united states as of early 2005. D-lib Mag. **11**(9), 1–11 (2005)
54. Smith, M., Barton, M., Bass, M., Branschofsky, M., McClellan, G., Stuve, D., Tansley, R., Walker, J.H.: DSpace: an open source dynamic digital repository. D-Lib Mag. **9**(1) (2003). http://www.dlib.org/dlib/january03/smith/01smith.html
55. Van Garderen, P.: Archivematica: using micro-services and open-source software to deliver a comprehensive digital curation solution. In: Proceedings of the 7th International Conference on Preservation of Digital Objects, Vienna, Austria, pp. 145–149 (2010)
56. Flannery, D., Matthews, B., Griffin, T., Bicarregui, J., Gleaves, M., Lerusse, L., Downing, R., Ashton, A., Sufi, S., Drinkwater, G., Kleese, K.: ICAT: integrating data infrastructure for facilities based science. In: Fifth IEEE International Conference e-Science '09, pp. 201–207 (2009)
57. Sufi, S., Mathews, B.: CCLRC scientific metadata model: version 2. Technical report, CCLRC technical report DL TR2004001 (2004)
58. Lecarpentier, D., Wittenburg, P., Elbers, W., Michelini, A., Kanso, R., Coveney, P., Baxter, R.: EUDAT: a new cross-disciplinary data infrastructure for science. Int. J. Digit. Curation **8**(1), 279–287 (2013)
59. Grainger, T., Potter, T., Seeley, Y.: Solr in Action. Manning, Cherry Hill (2014)
60. Beazley, M.: EPrints institutional repository software: a review. Partnership **5**(2), 1 (2010)
61. Jensen, S., Plale, B.: Using characteristics of computational science schemas for workflow metadata management. In: IEEE Congress on Services—Part I, pp. 445–452 (2008)
62. Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D. J., Naughton, J.F.: Relational databases for querying XML documents: limitations and opportunities. In: Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), pp. 302–314, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc (1999)
63. Jones, M.B., Berkley, C., Bojilova, J., Schildhauer, M.: Managing scientific metadata. IEEE Internet Comput. **5**(5), 59–68 (2001)
64. Yang, R., Deng, X., Kafatos, M., Wang, C., Wang, X.S.: An XML-based Distributed Metadata Server (DIMES) supporting earth science metadata. In: Proceedings Thirteenth International Conference on Scientific and Statistical Database Management. SSDBM, pp. 251–256 (2001)
65. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The SDSC storage resource broker. In: Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research, CASCON '98, p. 5. IBM Press, New York (1998)
66. Singh, G., Bharathi, S., Chervenak, A., Deelman, E., Kesselman, C., Manohar, M., Patil, S., Pearlman, L.: A metadata catalog service for data intensive applications. In: Supercomputing, 2003 ACM/IEEE Conference, pp. 33–33 (2003)
67. Deelman, E., Singh, G., Atkinson, M.P., Chervenak, A., Hong, N.C., Kesselman, C., Patil, S., Pearlman, L., Su, M.H.: Grid-based metadata services. In: Proceedings. 16th International Conference on Scientific and Statistical Database Management, pp. 393–402 (2004)

68. Pham, Q., Malik, T., Foster, I.T., Di Lauro, R., Montella, R.: SOLE: linking research papers with science objects. In: IPAW, pp. 203–208. Springer, Berlin (2012)
69. McCandless, M., Hatcher, E., Gospodnetic, O.: Lucene in Action, 2nd edn. Covers Apache Lucene 3.0. Manning Publications Co., Greenwich (2010)
70. Belhajjame, K., B'Far, R., Cheney, J., Coppens, S., Cresswell, S., Gil, Y., Groth, P., Klyne, G., Lebo, T., McCusker, J., Miles, S., Myers, J., Sahoo, S., Curt, T.: PROV-DM: the PROV data model. Project report (2013)
71. Schandl, T., Blumauer, A.: PoolParty: SKOS thesaurus management utilizing linked data. In: The Semantic Web: Research and Applications: 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30–June 3, 2010, Proceedings, Part II, pp. 421–425. Springer, Berlin, Heidelberg (2010)
72. Culhane, W., Kogan, L., Jayalath, C., Eugster, P.: LOOM: optimal aggregation overlays for in-memory big data processing. In: 6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14), Philadelphia, USENIX Association (2014)
73. Deelman, E., Berriman, B., Chervenak, A., Corcho, O., Groth, P., Moreau, L.: Metadata and provenance management. In: Scientific Data Management: Challenges, Technology, and Deployment, 1st edn. (2009)
74. Li, Y., Manoharan, S.: A performance comparison of SQL and NoSQL databases. In: IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), pp. 15–19 (2013)
75. Boicea, A., Radulescu, F., Agapin, L.I.: MongoDB vs oracle-database comparison. In: EIDWT, pp. 330–335 (2012)
76. Jensen, S., Ghoshal, D., Plale, B.: Evaluation of two XML storage approaches for scientific metadata. Indiana University Department of Computer Science Technical Report (2011)
77. Wood, L., Le Hors, A., Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Nicol, G., Robie, J., Sutor, R., Wilson, C.: Document object model (DOM) level 1 specification. W3C recommendation (1998)
78. Cremer, C., Kaufmann, R., Gunkel, M., Pres, S., Weiland, Y., Müller, P., Ruckelshausen, T., Lemmer, P., Geiger, F., Degenhard, S., Christina, W., Lemmermann, N., Holtappels, R., Strickfaden, H., Hausmann, M.: Superresolution imaging of biological nanostructures by spectral precision distance microscopy. Biotech. J. **6**(9), 1037–1051 (2011)
79. Prabhune, A., Stotzka, R., Jejkal, T., Hartmann, V., Bach, M., Schmitt, E., Hausmann, M., Hesser, J.: An optimized generic client service API for managing large datasets within a data repository. In: Big Data Computing Service and Applications (BigDataService), IEEE First International Conference, pp. 44–51 (2015)
80. Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guzar, A.: Web services business process execution language version 2.0. OASIS Stand. **11**(120), 5 (2007)
81. Chandna, S., Tonne, D., Jejkal, T., Stotzka, R., Krause, C., Vanscheidt, P., Prabhune, A.: Software workflow for the automatic tagging of medieval manuscript images (SWATI). In: SPIE/IS&T Electronic Imaging, p. 940206 (2015)
82. Forman, I.R., Forman, N.: Java Reflection in Action. Manning Publication Co., Greenwich (2004)
83. Altintas, I., Anand, M.K., Crawl, D., Bowers, S., Belloum, A., Missier, P., Ludäscher, B., Goble, C.A., Sloot, P.M.: Understanding Collaborative Studies Through Interoperable Workflow Provenance. Springer, Berlin (2010)
84. Braun, U., Seltzer, M.I., Chapman, A., Blaustein, B.T., Allen, M.D., Seligman, L.: Towards query interoperability: PASSing PLUS. In: TaPP, pp. 1–10 (2010)
85. Missier, P., Ludäscher, B., Bowers, S., Dey, S., Sarkar, A., Shrestha, B., Altintas, I., Anand, M.K., Goble, C.: Linking multiple workflow provenance traces for interoperable collaborative science. In: The 5th Workshop on Workflows in Support of Large-Scale Science, pp. 1–8 (2010)
86. Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media, Inc., Sebastopol (2015)