ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Making data platforms smarter with MOSES

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Making data platforms smarter with MOSES / Matteo Francia, Enrico Gallinucci, Matteo Golfarelli, Anna Giulia Leoni, Stefano Rizzi, Nicola Santolini. - In: FUTURE GENERATION COMPUTER SYSTEMS. - ISSN 0167-739X. - STAMPA. - 125:(2021), pp. 299-313. [10.1016/j.future.2021.06.031]

Availability:
This version is available at: https://hdl.handle.net/11585/827383 since: 2021-07-04

Published:
DOI: http://doi.org/10.1016/j.future.2021.06.031

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

18 April 2024

This is the final peer-reviewed accepted manuscript of:

Francia, M., Gallinucci, E., Golfarelli, M., Leoni, A. G., Rizzi, S., & Santolini, N. (2021). Making data platforms smarter with MOSES. Future Generation Computer Systems, 125, 299-313.

The final published version is available online at: **http://dx.doi.org/10.1016/j.future.2021.06.031**

Rights / License:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/)

When citing, please refer to the published version.

Making Data Platforms Smarter with MOSES

Matteo Francia, Enrico Gallinucci, Matteo Golfarelli*, Anna Giulia Leoni, Stefano Rizzi, Nicola Santolini

DISI – University of Bologna, Via dell'Università 50, 47522 Cesena, Italy

DISI, University of Bologna, Bologna, Italy

Abstract

The rise of data platforms has enabled the collection and processing of huge volumes of data, but has opened to the risk of losing their control. Collecting proper metadata about raw data and transformations can significantly reduce this risk. In this paper we propose MOSES, a technology-agnostic, extensible, and customizable framework for metadata handling in big data platforms. The framework hinges on a metadata repository that stores information about the objects in the big data platform and the processes that transform them. MOSES provides a wide range of functionalities to different types of users of the platform. Differently from previous high-level proposals, MOSES is fully implemented and it was not conceived for a specific technology. Besides discussing the rationale and the features of MOSES, in this paper we describe its implementation and we test it on a real case study. The ultimate goal is to take a significant step forward towards proving that metadata handling in big data platforms is feasible and beneficial.

Keywords: Data lake, Metadata, Big data, Data platform

1. Introduction

The huge volume of data collected to enable advanced analytics within an enterprise or an organization is becoming more and more heterogeneous and complex; as a consequence, traditional data warehouses can no longer be seen as the ideal data hubs for integration and analysis [1]. With the rise of big data architectures, *data lakes* (DLs) have increasingly taken the role of such hubs. Though the term was coined 20 years ago, a widely accepted definition is still lacking since features and goals of DLs are continuously evolving. The initial definition was mainly focused on ensuring storage for raw and heterogeneous data; however, raw data are difficult to obtain, challenging to interpret and describe, and tedious to maintain, especially as the number of sources grows. Thus, there is a need for thoroughly describing and curating the data to make them consumable [2]. The advantage of a DL over a purpose-built data store is that the former eliminates the up-front costs of data ingestion, like transformations, since data are stored in their original format. Thus, once in the DL, data are available for analysis by everyone in the organization.

The next step in the evolution of data hubs was the understanding that getting value from data is not only a matter of storage, but it also requires appropriate analytical skills and techniques to be applied in an integrated and multi-level fashion [3]. An attempt to condense these features into a single definition has been done by Couto et al. [4]: "*A DL is a central repository system for storage, processing, and analysis of raw data, in which the data is kept in its original format and is processed to be queried only when needed. It can store a varied amount of formats in big data ecosystems, from unstructured, semi-structured, to structured data sources.*"

Due to the progressive broadening of its definition and to the blurring of the architectural borderlines, the term DL is often replaced by the more general term *data platform* [5, 6] or even *data ecosystem* [7, 8]. Though these terms were previously used with an emphasis on the enabling technology, they are now revived to encompass systems supporting data-intensive storage and computation, as well as analysis of data with varying structure.

Today cloud data platforms, such as Google and Amazon, provide several tools for data ingestion, transformation, and analysis. These tools relieve users from the technological complexity of administration by providing additional services that enable companies to focus on functional aspects. What is still lacking is a smart support to govern the complexity of data and their transformations. Data transformations and analyses must be governed to avoid that the data platform turns into a *data swamp*: in fact, without descriptive metadata and appropriate mechanisms to maintain them, the analysts will soon lose control over their data, and they will have poor support in reusing their past processing workflows (which means they will have to start from scratch every time they need data). These problems are further amplified in the age of data science, which sees data scientists prevail over data architects —the former needing more assistance to avoid drowning in the lake. In the following,

*Corresponding author

Email addresses: m.francia@unibo.it (Matteo Francia), enrico.gallinucci@unibo.it (Enrico Gallinucci), matteo.golfarelli@unibo.it (Matteo Golfarelli), annagiulia.leoni2@unibo.it (Anna Giulia Leoni), stefano.rizzi@unibo.it (Stefano Rizzi), nicola.santolini2@unibo.it (Nicola Santolini)

we will use the terms data lake and *Big Data Platform* (BDP) as synonyms.

In this paper we propose MOSES, a *technology-agnostic*, *extensible*, and *customizable* framework for metadata handling in BDPs. The framework hinges on a metadata repository that stores information about the objects in the BDP and the processes that transform them. MOSES is specifically addressed at supporting data search/profiling, provenance, evolution, and orchestration; its main features are outlined below.

- MOSES is *technology-agnostic*, i.e., it does not rely on any specific technology. Though this prevents MOSES from taking advantage of the specific features of one technology or the other, it makes MOSES a general-purpose framework. Thus, although the framework has been developed and tested on the HADOOP ecosystem, its architecture and functionalities can be applied to any BDP.
- The metamodel of MOSES is inherently *extensible*: it can store (both structural and semantic) metadata at different levels of detail and precision, and describe user-defined properties depending on the type of objects treated and the type of processes executed. Symmetrically, new metadata extraction algorithms can be easily plugged in. In this way, MOSES can be adapted to capture and analyse metadata in different domains, and for different purposes.
- MOSES is *customizable* since it can be configured to monitor and collect only the metadata to support a subset of functionalities; as a consequence, the computational effort needed depends on the functionalities required. This feature makes MOSES suitable both for simple BDPs running on small clusters and for very large and complex ones.

A distinguishing feature of MOSES is that it supports both *push* and *pull* metadata updates. We claim that, to be effective and general, metadata updates must be semi-automatic: if a wrapper is available it can be plugged in MOSES, which will collect metadata from data in pull mode; conversely, if no wrapper is available for some data type, the application processes that handle the data can update metadata in push mode.

Although some other approaches to intelligent BDPs have been proposed so far in the literature, most of them have at least one of the following drawbacks: (i) they are described at a high level of abstraction, i.e., they are more focused on discussing challenges and desiderata rather than on proposing solutions, or they just sketch a high-level framework; (ii) they are strictly coupled to a single technological framework (e.g., Spark, Map Reduce, Google); (iii) they are focused on a specific service such as data provenance or data versioning. Without the ambition to be exhaustive in managing all types of metadata and all their possible uses, the originality of our work lies in proposing, designing, implementing, and testing on a real case study a general-purpose framework for the collection, management, and exploitation of data in a BDP.

The paper outline is as follows. After discussing the related literature in Section 2, in Section 3 we give an overview of the system and its architecture. Section 4 describes the MOSES metamodel. Sections 5 to 8 present the main MOSES functionalities; while Section 5 explains how metadata are extracted from data, the others focus on the three main classes of functionalities, namely, metadata search and profiling, data provenance, and process orchestration. Section 9 focuses on the implementation of MOSES we adopted for our real case study, described in Section 10. Finally, Section 11 draws the conclusions.

2. Related work

2.1. Architectures

The DL architecture describes how data are conceptually and physically organized. A recent review of the various DL architectures proposed in the literature can be found in [1]. The main two variants considered are *zone architectures* and *pond architectures*.

In zone architectures, data are assigned to a zone (also called *area*) according to the degree of processing that has been applied to them; thus, they can be available in multiple zones at different degrees of processing at the same time. An example of zone architecture is the one proposed in [9], which distinguishes a *refined data zone* (where data and structured and integrated), a *trusted data zone* (where data are cleaned), and a *discovery sandbox zone* (where data are stored for exploratory analyses). In [10], a zone architecture based on a *raw data zone* (unprocessed ingested data), a *process zone* (processed and transformed data), an *access zone* (data ready to be accessed for analytics), and a *govern zone* (ensuring data security, quality, life-cycle) is proposed.

In pond architectures, data are only available in one pond at any given time; as data flow through the ponds, they are transformed depending on the pond they currently belong to. An example of pond architecture is proposed in [11] for a DL that stores power grids data. It relies on a *raw data pond* (including mostly unprocessed data), an *analog data pond* (where reduction techniques are applied to analog signals), an *application data pond* (data here are stored in the form of a database), a *textual data pond* (storing unstructured data), and an *archival data pond* (data seldom used for analyses). Interestingly, in [10] pond architectures are recognized to contradict the definition of DL because they do not ensure the availability of all the raw data.

A further architecture reviewed in [1] is the *lambda architecture*, where incoming data are copied to two different branches, one where they are stored permanently and periodically processed in batches, one where they are processed in real-time.

The metamodel of MOSES supports a zone architecture based on a set of DL areas. Section 9.1 lists the specific zones adopted in the implementation used for our case study.

2.2. Metadata

Metadata capture information on the actual data, e.g., schema information, semantics, and provenance. They ensure that data can be found, trusted, and used [1].

A DL architecture specifically devised for power grid monitoring and diagnosis is described in [11]. Metadata are extracted at the ingestion stage to be used for data classification, governance, and protection. The KAYAK framework is proposed in [12] to support users in creating and optimizing the data processing pipelines in DLs. Metadata are extracted from ingested datasets using predefined profiling tasks, then they are collected in a catalog. Besides *intra-dataset* metadata, which describe a single dataset in statistical and structural terms, also *inter-dataset* metadata are stored to capture different relationships among datasets and enable advanced analytics. The same metadata classification is adopted in [10]; specifically, intra-dataset metadata here also include lineage, quality, and security metadata. In [13] a third class of *global* metadata is added to provide a contextual layer to the data and optimize their analysis.

The GEMMS system [14] automatically extracts metadata from a wide variety of data sources. These metadata are represented using an extensible metamodel that includes both *structural* and *semantic* metadata, and can be automatically extracted from the ingested datasets via specific extractor components. Metanome [15] is an extensible platform for data profiling based on the automatic discovery of complex metadata. Besides basic statistical metadata, also more complex metadata such as inclusion and functional dependencies are discovered. In [2], besides *schematic* metadata (e.g., data format and source) and *semantic* metadata (e.g., keywords and categorization), the authors introduce *conversational* metadata (e.g., who has used the data and where did they find real value in data) as a way to keep track of the users' experiences with the data.

The Goods system [16] collects metadata about each dataset and its relationship with other datasets, to make them available to users for better organizing, monitoring, and searching data. Besides crawling Google's storage systems to extract basic metadata, Goods performs metadata inference —e.g., to determine the schema of a non-self-describing dataset, to trace the provenance of data through a sequence of processing services, or to annotate data with their semantics. Ground [17] is a data context service encompassing both *applications* (how data are interpreted for use, corresponding to object types in MOSES), *behavior* (how data were created and used, corresponding to operations and agents in MOSES), and *change* (the version history of data, corresponding to objects and their versions in MOSES). The authors give design guidelines, describe some key services —such as ingestion, search, authentication, and scheduling— and propose a common metamodel based on three layers: *version graphs* (to represent data versions), *model graphs* (to represent application metadata), and *lineage graphs* (to capture usage information).

2.3. Processing

As already mentioned, it has been recognized that a DL should also encompass a comprehensive set of services to process data and make them usable by various analytical applications.

Metadata extraction has been widely covered in the literature, starting with the early work on schema discovery for XML files first [18, 19, 20], and then JSON files [21, 22, 23, 24, 25]. More recently, schema discovery has been coupled with semantic enrichment (e.g., in [26]). The approaches proposed are either manual, or automated, or a mix of these two.

Data ingestion is mentioned by most papers in the field (e.g., [27, 17, 28, 26]). In [28], a distinction is made between ingestion and *data collection*. Other services often considered are related to data search/querying/exploration/analysis [27, 17, 28], quality management [27, 10], authentication/authorization/security/auditing [17, 11, 10], and visualization/presentation [28]. Real-time analyses and cleaning/integration are explicitly mentioned in [28, 4] and [26, 28], respectively. User collaboration is considered in [29]. In [17] the authors also mention scheduling, workflow, and reproducibility as basic services.

Version management is considered as a crucial service in several papers, e.g., [17, 26]. According to [26], versioning is a cross-cutting concern over all stages of a DL since new datasets and new versions of existing files enter the DL, and extractors can evolve over time and generate new versions of raw data. In [17], DAGs of object versions are maintained to ensure flexible version management of code and data, general-purpose model graphs, and lineage storage.

A lot of attention has been devoted to *provenance* in the context of DLs. To track provenance in big data applications, in [30] the authors propose a reference architecture based on an extension of the Kepler provenance data model. The goal is to record provenance and use it not only to reproduce workflows, but also to predict their performance. Another architecture, which uses a central provenance collection subsystem to track provenance, is proposed in [31]. Titian [32] is a provenance framework designed for data-intensive scalable computing systems; it provides scalable fine-grained provenance capturing, interactive provenance query capabilities, and an API to seamlessly move between provenance and data records.

All the services mentioned above rely on a proper *metadata management*, covering metadata discovery/extraction, annotation, and querying [15, 27, 14]. The term *data governance* is often used to encompass several functions ranging from lineage, security, and quality to metadata management [33]. Finally, *data wrangling* has been defined as the set of processes including creating, filling, maintaining, and governing a DL, aimed at obtaining a *curated data lake*, meant as a DL whose content is made accessible to users beyond the enterprise IT staff [2]. Noticeably MOSES offers services in all areas mentioned above.

2.4. Comparison with MOSES

The analysis of the literature has confirmed the relevance of metadata modeling and the interest of the scientific community; at the same time it has shown that most efforts were focused on discussing the research challenges and providing high-level architectures and metamodels, so only a small number of proposals apparently reached a prototypical stage. Below we discuss

the differences between MOSES and the proposals that explicitly describe an implementation:

- **Constance** [27]: this is a high-level proposal and few details are given on metamodel and functionalities. No metadata on operations are collected. The system has not been tested on real case studies.
- **GEMMS** [14]: the proposal mainly focuses on metadata extraction; no discussion is made about the functionalities provided for metadata exploitation. Metadata concerns objects only, and no information is collected about operations and agents.
- **Goods** [16]: it is described by their authors as a system to index the DL of all Google datasets. Differently from MOSES it uses fine-grained metadata, whose handling requires a significant computational effort. The system is strictly coupled with the Google platform and mainly focuses on object description and searches. No formal description of the metamodel is reported.
- **Ground** [17]: although it is described as an open-source system able to capture the full context of data, not enough details are given to clarify which metadata are actually handled. The provided functionalities are described at a high level as well. Apparently, the system has not been tested on real case studies.
- **KAYAK** [12]: the proposal mainly focuses on the pipeline optimization functionality, so only goal-related metadata are collected.
- **Titian** [32]: it mainly focuses on supporting data provenance in data-intensive computing systems and is strictly coupled with Apache Spark. Titian provides fine-grained provenance, and only the related metadata are collected. No tests on real case studies are described.

Finally, we briefly mention the commercial (and proprietary) solutions that exist for metadata management in a DL. Gartner[1] indicates Informatica's CLAIRE system as the market leader in this sector. Although some of its functionalities overlap with MOSES (as it appears from the white papers published on the website), no scientific nor technical documentation is available to explain how it is implemented. As to Cloud service providers, Google[2] and Microsoft[3] only offer a basic Data Catalog service to manually associate custom key-value metadata to DL files and tables. Conversely, Amazon offers AWS Glue[4], which also supports metadata extraction in pull mode. Lastly, Databricks's Delta Lake[5] is tightly coupled with Apache Spark, thus being able to automatically keep track of data evolution and transformations that are carried out within such environment.

[1]`https://www.gartner.com/en/documents/3993025/magic-quadrant-for-metadata-management-solutions`

[2]`https://cloud.google.com/data-catalog`

[3]`https://azure.microsoft.com/en-us/services/data-catalog`

[4]`https://docs.aws.amazon.com/whitepapers/latest/building-data-lakes/data-cataloging.html`

[5]`https://delta.io`

3. System Overview & Architecture

MOSES is meant to support the work of three broad categories of agents:

- *Data Scientists*: the *real* users of the data. They perform analyses, run algorithms, and display the results. MOSES mainly supports them in selecting data and understanding their characteristics.
- *Administrators*: all users who contribute to the management of the DL, such as security managers, technicians, and DB administrators. MOSES supports them in monitoring the activities of data scientists and in keeping the BDP and its processes under control.
- *Processes*: BDP processes may rely on metadata to be triggered or to trigger a task or a command.

Overall, MOSES helps the users and their processes to operate efficiently and effectively without turning the BDP into a data swamp. We can classify the MOSES functionalities into three main groups, which will be described in Sections 6, 7, and 8, respectively:

- *Object profiling and search*: includes all the functionalities aimed at searching, selecting, and describing the objects stored in the DL. Descriptions range from simple object properties (e.g., creation date) to complex schema descriptions. Searching is based on properties, and the search criteria can be organized according to a complex syntax.
- *Provenance and versioning*: includes all the functionalities aimed at describing, at different levels of detail, the object transformations and relationships.
- *Orchestration support*: includes all the functionalities used by the orchestrator. For example, a workflow can be triggered or tuned based on an object property stored in the metadata repository.

Figure 1 shows the functional architecture of a BDP empowered with MOSES. The components of MOSES are those in the rightmost part of the figure (in orange), the others are standard components in charge of producing/consuming, processing, storing, and visualizing data. The orchestrator (e.g., Oozie in the Hadoop ecosystem) is shown separately due to its importance in managing the data transformation processes.

MOSES includes a repository aimed at storing all the metadata collected, plus an extensible set of components that support the MOSES functionalities. A crucial component is the *Metadata Extractor*, in charge of automatically feeding the metadata repository (Section 5). The *Metadata Search Engine* supports complex queries on the metadata repository (Section 6). Other important components are the *Provenance Manager*, which traces the transformations of the BDP objects (Section 7) and the *Trigger Engine*, which trigger events on metadata changes (Section 8). The "other components" box is a placeholder to

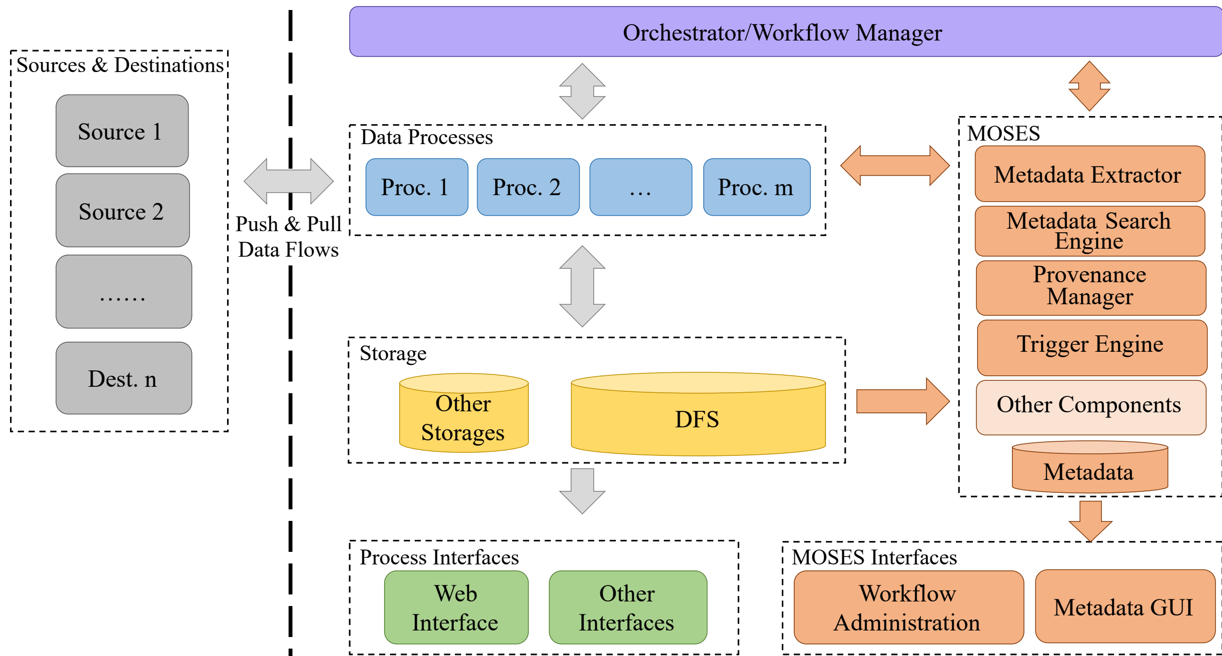

Figure 1: Functional architecture of MOSES

emphasize the possibility of extending the system with additional features (as discussed in the conclusions). MOSES is completed by a set of ad-hoc interfaces related to the functionalities implemented. Gray arrows show generic data flows occurring within the DL, while orange arrows show the metadata flows related to MOSES. The control flows between the Orchestrator and the functional processes are not represented in the figure.

As to orange arrows, the metadata flows entering MOSES from the data processes and the Orchestrator mainly consist of push notifications (e.g., the creation of a new object, the parameters of an operation, or a change of state in a process). As already mentioned, push notifications are necessary to maintain MOSES technology-agnostic. Whenever a wrapper for an application/process is not available in MOSES, the application/process itself must feed the metadata repository. The outgoing flows sent from MOSES to the Orchestrator and to the data processes carry the metadata needed by the latter to operate (e.g., the Orchestrator triggers a transformation when a new object has been detected by MOSES). Finally, incoming metadata from the storage system are automatically extracted from the BDP objects by the Metadata Extractor in pull mode.

4. Metamodel

The metadata repository is the core of MOSES since it defines its potentialities. In this section we initially describe the different concepts, then we justify our modeling choices.

The metamodel is shown in Figure 2; its concepts, described below, can be organized in three main regions:

- *Object region*: includes, for each object, the information that can be extracted without analyzing its content (i.e., the object is considered a black box).
  - Object: a data item being stored and processed (e.g., a table or a file). An object may aggregate other objects (e.g., a relational database and its tables), and it may be related to a previous version of the same object (e.g., in case that object has been updated).
  - Object type: the semantic category of the object.
  - Data lake area: the area of the DL each object is located in (in some cases, an object may not be stored in any area, e.g., a database).
  - Project: the project(s) each object belongs to.
- *Operation region*: includes the information related to the transformations the objects went through.
  - Operation: any processing operation executed in the BDP.
  - Operation type: the process whose execution is modeled by the operation.
  - Source: a website, web service, or organization that provides data.
  - Agent: the subject that activated the operation, it can be either a human or a process.
- *Schema region*: includes the information collected by analyzing the object content.
  - Schema: an object may have one or more schemas, each consisting of a set of attributes.
  - Attribute: a single attribute of an object schema.
  - Domain: provides a semantic description of an attribute and indicates the set of values that the attribute can take.

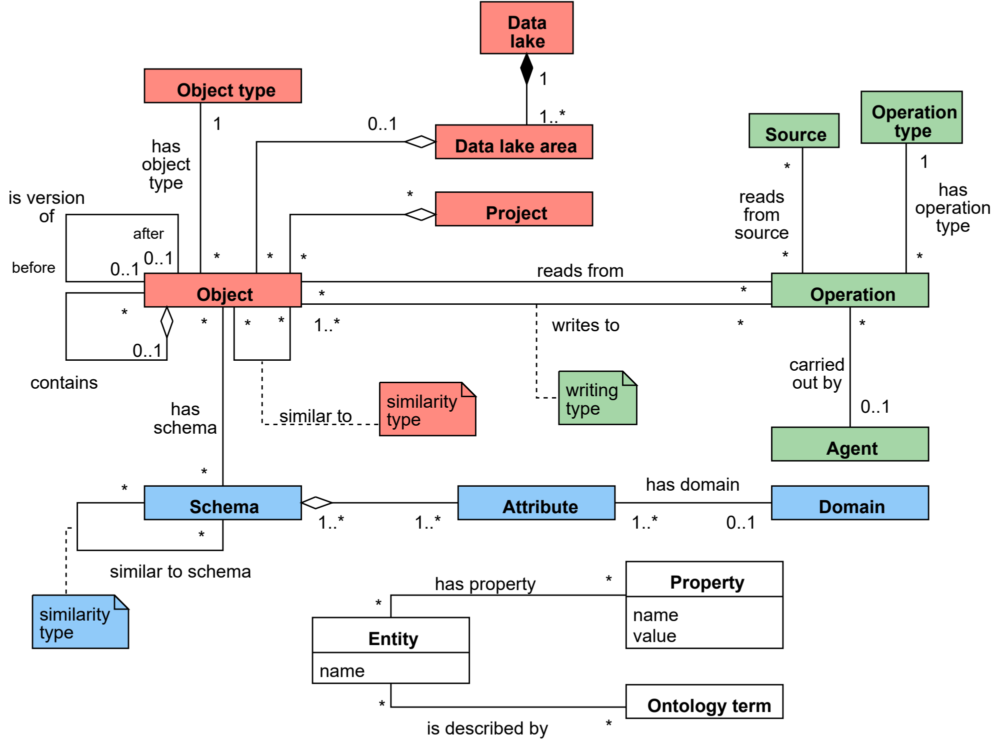

Figure 2: UML representation of the MOSES metamodel (for simplicity, we omit the generalizations from all classes in the upper part of the figure to class Entity); the red, green, and blue background reflects the organization into object, operation, and schema regions, respectively.

Importantly, all these classes specialize class Entity, so they all have a variable set of properties. Some commonly used properties are summarized in Table 1. The writes to association has a property as well, named writing type, which can take values "update", "append", "create", and "overwrite".

Example 1. *Consider a JSON file containing customer data ingested into the DL from an e-commerce application. The file is modeled as an* Object *with "Customer data" as* Object type*, stored in the "Landing"* Data lake area *of* Project *"Customer profiling". As the JSON documents within the file are heterogeneous, the object is linked to several* Schema *instances (one for each JSON schema found within the documents), each linked to its* Attributes*. Some attributes are linked to a* Domain*; for instance, "fiscalCode" is linked to the "ItalianFiscalCode" domain since its value (e.g., "RSSLCU80A01H501K") matches with the regex of the latter. The object is also linked to the* Ontology term *"dbpedia.org/ontology/person". While schema information is pulled by the Metadata Extractor, the relationship to the ontology term is pushed by a user. Now, let data scientist Alice launch a Python script to clean and pre-aggregate the data in the aforementioned object. The script is referenced in the metamodel as an* Operation type *and the actual execution is referenced as an* Operation*, which reads from the previous object and writes to a new one (with* writing type *"create"). Figure 3 shows the UML representation of this example.*

In the following we list some considerations that motivate our choices in defining the metamodel:

- Since MOSES has been conceived to be domain-independent and extensible, we do not define a priori the set of instances of Object type, Operation Type, and Domain.

Table 1: Main properties for some entities

Entity	Property	Sample values
Object	category	file, table, database
	path	hdfs://landing/IMG_20190926T.tif
	format	csv, xml, tif
	size	11 MB
	storage	hive, hbase, neo4j
	version	1.0
	quality	3.5
Attribute	datatype	integer, string
Domain	reg. exp.	{^([a-zA-Z0-9]+)$} (an alphanumerical string)
	values	[female, male]
Operation	timestamp	
	parameters	[id = value1, title = value2]

- Since MOSES has been conceived to be customizable, the level of detail with which the metadata are collected determines the trade-off between the level of detail of the functionalities and the computational effort required to keep track of the events occurring in the BDP. Customization is not only achieved through the generic Property class, but also through the level of detail at which objects can be described. For example, a database can be traced

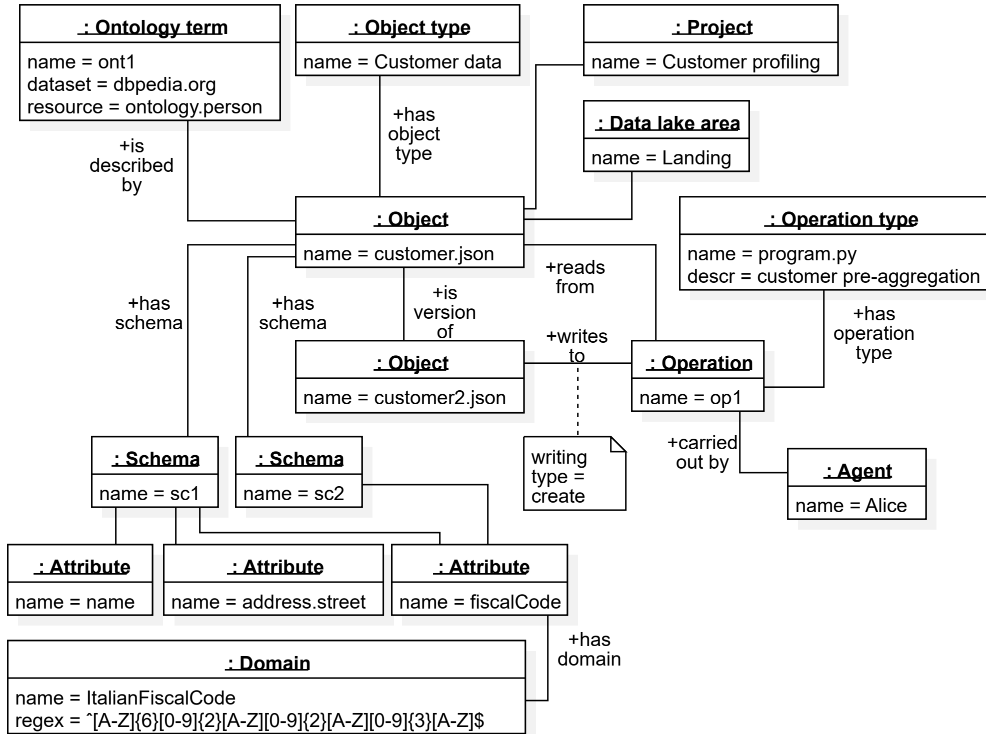

Figure 3: UML object diagram corresponding to Example 1 (for drawing simplicity, entity properties are represented as attributes)

as an atomic repository or by distinguishing the single tables it includes.

- MOSES supports multi-zone, multi-project and multi-tenant BDPs. While the first two dimensions are explicitly modeled through Data lake area and Project entities respectively, objects' owners can be stored as properties of the Project entity.

- The Domain entity is meant to provide a semantic description of an attribute. This is far more expressive than the simple datatype information (e.g., integer, string), which is indicated as a property of the Attribute entity (see Table 1). Noticeably, the idea of "domain" does not appear in other approaches in the literature on DLs, where only mentions of linking to other ontologies and to "proprietary knowledge graphs" can be found.

- The concept of schema changes radically based on the type of objects stored (e.g., database, JSON document, CSV files) and the level of precision with which they are described (e.g., a database as a whole vs. single database tables). Since BDP data can also be schemaless, we provide a schema description for every single object rather than for a generic object type like in [25].

- To have a complete understanding of data flows, external sources must be monitored too. In MOSES, external data sources are explicitly modeled by the Source entity.

Remarkably, the extensible and general-purpose nature of MOSES pushes the expressiveness of its metamodel beyond the one offered by related approaches, especially those focused on specific services only (e.g., [12, 32]) and those that provide a fixed grain of metadata (e.g., [27, 14]).

To evaluate the coverage of our metamodel from a functional point of view we refer to [13], which classifies metadata into *intra-object* (that describe a single object), *inter-object* (that relate multiple objects), and *global* (that store information about the whole BDP and are not strictly related to a specific object, e.g., an event taking place in the BDP independently of a specific object). MOSES metadata are either intra- or inter-object; we do not consider global metadata since our focus is on BDP objects.

In [13] metadata are then related to functionalities. Those covered by MOSES are:

- *Semantic enrichment*, which consists in generating a description of the context of data (e.g., with tags) to make them better interpretable and understandable. This is one of the main goals of MOSES. Single objects or groups of objects are characterized with both structural and ontological tags: each Entity of the metamodel optionally is described by an Ontology term. In the current implementation, ontological tagging is carried out manually, i.e., no component has been developed for automatic tagging.

- *Link generation* is the process of detecting similarities between objects. In MOSES, similarity is expressed both at the object and at the schema levels. The metamodel does not intentionally code the way similarity is computed, thus different semantics can be adopted.

- *Data versioning* refers to the ability of the metadata to keep track of data changes. The classes and associations used to this end are is version of, reads from/writes to, and Operation.

- *Usage tracking* records the interactions between users and the objects. This group of functionalities is basically enabled by class Agent, which specifies which user created/deleted/transformed a given object or run a given operation.

Noticeably, the only group of functionalities in [13] not covered by MOSES is *data indexing*, which consists in setting up a data structure to retrieve datasets based on specific characteristics (keywords or patterns).

Another functionality that is not explicitly supported by MOSES is data integration [34]. In fact, data integration is a very complex process, and a hardly-automatable one, especially for big data applications within a multi-tenant and multi-project context. However, the metadata collected by MOSES include information (e.g., the similar to schema property and Domain entity) that can be used to guide a project-specific data integration process.

Finally, while the grain of metadata is customizable and extensible, MOSES does not reach the high level of detail used in single vertical applications [35, 36], as the latter requires the collection of huge amounts of metadata. Conversely, MOSES favors a more coarse-grained approach, which is more appropriate for a general-purpose framework aimed at covering highly-heterogeneous datasets and transformations.

5. Feeding the Metamodel

Precondition to a powerful metadata exploitation is an effective metadata feeding strategy. MOSES implements both pull (i.e., MOSES-driven) and push (i.e., application-driven) modes. While the mostly-static entities (such as Object type, Operation type, and Data lake area) are manually fed, for dynamic entities an automatic approach is mandatory. In all cases, the metamodel is fed through a set of standard APIs as described in Section 9.2.

Defining a pull strategy requires defining *when* MOSES actively collects metadata and *what* metadata are collected. MOSES actively monitors only the changes that take place on the file system (i.e., insert/update/delete of objects), while the information related to the operation region must be pushed into MOSES by the functional processes. Within a highly heterogeneous computing environment, monitoring the different functional processes would mean monitoring several different computing engines; besides being practically infeasible, this would violate the requirement of being technology-agnostic. Note that adopting a mix of feeding strategies on the one hand enlarges the quantity of metadata acquired by MOSES and respects its basic philosophy, on the other hand it makes metadata on a given object possibly incomplete. More in detail, the metadata in the object region are available for all objects, those in the operation region are present if the functional process that carried

Algorithm 1 Extractor

Require: $\mathcal{MD}$: the current version of the metadata repository, O: a new object
Ensure: $\mathcal{MD}$: the updated version of the metadata repository
1: $\mathcal{MD} \leftarrow \mathcal{MD} \cup \mathsf{CollectObjectRegionMD}(O)$ ▷ Collect metadata of the object region and update the metadata repository
2: $wr \leftarrow \mathsf{SpecificWrapper}(O)$
3: **if** $wr = null$ **then**
4: $\quad wr \leftarrow \mathsf{ApplicableWrapper}(O)$
5: **if** $wr \neq null$ **then**
6: $\quad \mathcal{MD} \leftarrow \mathcal{MD} \cup wr.\mathsf{CollectSchemaRegionMD}(O)$ ▷ Collect metadata of the schema region and update the metadata repository
7: **return** $\mathcal{MD}$

out the operation pushed them to MOSES; finally, the metadata in the schema region are present only if an appropriate wrapper is available.

The pull mode is implemented by the Metadata Extractor (see Figure 1), which initially determines if an appropriate wrapper is available for the object. MOSES extracts different metadata depending on the availability of an applicable wrapper and on its specificity. The applicability of a wrapper is driven by rules (e.g., the file extension or the combination of the DL area and the project). If no specific wrapper is identified, all the available ones can be tried to verify their applicability. The wrapper applicability depends on the number of metadata properly interpreted [27]. In the worst case no applicable wrapper is found; the object is considered as a black box and only the information related to the object region are collected. In other cases, several alternative wrappers can be adopted. For example, a satellite image produced by ESA has tiff extension; although the standard tiff tags (e.g., the band list) can be extracted by a generic tiff wrapper, only a geotiff wrapper can extract the geographical information (e.g., geographic boundaries). Similarly, a JSON file can be analysed either through a generic JSON parser or through the one for a specific type of content [14]. Algorithm 1 provides a more formal description of the steps described so far when a new object is detected.

In the following, we give an insight into how the Metadata Extractor identifies different information. Noticeably, each information can be modified via a push notification by a user or a functional process:

- *Object region*:
  - Object: the creation/update/deletion of objects is notified to the Metadata Extractor by the BDP.
  - Object type: the object type an object belongs to is determined through a pattern-based approach; typical patterns involve the file extension or a specific format of the file name [27, 16].
  - Data lake area: the DL area an object belongs to is determined through the folder storing the object.

- Project: the project an object belongs to is determined through the folder storing the object.
- is version of: an instance is created if the BDP notifies an update.

- *Schema region*:
  - Schema: determined by parsing the object content.
  - Attribute: determined by parsing the object content.
  - Domain of an attribute: inferred by CollectSchemaRegionMD either by matching the attribute name with other attributes with a known domain or by verifying the compatibility of the attribute value with the values allowed by the domains. In particular, we check whether the value matches the regular expression associated with the domain if available, otherwise if it matches the sample values associated with it.
  - similar to schema: measured by scoring (i) the attributes that are present in both schemas [37] and (ii) those in one schema whose domain is used in the other schema.
  - is similar to: an instance of this association is created if the similarity exceeds a threshold. The similarity is computed by scoring the similarity of schemas (i.e., similar to schema) included in the object.

As to semantic metadata, each entity in the metamodel can be annotated with a set of ontology terms (see Figure 2). This annotation can be done either incrementally (by training a model after some manual linking) or by applying some entity linking technique. For instance, in Analyza it is done either manually or by applying some heuristics [29], while in GEMMS it is done manually [14]. Constance [27] is another approach that, in addition to extracting metadata from sources, enriches data sources by annotating data and metadata with semantic information.

6. Object profiling and search

Access to metadata is expected to be complex and varied, ranging from basic search functionalities to more sophisticated ones. Indeed, discoverability is considered to be a key requirement for data platforms, and it should cover both simple searches to let users locate "known" information and data exploration to let them uncover "unknown" information [38].

Search functionalities in MOSES operate at the metadata level, to let the user extract relevant information from the metamodel. Direct querying of the data could be enabled by introducing support for different connectors and querying languages. However, the focus in MOSES is on helping the user unravel the chaos in the DL, which first of all requires to enable search at a higher level. Additionally, object profiling aims at providing an abstract view of a certain object (or a group thereof) aimed at summarizing its characteristics and relationships with other objects.

Table 2: Supported search functionalities with description and targeted agent (Data Scientist, Process, Administrator)

Functionality	Description	Agent
Basic search	Search entities by structural properties	DS, P, A
Schema-driven search	Search objects by their intensional features	DS, P
Provenance-driven search	Search objects based on the history of operations they were subject to	DS, P
Similarity-driven search	Find objects that can be either joined or unioned	DS, P
Semantics-driven search	Find entities based on their semantic properties	DS, P
Profiling	Summarize the most important characteristics of an object (or a group thereof)	DS, A

The search-related functionalities supported by MOSES are summarized in Table 2 and briefly discussed below. The whole metamodel is open to querying; thus, we devise different functionalities depending on the concepts involved in the search, where the common goal is the identification and description of Objects.

1. *Basic search.* In the simplest scenario, the user looks for some objects by means of their static properties (e.g., their names, paths, sizes, etc.) or their assigned Object types, Projects, or Data lake areas. The focus of this search can range from a wide spectrum (e.g., searching for every object of a certain project) to a narrow one (e.g., searching the landing area for small objects that contain a certain string in their name).

2. *Schema-driven search.* MOSES supports the search for objects based not only on their simple properties but also on their intensional features. This is especially useful when the user explores the DL to discover objects that conform to a specific Schema or that contain information referring to a given Domain.

3. *Provenance-driven search.* Exploiting provenance metadata means finding objects by navigating the history of Operations. Such navigation can be carried out in both directions, i.e., either to discover all the objects obtained from a certain *ancestor* or to track down the object(s) from which another has originated. The latter case is also referred to as the discovery of *canonical* datasets, i.e., reference datasets from which several derivations have been obtained [39].

4. *Similarity-driven search.* The similar to arc in the metamodel enables the discovery of objects based on their similarities. On the one side, this is useful to discover datasets to be included in a certain query, either by merging objects (if they contain the same kind of information in different forms) or by joining them (if their similarity identifies a foreign key-primary key relationship); this is similar to the usage of joinability and affinity relationships in [37]. On the other side, similarity search is useful to group similar objects and enrich the search results, either by providing a diversified result that lists the main

objects from each group or by restricting the search to the objects of a single group.

5. *Semantics-driven search.* The semantic metadata in MOSES further enrich the search capabilities of objects. In particular, the Ontology terms can be exploited to search objects without having any knowledge of their physical or intensional properties, but simply exploiting their traceability to a certain semantic concept. Clearly, the expressiveness of the previously described search capabilities can be extended by coupling them with semantic ones.

6. *Profiling*. Object profiling aims at providing an abstract view of a certain object (or a group thereof) that summarizes its characteristics. When looking at single objects, the profile shows its properties and lists the relationships with other objects, both in terms of similarity and provenance. A profile can be generated also for groups of objects (based on some grouping feature, e.g., the Object type); in this case, the profile also adds a representation of the intensional features that mostly characterize such group [25].

The different search functionalities are supported by a multi-level API layer (see Section 9.2) that translates the users' requests into queries on the metadata; this ensures also systematic access to the metadata from any software tool. Remarkably, search-related functionalities serve as a building block for provenance- and orchestration-related functionalities (see Sections 7 and 8), as the latter ultimately consist of more extensive and expressive metadata searches.

7. Provenance and versioning

Data provenance[6] (or simply provenance) is metadata pertaining to the history of a data item (an object in our metamodel) [41]. Essentially, it describes a transformation pipeline [42], including the origin of objects and the operations they are subject to [30]. Since its introduction for database systems [41, 35, 30], the relevance of provenance has been well understood also in other fields [43] due to the growing need for automated data-driven applications [44].

Following the systematic organization of provenance features in [41], we categorize the provenance in MOSES as coarse-grained and data-oriented since it focuses on storing Operations applied to Objects. Fine-grained provenance is typically used for single vertical applications [35, 36] since it requires to collect huge amounts of detailed information to enable a very detailed tracing. Conversely, coarse-grained provenance is appropriate to ensure a broad coverage of highly-heterogeneous transformations possibly involving several applications and datasets. The grain of provenance data in MOSES is actually customizable and extensible, depending on the set of properties that are captured. Choosing a granularity is the result of a trade-off between accuracy and computational effort. For example, storing only the name and the version of a clustering algorithm enables an approximate reproducibility of the results, while storing all its parameters makes this functionality much more accurate. The granularity of objects is customizable as well since MOSES can metamodel a database as a whole or even store a separate object for every single table.

[6] *Data provenance* and *data lineage* are used in the literature as synonyms or with slightly different flavors. Henceforth, we will consider them as synonyms and refer to provenance as in [40].

Table 3: Supported provenance functionalities with description and targeted agent (Data Scientist, Process, Administrator)

Functionality	Description	Agent
Data quality	Assessing data degradation over pipeline/time	DS, P
Debugging	Finding issues with operations	DS
Reproducibility	(Partially) reproducing a pipeline	DS, P
Trustworthiness	Assessing the reliability of data sources and agents	DS
Versioning	Marking stable versions of data	DS, P

As in [31], MOSES adopts an integration approach; indeed, its APIs can capture provenance events from individual components of the DL. Each transformation adds a new Operation between two Objects carried out by one Agent. Though we mainly rely on a push approach (i.e., applications send transformation events to MOSES through its APIs), a pull approach could be pursued as well for specific engines. For example, the parameters adopted in a machine learning computation can be easily extracted from the output file through the Metadata Extractor.

The provenance-related functionalities supported by MOSES are summarized in Table 3 and briefly discussed below:

- *Data quality*. MOSES supports the monitoring of the quality (e.g., accuracy, precision, recall) of the objects produced, to notify the data scientist when a transformation pipeline is not behaving as expected. In the case of garbage-in/garbage-out pipelines, new objects are generated with lower accuracy than the previously generated ones.

- *Debugging*. Inferring the cause of pipeline failures is challenging and requires an investigation of the overall processing history [45, 38]. To support this process, MOSES stores the inputs of each operation along with their versions and the environmental settings in which each operation is performed (e.g., RAM and CPUs).

- *Reproducibility*. MOSES enables the re-execution of all or part of the operations belonging to a pipeline. Given the objects and operations involved in a new pipeline, MOSES can determine which operations of precedent pipelines have to be executed again. For instance, training multiple times a machine learning model on a dataset does not require re-executing the ETL that stored the dataset.

- *Trustworthiness*: MOSES helps data scientists to trust the objects produced by tracing them back to their sources

Table 4: Supported orchestration functionalities with description and targeted agent (Data Scientist, Process, Administrator)

Functionality	Description	Agent
Dynamic/condition-based behavior	Deciding *what* computation to run based on different metadata conditions	P
Triggering	Deciding *when* to run the computation based on the presence of some metadata	P
Scoping	Deciding *if* the computation must be run, based on data trustworthiness	DS, P
Resource estimation/prediction	Deciding *how* to run the computation in terms of resources needed	DS, P, A

and storing the agents who operated on those objects. Indeed, some data sources and agents might be more reliable than others.

- *Versioning*: by marking a generated object and its consequent versions (due for instance to changes in a database schema), MOSES helps data scientists in identifying relevant/stable objects along with their semantic versions, and to operate with legacy objects.

To enable the collection of the metadata required by the above functionalities, we exploit the compliance of our metamodel with the PROV standard [46]. PROV formalizes provenance as a directed acyclic graph, whose nodes correspond to Objects or Operations in Figure 2. To make the graph acyclic, every Operation has to write a new Object. This enables a standard integration with existing provenance tools for the querying and visualization of provenance metadata. Note that all the functionalities described so far can be easily implemented by querying the PROV graph.

8. Orchestration support

In a big data architecture, the system orchestrator (or simply *orchestrator*) is the component in charge of controlling the execution of computation activities (or data processes, as we call them in Figure 1). The orchestrator ensures the activation and execution of data processes in an automatic manner, either through a regular scheduling of the activities or by triggering a process in response to a certain event. Besides the orchestrator, several other entities (either processes or human beings) can cover this role to activate some data processes in the BDP.

MOSES supports orchestration activities not simply in a passive manner (because the activated data processes will prompt metadata updates), but it is also able to actively support and enhance such activities. In particular, the orchestration-related functionalities supported by MOSES are summarized in Table 4 and briefly discussed below.

- *Dynamic/condition-based behavior*. The variety of metadata collected about each object supports the orchestrator in deciding *what* data process should be activated under different conditions, and deciding how to tune the parameters in case of parametric data processes. For instance, when the orchestrator needs to run a transformation on a certain Object, the properties and the provenance of the latter can improve the decision as to the specific Operation type most suitable for such object.
- *Triggering*. The orchestrator usually takes into account several factors to decide *when* to trigger a certain data process. To this end, MOSES actively works as a trigger mechanism by implementing a messaging queue where, at any changes on a Entity, it pushes a notification. Furthermore, complex conditions determined by the combination of several events can be defined and pushed on the messaging queue too. The orchestrator, as well as any functional process, can subscribe to the queue and receive the triggering messages.
- *Scoping*. Before triggering a data process, it is essential for the orchestrator to assess the trustworthiness of Objects. By examining the properties and provenance of the latter, the orchestrator is able to understand where the data came from, who handled its collection or manipulation, and ultimately *if* a certain data process should be activated or not.
- *Resource estimation/prediction*. When activating a data process, the orchestrator can negotiate with the cluster's resource manager (e.g., YARN) the amount of resources to assign to such process. To this end, the history of previous Operations applied to the same Object or to objects of the same Object types supports the orchestrator in deciding *how* the data process should be activated, i.e., which is the optimal amount of resources required to terminate successfully while leaving sufficient resources to the other concurrent process.

9. Implementation

In this section we discuss the implementation of MOSES we are currently using on different projects; further details on the case study are provided in Section 10.

9.1. Technological stack

Figure 4 provides the technological details about our implementation of the functional architecture proposed in Figure 1. In this section, we discuss both the implementation choices for the MOSES core and the reference BDP in which MOSES is used.

MOSES is used within a BDP that builds on the Apache Hadoop ecosystem, which provides a solid environment with hundreds of tools that can work on top of it. The hardware architecture is a two-rack cluster of 18 Ubuntu machines, each with a minimum configuration of i7 8-core CPU @3.2GHz, 32GB RAM, and 6TB hard disk drives. At the software level, the cluster runs the Cloudera Distribution for Apache Hadoop (CDH) 6.2.0 and Docker; the former is a distribution that simplifies the installation, integration, and management of a Hadoop cluster, while the latter provides a virtual layer to deploy at runtime the framework (e.g., a Python environment) to run any kind of process (be it a MOSES component or a functional process).

The metadata collected and manipulated by MOSES are stored in Neo4J, an open-source graph database used in both

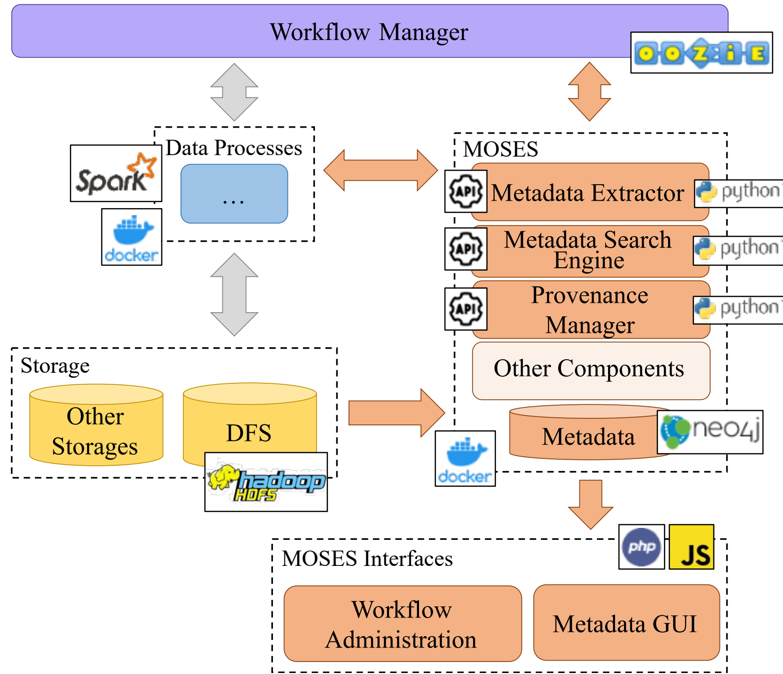

Figure 4: Technical architecture of MOSES

traditional and big data architectures. The graph data model is recognized in the literature as a unifying model for the representation of metadata of heterogeneous objects [16, 47, 48, 49], as it favors the modeling and storage of highly interconnected data and it easily accommodates the absence of fixed schemas (as shown by the metamodel presented in Figure 2, both data instances and relationships are extensible with a variable set of properties). The functional components of MOSES are mostly implemented as a multi-layered architecture of RESTful APIs (see Section 9.2); the only exception is the Metadata Extractor, which is implemented as a simple process. All components and APIs are implemented in Python. As to the MOSES interfaces, they are implemented as web applications in PHP and JavaScript. In particular, we rely on the D3 JavaScript library for visualizing metadata and interacting with them.

The DL is hosted on the Hadoop Distributed File System (HDFS) and is partitioned into several areas, representing the processing state of the data.

1. **Landing**: where data are stored immediately after ingestion.
2. **Working**: it contains temporary tables and configuration data.
3. **Harbor**: it contains data (completely or partially) processed from the landing area.
4. **Discovery**: it contains data created by data scientists for experimental purposes; it serves as a safe sandbox, where data is only taken from (and not published to) other areas.
5. **Access**: it contains data ready to be accessed by reporting and visualization tools.

This partitioning is a slight extension of the one proposed in [9]: the Landing, Harbor, Discovery, and Access areas respectively correspond to the Raw, Trusted, Discovery Sandbox, and Refined zones in [9]. The Working area differs from the Transient Loading zone in that the latter is only used in the ingestion of new data, whereas we use it as a general-purpose temporary area throughout the whole lifecycle.

Besides files on HDFS, we also employ some DBMSs to store data, depending on the different case studies.

The BDP also implements the orchestrator through Apache Oozie, a workflow scheduler system that manages jobs on the Apache Hadoop ecosystem. In particular, Oozie is used to schedule and control the execution of both functional and MOSES processes. The implementation of functional processes clearly depends on the specific project, but they typically consist of applications that either run directly on the underlying framework (e.g., an Apache Spark application), or exploit Docker to instantiate the required framework at runtime and then run the actual application.

9.2. API

MOSES metadata and functionalities are made available to users and processes through a set of RESTful APIs designed in a multi-layer architecture. APIs are exposed at different levels of expressiveness to decouple high-level calls exposed by public interfaces (e.g., to answer a provenance-related search on the metadata) from low-level calls that manage basic interactions with the metadata (e.g., to create or return a specific node). A subset of the implemented APIs is shown in Table 5 and is publicly available for testing on our case study at `http://big.csr.unibo.it/moses/`. In particular, the low-level layer is made by the following calls.

- *Single manipulation*. The /node and /relationship calls enable the retrieval/insertion of single elements from/into the metadata repository.
- *Transaction management*. The /transaction calls provide a mechanism to build complex sequences of atomic manipulations that need to be executed as a single transaction. The creation of a transaction returns an ID to be used as the reference trans_id not only to commit or rollback the transaction, but also as an optional parameter to create single elements.

The processes that need to push new metadata (e.g., the Metadata Extractor, or functional processes in general) often rely on the transaction management APIs to safely write the metadata via the atomic manipulation APIs. As to high-level APIs, a wide set of calls is defined to cover the functionalities described in Sections 6 to 8. For instance, the search call enables a wide, generic search on the entire metadata repository. It requires a JSON object that defines (i) the metadata classes that the user is interested in, and (ii) an optional set of disjunctive filters to be applied on any of the node or relationships available; the JSON object is then converted into a Cypher query to be executed on the metadata graph. The /metadata calls enable the addition of region-specific subgraphs of metadata and are used in both push and pull modes. The /prov calls allow users to obtain the provenance graph for a given node_id; the APIs enable the search to be conducted in either direction (forward or backward) and for a certain number of steps. Finally, the /orchestration calls return information useful to the orchestrator to make decisions.

10. Case study

Our implementation of MOSES is multi-project, as it is actively supporting different projects that rely on our cluster, ranging from practical applications (mostly in the field of agriculture) to pure research activities. The project we present as a case study is called *Agro.Big.Data.Science* (ABDS, `http://agrobigdatascience.it/`). It is a regional project whose goal is to rely on a big data solution to support the control and management of product chains in the field of agriculture. By collecting and integrating data from several sources, the technicians are provided with a comprehensive view of agricultural fields that supports the decision-making process, especially in terms of efficient usage of resources (e.g., irrigation, fertilization) and timely actions to contrast alarming situations (e.g., bug infestations) [50]. We choose this case study in order to show how MOSES operates in a real-world scenario and how its metadata and functionalities support the users' daily activities.

10.1. Metadata collection

In the context of ABDS, it is necessary to continuously ingest new data from a variety of sources that differ by type (either project-specific or general-purpose) and communication pattern (streaming or batch). Project-specific sources include IoT devices that stream to our cluster *in situ* details of a certain culture (e.g., the humidity registered by a sensor on the field). General-purpose sources include services that publish open data on the web (mainly satellite images and weather information); in these cases, data collection is simply scheduled at regular intervals. Depending on the source and the type of data, the ingestion process includes a more or less complex pipeline of operations that prepare the data for the analysis. Each operation is a functional process (scheduled by the orchestrator) that puts new data into the DL (either by creating new files or by appending rows into a database) and actively pushes to MOSES the metadata about the whole process run.

One of the most complex pipelines is the one involving satellite images, downloaded from the European Spatial Agency (ESA) webservices and processed to obtain *vegetation indexes* [51] (i.e., quantitative measures that are used to verify the state of the culture in a certain field). The satellite images pipeline is broken down into three steps, namely (i) download from the webservice, (ii) pre-processing to convert images from JPEG to TIF, (iii) actual processing to compute vegetation indexes. An excerpt of the metadata pushed by a whole pipeline run is shown in Figure 5. Each graph node (i.e., each instance of Entity in Figure 2) includes not only standard properties (e.g., the timestamp of an Operation), but also project-specific properties (i.e., the granule of the image Object, with reference to the US National Grid (`https://www.fgdc.gov/usng`) and application-specific properties (e.g., the yarn_id of the Operation, storing the ID of the YARN application that run the process, which is useful for debugging purposes).

As the functional processes create/update data Objects and push metadata into MOSES, the Metadata Extractor is triggered to analyse these objects and pull further information. In the case of satellite images, the Metadata Extractor relies on GDAL (an open-source geospatial library) to extract structural metadata, such as the adopted coordinate system (the EPSG), the extent of the image (i.e., the minimum bounding box, identified by the coordinates of the four corners), and the image bands (red, green, blue, infrared, etc.). All these metadata correspond to Attributes in MOSES and are mapped to the respective Domains.

The domains have been manually defined to describe project-relevant entities in terms of the values they can take; for instance, the EPSG domain is described by the regular expression "^EPSG:[0-9]{4,5}$", which accepts strings beginning with "EPSG:" followed by 4 or 5 numbers. Similar regular expressions are used to recognize the domains of dates and geographic coordinates. Other domains with simpler values are described only by their name and the range of values they admit (e.g., the "Infrared band" domain admits values in $[0, 1]$).

10.2. Metadata fruition

The metadata collected via push and pull methods serve both data scientists and functional process for several purposes. Among those described in the previous sections, we reckon the following main applications.

Object profiling and search (Section 6). The most used functionality is the *Basic search*, which enables a broad set of

Table 5: Description of some low- and high-level APIs to interact with MOSES metadata.

Level	Method	Path API	Description
Low	GET	/node/id/{node_id}	Return node with the given ID
	GET	/node/name/{name}	Return node(s) with the given name
	POST	/node	Create a new node, return ID
	POST	/node/{trans_id}	Create a new node within the given transaction, return ID
	GET	/relationship/id/{rel_id}	Return relationship with the given ID
	POST	/relationship	Create a new relationship, return ID
	POST	/relationship/{trans_id}	Create a new relationship within the given transaction, return ID
	POST	/transaction	Create a new transaction, return ID
	POST	/transaction/commit/{trans_id}	Commit transaction with given ID
	DELETE	/transaction/{trans_id}	Rollback transaction with given ID
High	POST	/search	Return subgraph that matches the given filters
	GET	/prov/forward/{node_id}/{step}	Return the provenance graph originating from node_id
	GET	/prov/backward/{node_id}/{step}	Return the provenance graph leading to node_id
	POST	/metadata/object	Create the portion of graph referred to the object region
	POST	/metadata/schema	Create the portion of graph referred to the schema region
	POST	/metadata/operation	Create the portion of graph referred to the operation region
	GET	/orchestration/operation_type/{node_id}	Return the OperationType more frequently used on objects of the same type of the given ID

queries to be issued on the metadata. Administrators use it to find and verify the results of functional processes, especially in the early stages of workflows' development and deployment. Most importantly, data scientists use it to carry out situational analysis on the data in an exploratory manner. For instance, in the process of assessing the feasibility of computing a statistical model for irrigation advice, search functionalities have been used to locate and study the data collected from IoT devices and weather services. To this end, *Schema-driven search* has also come into play to extend the research and discover data with similar properties (e.g., data Objects with Attributes belonging to weather-related Domains). The latter property is also used by functional processes, which rely on domains rather than specific attribute names or paths. This makes functional processes robust to schema evolution, especially with respect to data ingested from external sources. For instance, if ESA were to change the structure of the published files, the elaboration processes would still be able to complete successfully (provided that the Metadata Extractor is still able to establish the correct association between attributes and domains).

Provenance and versioning (Section 7). Among the most used functionalities there is *Debug and verification*: it is not unusual for functional processes to fail or to compute wrong results — especially during the development and testing phases. The provenance functionalities greatly help data scientists and administrators in tracing the history of process pipelines, which are often centered on the IDs of the implicated processes (e.g., the workflow ID provided by the orchestrator, or YARN's application ID). The main reasons registered for process failures are unexpected cluster downtime and sources unavailability. In both cases, the history regarding objects' Operations is fundamental to understand when and where there has been a problem to investigate. Another important provenance functionality concerns the *Trustworthiness of data*. Besides satellite images published by ESA, we collect open-source weather information from two different services, provided by a public institution and a private non-profit organization respectively. The latter source is more accurate, as the provided weather data are an extension of those provided by the former source with the data collected from additional IoT sensors in certain areas of the region, but it is not always available. Thus, functional processes that work on weather data (e.g., to compute the amount of rain fallen in a certain field) may use either source depending on their availability. In this scenario, provenance functionalities are used to establish the trustworthiness of the obtained results by verifying which source has been used.

Orchestration support (Section 8). Many of the employed functional processes rely on *Triggering* functionalities to activate data transformations; for instance, the workflow in charge of computing vegetation indexes looks for image Objects obtained from Source ESA that are not yet linked to the Operation for computing vegetation indexes. MOSES metadata is also used for *Resource estimation*. The process for vegetation index calculation is among the most expensive ones; thus, the corresponding functional process uses the knowledge on previous executions to tune the request of resources (i.e., the amount of memory and CPU power) for the current execution.

11. Discussion & Conclusions

In this paper we presented MOSES, a framework for metadata handling in BDPs. MOSES has been devised to support BDP users in governing the complexity of data and their transformations. With respect to previous research proposals in this area, which are either described at a high level of abstraction or are limited to a specific technology or functionality, MOSES has been designed to cover a broad set of features and is fully implemented. This allowed us to evaluate its effectiveness in a set of real projects. Although performing a stress test of MOSES on huge BDPs is out of the scope of this paper, our HADOOP-based implementation largely succeeds in supporting all common users' tasks.

The main lessons we learned from the feedback given by users who worked on projects ABDS (the one presented in Section 10) and *Cimice.net* (i.e., a project to monitor and analyze the spreading of stink bugs in the Emilia-Romagna region in Italy) using MOSES can be summarized as follows:

- MOSES largely reduces the user search-time even for projects with limited size (in terms of objects and carried

Figure 5: UML object diagram representing the metadata pushed to MOSES by the functional process in charge to download and elaborate satellite images (for drawing simplicity, entity properties are represented as attributes)

out operations) due to the availability of very detailed information.

- The design of applications and processes must take into account the information collected by MOSES to fully exploit orchestration-related and search functionalities. At the same time, MOSES customizability is very useful to adapt the set of collected information to those used by processes (and human users).

- The effort made to avoid transforming the BDP into a data swamp shifts from the experience and rigor of users to that of identifying the correct trade-off between the amount of information collected and the effort required to collect it.

- Having a graphical interface available greatly reduces the learning curve of MOSES and makes it fully exploitable by users with limited background on BDPs such as data scientists.

- In our projects, 38% of the metadata have been collected in push mode. This confirms that adopting a semi-automatic update strategy is a key factor to make a rich set of metadata available.

We argue that the last point is crucial for the proper setting of MOSES. Our next research efforts will be aimed at defining a methodology to identify the most appropriate set of information to be collected based on the functions to be activated, on the complexity of the applications, and on the resources available. Among the many extensions that could be useful, we believe that the two most important ones are (i) handling metadata access and privacy, which is mandatory in multi-tenant and multi-user scenarios; (ii) improve MOSES user interfaces to further facilitate its use.

Acknowledgment

This research has been partially funded by Regione Emilia Romagna. Funding code PG/2018/630767, funding program POR FESR Programma 2014-2020, Asse 1, Azione 1.2.2.

References

[1] C. Giebler, C. Gröger, E. Hoos, H. Schwarz, B. Mitschang, Leveraging the data lake: Current state and challenges, in: Proc. DaWaK, Linz, Austria, 2019, pp. 179–188.

[2] I. G. Terrizzano, P. M. Schwarz, M. Roth, J. E. Colino, Data wrangling: The challenging journey from the wild to the lake, in: Proc. CIDR, Asilomar, CA, USA, 2015.

[3] Hype cycle for artificial intelligence, `https://www.gartner.com/` (2014).

[4] J. Couto, O. T. Borges, D. D. Ruiz, S. Marczak, R. Prikladnicki, A mapping study about data lakes: An improved definition and possible architectures, in: Proc. SEKE, Lisbon, Portugal, 2019, pp. 453–578.

[5] B. Cheng, S. Longo, F. Cirillo, M. Bauer, E. Kovacs, Building a big data platform for smart cities: Experience and lessons from santander, in: Proc. Big Data, New York City, NY, USA, 2015, pp. 592–599.

[6] A. Behm, V. R. Borkar, M. J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, V. J. Tsotras, ASTERIX: towards

a scalable, semistructured data platform for evolving-world models, Distributed and Parallel Databases 29 (3) (2011) 185–216.
[7] S. Nadal, O. Romero, A. Abelló, P. Vassiliadis, S. Vansummeren, An integration-oriented ontology to govern evolution in big data ecosystems, Inf. Syst. 79 (2019) 3–19.
[8] S. Sharma, Expanded cloud plumes hiding big data ecosystem, Future Generation Computer Systems 59 (2016) 63–92.
[9] A. LaPlante, B. Sharma, Architecting Data Lakes, O'Reilly Media, Sebastopol, 2014.
[10] F. Ravat, Y. Zhao, Data lakes: Trends and perspectives, in: Proc. DEXA, Linz, Austria, 2019, pp. 304–313.
[11] Y. Li, A. Zhang, X. Zhang, Z. Wu, A data lake architecture for monitoring and diagnosis system of power grid, in: Proc. AICCC, Tokyo, Japan, 2018, pp. 192–198.
[12] A. Maccioni, R. Torlone, KAYAK: A framework for just-in-time data preparation in a data lake, in: Proc. CAiSE, Tallinn, Estonia, 2018, pp. 474–489.
[13] P. Sawadogo, É. Scholly, C. Favre, É. Ferey, S. Loudcher, J. Darmont, Metadata systems for data lakes: Models and features, CoRR abs/1909.09377.
[14] C. Quix, R. Hai, I. Vatov, Metadata extraction and management in data lakes with GEMMS, CSIMQ 9 (2016) 67–83.
[15] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, F. Naumann, Data profiling with Metanome, PVLDB 8 (12) (2015) 1860–1863.
[16] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, S. E. Whang, Goods: Organizing Google's datasets, in: Proc. SIGMOD, San Francisco, CA, USA, 2016, pp. 795–806.
[17] J. M. Hellerstein, V. Sreekanti, J. E. Gonzalez, J. Dalton, A. Dey, S. Nag, K. Ramachandran, S. Arora, A. Bhattacharyya, S. Das, M. Donsky, G. Fierro, C. She, C. Steinbach, V. Subramanian, E. Sun, Ground: A data context service, in: Proc. CIDR, Chaminade, CA, USA, 2017.
[18] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, XTRACT: a system for extracting document type descriptors from XML documents, SIGMOD Rec. 29 (2) (2000) 165–176.
[19] J. Hegewald, F. Naumann, M. Weis, XStruct: Efficient schema extraction from multiple and large XML documents, in: Proc. ICDE Workshops, 2006, pp. 81–81.
[20] G. J. Bex, W. Gelade, F. Neven, S. Vansummeren, Learning deterministic regular expressions for the inference of schemas from XML data, ACM TWEB 4 (4) (2010) 14.
[21] M. Klettke, U. Störl, S. Scherzinger, O. Regensburg, Schema extraction and structural outlier detection for JSON-based NoSQL data stores., in: Proc. BTW, 2015, pp. 425–444.
[22] L. Wang, S. Zhang, J. Shi, L. Jiao, O. Hassanzadeh, J. Zou, C. Wangz, Schema management for document stores, Proc. VLDB Endowment 8 (9) (2015) 922–933.
[23] J. Izquierdo, L. Cánovas, J. Cabot, Discovering implicit schemas in JSON data, in: Proc. ICWE, 2013, pp. 68–83.
[24] D. S. Ruiz, S. F. Morales, J. G. Molina, Inferring versioned schemas from NoSQL databases and its applications, in: Proc. ER, 2015, pp. 467–480.
[25] E. Gallinucci, M. Golfarelli, S. Rizzi, Schema profiling of document-oriented databases, Inf. Syst. 75 (2018) 13–25.
[26] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, P. C. Arocena, Data lake management: Challenges and opportunities, PVLDB 12 (12) (2019) 1986–1989.
[27] R. Hai, S. Geisler, C. Quix, Constance: An intelligent data lake system, in: Proc. SIGMOD, San Francisco, CA, USA, 2016, pp. 2097–2100.
[28] H. Mehmood, E. Gilman, M. Cortés, P. Kostakos, A. Byrne, K. Valta, S. Tekes, J. Riekki, Implementing big data lake for heterogeneous data sources, in: Proc. ICDE, Macao, China, 2019, pp. 37–44.
[29] K. Dhamdhere, K. S. McCurley, R. Nahmias, M. Sundararajan, Q. Yan, Analyza: Exploring data with conversation, in: Proc. IUI, Limassol, Cyprus, 2017, pp. 493–504.
[30] J. Wang, D. Crawl, S. Purawat, M. H. Nguyen, I. Altintas, Big data provenance: Challenges, state of the art and opportunities, in: Proc. BigData, Santa Clara, CA, USA, 2015, pp. 2509–2516.
[31] I. Suriarachchi, B. Plale, Crossing analytics systems: A case for integrated provenance in data lakes, in: Proc. e-Science, Baltimore, MD, USA, 2016, pp. 349–354.
[32] M. Interlandi, T. Condie, Supporting data provenance in data-intensive scalable computing systems, IEEE Data Eng. Bull. 41 (1) (2018) 63–73.
[33] C. Mathis, Data lakes, Datenbank-Spektrum 17 (3) (2017) 289–293.
[34] M. Lenzerini, Data integration: A theoretical perspective, in: Proc. SIGACT-SIGMOD-SIGART, Madison, Wisconsin, USA, 2002, pp. 233–246.
[35] M. Zhang, X. Zhang, X. Zhang, S. Prabhakar, Tracing lineage beyond relational operators, in: proc. VLDB, ACM, 2007, pp. 1116–1127.
[36] R. Diestelkämper, M. Herschel, Tracing nested data with structural provenance for big data analytics, in: Proc. EDBT, 2020, pp. 253–264.
[37] A. Maccioni, R. Torlone, Crossing the finish line faster when paddling the data lake with Kayak, Proc. VLDB Endowment 10 (12) (2017) 1853–1856.
[38] P. Agrawal, R. Arya, A. Bindal, S. Bhatia, A. Gagneja, J. Godlewski, Y. Low, T. Muss, M. M. Paliwal, S. Raman, V. Shah, B. Shen, L. Sugden, K. Zhao, M. Wu, Data platform for machine learning, in: Proc. SIGMOD, ACM, 2019, pp. 1803–1816.
[39] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, S. E. Whang, Managing Google's data lake: an overview of the Goods system, IEEE Data Eng. Bull. 39 (3) (2016) 5–14.
[40] B. Glavic, Big data provenance: Challenges and implications for benchmarking, in: Proc. WBDB, San Jose, CA, USA, 2012, pp. 72–80.
[41] Y. Simmhan, B. Plale, D. Gannon, A survey of data provenance in e-science, SIGMOD Rec. 34 (3) (2005) 31–36.
[42] M. Herschel, R. Diestelkämper, H. Ben Lahmar, A survey on provenance: What for? What form? What from?, VLDB J. 26 (6) (2017) 881–906.
[43] J. Stefanowski, K. Krawiec, R. Wrembel, Exploring complex and big data, Int. J. Appl. Math. Comput. Sci. 27 (4) (2017) 669–679.
[44] I. Altintas, O. Barney, E. Jaeger-Frank, Provenance collection support in the kepler scientific workflow system, in: Proc. IPAW, 2006, pp. 118–132.
[45] R. Lourenço, J. Freire, D. E. Shasha, Debugging machine learning pipelines, in: Proc. DEEM@SIGMOD, 2019, pp. 3:1–3:10.
[46] L. Moreau, P. T. Groth, Provenance: An Introduction to PROV, Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, 2013.
[47] I. Megdiche, F. Ravat, Y. Zhao, A use case of data lake metadata management, Data Lakes 2 (2020) 97–122.
[48] T. Heath, C. Bizer, Linked Data: Evolving the Web into a Global Data Space, Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers, 2011.
[49] G. Klyne, Resource description framework (RDF): Concepts and abstract syntax, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (2004).
[50] E. Gallinucci, M. Golfarelli, S. Rizzi, Mo.re.farming: A hybrid architecture for tactical and strategic precision agriculture, Data Knowl. Eng. 129 (2020) 101836.
[51] D. W. Deering, Rangeland reflectance characteristics measured by aircraft and spacecraft sensors, Ph.D. thesis, Texas A&M Univ., College Station (1978).