University of Stuttgart - Institute for Parallel and Distributed Systems / AS

Modeling Metadata in Data Lakes - A Generic Model

Rebecca Eichler, Corinna Giebler, Christoph Gröger, Holger Schwarz and Bernhard Mitschang

In: *Data & Knowledge Engineering (2021)*, 101931.

```
BIBTEX:
@article{eichler2021,
  title={Modeling metadata in data lakes—A generic model},
  author={Eichler, Rebecca and Giebler, Corinna and Gr{\"o}ger,
Christoph and Schwarz, Holger and Mitschang, Bernhard},
  journal={Data \& Knowledge Engineering},
  year={2021},
  publisher={Elsevier}
}
```

Modeling Metadata in Data Lakes - A Generic Model

Rebecca Eichler[1], Corinna Giebler[1], Christoph Gröger[2], Holger Schwarz[1], and Bernhard Mitschang[1]

[1] University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{Firstname.Lastname}@ipvs.uni-stuttgart.de
[2] Robert Bosch GmbH, Borsigstraße 4, 70469 Stuttgart, Germany
{Firstname.Lastname}@de.bosch.com

Abstract Data contains important knowledge and has the potential to provide new insights. Due to new technological developments such as the Internet of Things, data is generated in increasing volumes. In order to deal with these data volumes and extract the data's value new concepts such as the data lake were created. The data lake is a data management platform designed to handle data at scale for analytical purposes. To prevent a data lake from becoming inoperable and turning into a data swamp, metadata management is needed. To store and handle metadata, a generic metadata model is required that can reflect metadata of any potential metadata management use case, e.g., data versioning or data lineage. However, an evaluation of existent metadata models yields that none so far are sufficiently generic as their design basis is not suited. In this work, we use a different design approach to build HANDLE, a generic metadata model for data lakes. The new metadata model supports the acquisition of metadata on varying granular levels, any metadata categorization, including the acquisition of both metadata that belongs to a specific data element as well as metadata that applies to a broader range of data. HANDLE supports the flexible integration of metadata and can reflect the same metadata in various ways according to the intended utilization. Furthermore, it is created for data lakes and therefore also supports data lake characteristics like data lake zones. With these capabilities HANDLE enables comprehensive metadata management in data lakes. HANDLE's feasibility is shown through the application to an exemplary access-use-case and a prototypical implementation. By comparing HANDLE with existing models we demonstrate that it can provide the same information as the other models as well as adding further capabilities needed for metadata management in data lakes.

Keywords: Metadata Management · Metadata Model · Data Lake · Data Management · Data Lake Zones · Metadata Classification.

1 Introduction

In the course of the digitalization, new technological developments such as the Internet of Things led to a considerable increase in generated data [28]. This data contains important knowledge and has the potential to provide new insights that can be used, for example, to optimize processes or to achieve a competitive advantage [2]. However, handling the generated data volumes and extracting the data's value poses a challenge for organizations [28] and requires new strategies and concepts beyond, e.g., traditional data warehousing and business intelligence solutions. In this context, the data lake concept was born. It is a data management platform designed to incorporate data at scale, from heterogeneous sources, of varying structure, and in its raw format [12,9]. With these properties, the data lake enables varying analysis such as exploration or machine learning, to exploit the data's value [9].

In order to preserve the data's value and prevent the data lake from turning into a data swamp, which is an inoperable data lake, metadata management is required [24]. Metadata is a type of data, which provides information about other entities, such as other data, processes or systems [1]. Metadata management constitutes all activities which involve managing an organizations' knowledge on its data [3]. Without this knowledge, data may not be applicable for the intended purpose, e.g., due to a lack of quality or trust.

One important aspect of metadata management is the definition of a metadata model (e.g., [15,22,25]). By our definition a metadata model describes the relations between data and metadata elements and what metadata is collected, e.g., in the form of an explicit schema, a formal definition, or a textual description. For the purpose of creating such a metadata model, it is necessary to know roughly what metadata will be acquired so that the model can reflect these. However, both general and data-lake-specific definitions of metadata management focus on different purposes for conducting it, such as the implementation of data governance specifications [11], the support of query processing and data quality management [12] or topics like tracing the data's lineage [6]. Since these topics are very diverse and cannot be easily combined into an overall picture, yet all require different metadata, there is no comprehensive overview of what metadata needs to be collected within a data lake. Nonetheless, to increase or retain the data's value the metadata model must be able to record a broad range of metadata and must therefore be very generic. To be sufficiently generic, such a metadata model must reflect any potential metadata management use case of a data lake. This includes standard use cases like the collection of lineage information, as well as organization-specific use cases such as use cases for the manufacturing domain. The generic nature also includes the ability to map metadata of any kind, i.e. metadata on different topics, metadata which describes data properties as well as its relations, and also metadata that refers to specific data as well as metadata that is relevant to the entire data lake.

However, existent metadata models applicable to data lakes, e.g., [4,23,26], are not sufficiently generic as they cannot support every potential metadata management use case. For instance, some of them were developed for only one specific metadata management use case [13,16,30]. The existent metadata models are based on metadata categorizations and/or lists of metadata management features. As our discussion reveals, both do not produce truly generic models. In this paper we address this gap by making the following contributions:

1. We provide a tangible definition of metadata management within organizations, together with a list of associated tasks. It is shown in which way metadata modeling fits into this context and which role it plays.
2. We introduce a new approach for constructing a generic metadata model for data lakes by investigating existent models and their shortcomings.
3. Based on this approach, we developed a generic metadata model called HANDLE, short for Handling metAdata maNagement in Data LakEs.
4. We highlight HANDLE's ability to model the same metadata in various ways according to the intended usage, together with HANDLE's ability to reflect both metadata specific to a data element or the entire data lake.
5. We demonstrate how HANDLE supports the inheritance of metadata, meaning the ability to transfer metadata across data elements.
6. Finally, we assess HANDLE by firstly, testing its applicability on a standard use case in the Industry 4.0 context, secondly, we prototypically implemented HANDLE based on this use case, and lastly, compare it to existing metadata models. The comparison yields that HANDLE can reflect the content of the existent metadata models as it is defined on a higher abstraction level which also makes it more generic and that it provides additional metadata management capabilities.

HANDLE has been presented previously in [5]. The paper at hand extends the published version by examining additional aspects, reflected in the contributions 1, 4 and 5, and by explaining individual aspects in more detail. In the following, Section 2 provides a definition on metadata management and the involved tasks. Metadata models presented in scientific literature are introduced

and assessed in Section 3. Section 4 specifies the requirements for the new metadata model, which is presented in Section 5, followed by an assessment in Section 6. Lastly, the paper is concluded by Section 7.

2 Metadata Management

This paper presents a metadata model and consequently also deals with the modeling of metadata. In order to understand the role of metadata modeling and its relevance, we take a look at metadata management in a broader context. At the heart of metadata management is metadata. Metadata is data that provides information on other entities, such as other data, processes or systems [1]. For example, the information that "John Doe" created dataset X provides a description on the author of dataset X and is therefore metadata. As metadata is data which describes a specific type of content, it can be seen as a type of data, like master data. Like other data, metadata must also be managed, which is referred to as metadata management [3]. To gain an overview of metadata management tasks and which metadata must be collected we examined existent definitions of metadata management.

Present definitions often refer to the management of metadata [11,6] or the term is circumscribed through a list of the negative outcomes when metadata management is omitted, as in [3,12]. The definitions are frequently accompanied by a list of benefits and example topics which it involves or enables. However, we have found that these topics are very different, ranging from the implementation of data governance specifications [11], over productionizing data science [34] and query processing to data quality management [12]. These cannot be integrated easily to form an overall picture of metadata management and derive a set of metadata management tasks. Therefore, we provide a more tangible definition: *Metadata management is data management for metadata.*

Based on this definition there are different variants of data management depending on the type of data, as illustrated in Figure 1. The basic set of data management tasks, however, remains the same for all data types, including metadata. These are depicted in Figure 2, and involve the three main blocks: data governance, lifecycle management and foundational activities [3]. The blocks and contained tasks are mainly based of DAMA's data management function framework [3]. Data governance presents the basis for data management as it involves the planning, monitoring and enforcement of both the data lifecycle management and foundational activities. This includes creating policies, which specify what has to be done and standards, which specify how to do things [3]. Lifecycle management involves all processes related to the design, creation, obtaining, storing, using, maintaining, enhancing as well as archiving and deleting of data [3,32]. According to DAMA, data modeling and hence metadata modeling is part of the lifecycle management "Design"

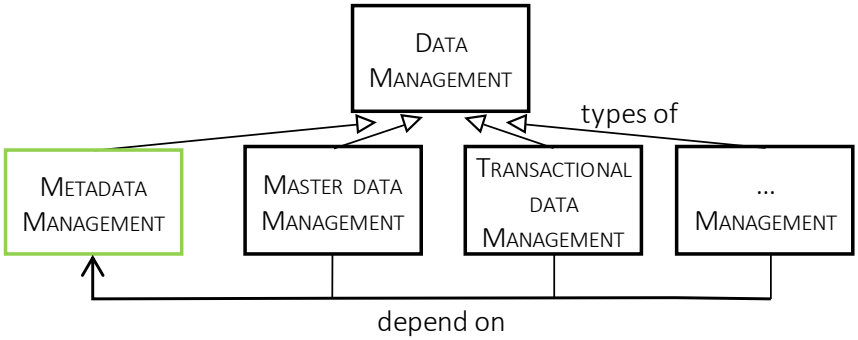

Figure 1. The relation and interdependencies between data management and metadata management.

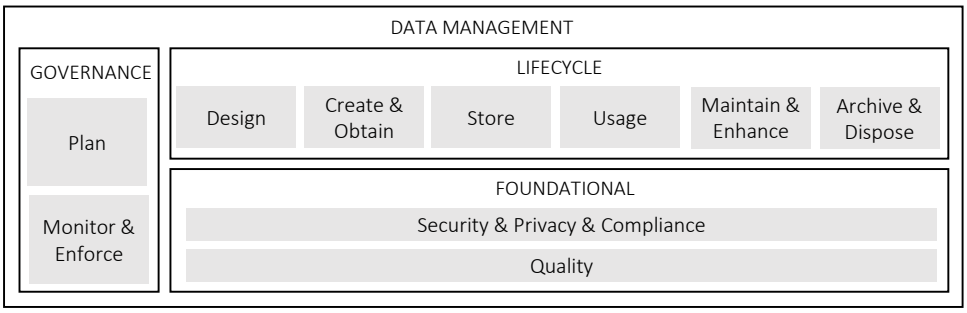

Figure 2. Data Management Activities (Based on [3]).

step [3]. The metadata model developed within this step determines what metadata will be stored and how it will be structured. The following lifecycle steps build on this metadata model. The foundational tasks are performed throughout all of the above described lifecycle steps and include the realization of security, privacy, compliance and data quality management. With this definition of metadata management there is an overview of the tasks involved in metadata management and in which of these metadata modeling is conducted. These tasks can also be understood as requirements for metadata management.

Besides the tasks metadata management involves, the explanations above also provide a rough overview of the metadata which need to be collected and reflected by the metadata model. Metadata management and data management are generally viewed as two separate procedures as there is a dependency between metadata management and the other data management types, as shown in Figure 1. In order to conduct data management, metadata is required throughout the depicted data management tasks in Figure 2. For example, retaining data privacy and security involves introducing and enforcing access rights, which are essentially metadata as these describe who is allowed to access the data. Data quality is also documented through metadata as well as data models and the process of creating and obtaining data. Each task can contain several management use cases which may require the collection of metadata, like the "Usage" task may contain the tracking of data lineage, data access and more. Furthermore, metadata specific to various domains may be collected throughout these tasks such as ontologies on manufacturing or biological aspects. In terms of the metadata modeling, this means there is a large amount of use cases, including various domain-specific use cases which require documentation through metadata. The metadata model must be very generic in order to be able to reflect all of these use cases.

Today, metadata management is considered to be a challenge in the context of data lakes [24]. Therefore, we focus on metadata models for data lakes in the remainder of this paper and investigate which characteristics these must have.

3 Related Work: Discussion of Existent Metadata Models

A literature research was conducted to get an overview of the existent models. Metadata models presented in the scope of metadata management systems applicable to data lakes include the model for the Generic and Extensible Metadata Management System (GEMMS) [22], for Walker and Alrehamy's personal data lake solution [31], and lastly, the metadata model for the data context service Ground [15]. Many systems both in the research and commercial context do not disclose their metadata model and thus we cannot examine their generic extent, e.g., [12,14,17]. Other models exist which are not part of a specific system. However, many of these, also including that by Walker and Alrehamy, focus on a specific topic and thus, only support a limited set of use cases which

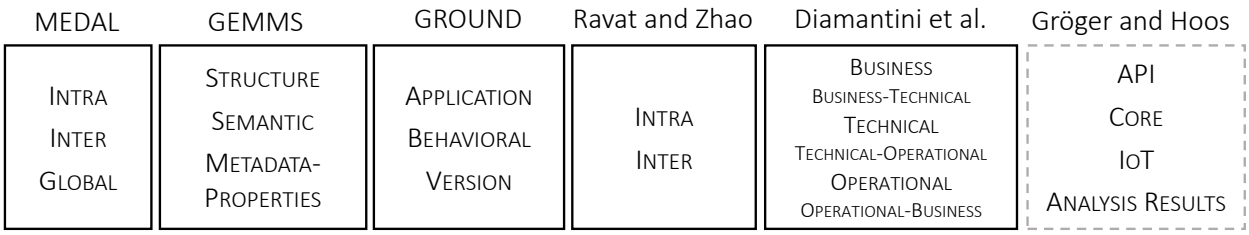

Figure 3. Set of metadata categorizations, the first five belong to the selected metadata models [26,22,15,23,4]. The sixth, in dashes, does not belong to a metadata model [11].

makes them non-generic, e.g., [13,16,25,29,30]. Thenceforth these models are not considered here. More general models also created for data lakes include those by Ravat and Zhao [23], Diamantini et al. [4], and lastly, Sawadogo et al.'s model MEtadata for DAta Lakes (MEDAL) [26].

The generic degree of the five models GEMMS, Ground, MEDAL, and those by Ravat and Zhao, and Diamantini et al. is examined and discussed in the following. Section 3.1 shows that the basis on which the existent models were constructed is insufficient for building a generic metadata model. A representative use case is presented in Section 3.2 and Section 3.3 shows that it cannot be realized by the existing models, thereby demonstrating that these are not sufficiently generic.

3.1 Assessing the Basis of Existent Models

An examination of the five selected metadata models yields that these were built with two general approaches. The first approach uses a *categorization of metadata*, the second, employs a *list of metadata management features* that must be supported.

The categorization-based approach differentiates types of metadata. As can be seen in Figure 3, each categorization differentiates metadata through other qualities, thereby providing different perspectives on metadata. For example, the categories in MEDAL refer to how the metadata is modeled whereas Diamantini et al. categorize the metadata by its content. Building a metadata model based on only one of these perspectives makes it less generic. Furthermore, a categorization does not provide any guidance on modeling use cases and therefore does not contribute to building a generic metadata model.

The feature-based approach involves building the model to support a predefined list of features. Features are derived from metadata management use cases. If the list covers all relevant metadata management features, and if the metadata model supports all of these features, then the model would be complete. As can be seen in Figure 4, some of the lists contain high-level and some

Figure 4. This is a display of three lists of metadata management features. The first belongs to the model MEDAL [26], whereas the other two, in dashes, are from Gartner [27], and Gröger and Hoos [11] and are created independent of a metadata model.

detailed feature descriptions, making it impossible to combine them. Defining high-level features might not suffice to derive all necessary requirements for a metadata model. However, defining one comprehensive list of detailed features is not realistic as each organization will have its own set of relevant metadata management use cases and a different thematic focus, also visible in Figure 4.

In conclusion, neither the categorization-based nor the feature-based approach are an adequate foundation for building a generic metadata model.

3.2 Metadata Management Use Case for Model Evaluation

To evaluate the existent models by testing their limits and generic extent we use a representative metadata management use case, which is based on an Industry 4.0 scenario with an enterprise data lake. The data lake contains data on products, processes and customers including personal data (see [18] for data management in industrial enterprises and Industry 4.0).

Data lake projects which involve personal data on EU citizens, e.g., data on customers, are subject to legal requirements such as the General Data Protection Regulation (GDPR) [8]. Conformity to the GDPR requires the collection of information on the personal data's use and the users accessing it [7]. Therefore, we introduce the data-access-use-case, which involves collecting access information. It is a representative metadata management use case frequently implemented in the data lake context.

Within our Industry 4.0 scenario, access information is collected amongst other things on a table containing customer data, as depicted in Figure 5. Data access information may include details on the user or application accessing it, and the type and time of the action. Hence, the model must support some form of metadata object with properties to reflect most varied information. For example, an object could be created for each user and action with properties such as name, id, or action time and type. Figure 5 illustrates metadata objects, in dashed/green, grouped by a metadata object called "Access Info". In order to allocate the metadata to a specific dataset some form of pointer or reference is required, depicted as the solid/blue "Data Pointer" object.

Data within data lakes is often organized in so-called zones according to its processing degree [10], e.g., as standardized data in the trusted zone [33]. Therefore, the same dataset may be stored multiple times in different zones in varying processing degrees. The metadata collected on datasets throughout zones should be distinguishable according to the particular zone. Consequently, the access information must be collected per zone. Assuming the example data lake has two or more zones, such as a raw zone containing the data in its native format, and a trusted zone holding pseudonymised data. It should be recognizable who accessed a customer's personal

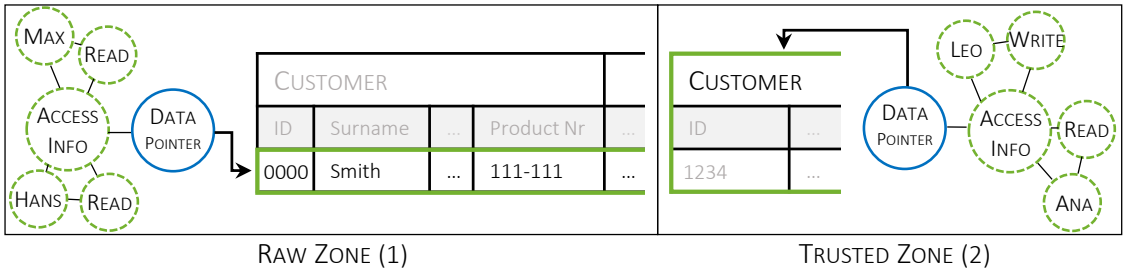

Figure 5. The image displays access metadata collected on varying granular levels of the customer table. The customer table is stored twice, once in each data lake zone. The dashed/green circles are metadata objects with properties, e.g., a user object with the name "Max". They belong to the accordingly highlighted part of the table. The solid/blue circles denote a pointer containing the data's storage location.

information and who only saw the customer's pseudonym. For example, in Figure 5 "Max" read personal data, but "Ana" only saw pseudonymised data.

Assuming it is desired to track the access to each customer's personal data, then the access information must be collected per customer. The pseudonymised version of the customer table does not yield any personal information and consequently does not require collecting the access info per customer. In this case, it is sufficient to collect the access information on the entire table as opposed to a single row. Therefore, our use case requires collecting metadata on varying granular levels.

The presented scenario imposes three requirements which we use to test the metadata models' limits. For this use case the metadata models must be flexible in creating *Metadata properties* for Metadata objects to reflect most varied information, the model must *support data lake zones* and it must support the collection of metadata on *various granular levels*.

3.3 Assessing the Generic Extent of the Existent Models

Within this section, the five models selected in the beginning of Section 3 are examined in respect to the three use case requirements: *metadata properties, data lake zones* and *granularity*.

As signified in Table 1, all models except that by Diamantini et al. support adding metadata properties in some way or another, and therefore fulfill the first requirement. Ravat and Zhao's model is partially checked as they support adding keywords describing their data elements, which does not however, suffice for modeling, e.g., an actor accessing the data. For this purpose, they have explicitly defined access properties, but they are missing the type of action performed.

Of the five models, only that by Ravat and Zhao addresses the zone concept of data lakes. They use a specified zone architecture. However, their model description does not reveal how they allot their data and metadata to specific zones. Therefore, this quality is partially checked for their model. Diamantini et al.'s model and MEDAL both support data polymorphism, which denotes the ability to store multiple representations of the same data [4,26]. This is required for building zones. It does not however, enfold all the zone concept's characteristics, such as a clear specification of the data's processing degree within each zone. Therefore, they are partially checked in Table 1.

GEMMS and Diamantini et al.'s model define two levels of granularity, partially fulfilling requirement three. Ravat and Zhao mention dataset containment, but it is not clear whether this can be used to implement the granularity topic. Therefore, none of the models support adding multiple levels of granularity.

In conclusion, none of the five metadata models are flexible enough to support the presented access-use-case and thus, are not sufficiently generic.

Table 1. Coverage of Access-Use-Case Requirements by the Metadata Models. The $\surd$ represents a fulfilled requirement and the $(\surd)$ a partially fulfilled requirement.

Requirements	GEMMS	Ravat and Zhao	Ground	Diamantini et al.	MEDAL
Metadata Properties	$\surd$	$(\surd)$	$\surd$		$\surd$
Data Lake Zones		$(\surd)$		$(\surd)$	$(\surd)$
Granularity	$(\surd)$			$(\surd)$	

4 Requirements for a Generic Metadata Model

Section 3 demonstrated the necessity for a new generic metadata model for data lakes. We acquired the knowledge that both a categorization- and feature-based approach do not yield a truly generic

model. This was demonstrated with a set of use case specific requirements. Therefore, a different approach is proposed to define *a new set of more general requirements* for building a generic model which can reflect a broader scope of use cases across various domains. This approach is flexibility-oriented, whereby the requirements are based on the existent models' strengths and limits, but mainly aim at providing a basis for a highly flexible model.

In order to support any metadata management use case, the model must be very flexible in its ability to assimilate metadata. Therefore, the first requirement is *(1) modeling the metadata as flexible as possible*. According to our analysis of the existent models, a high level of flexibility is achieved through the following six conditions: *(1.1)* Metadata can be stored in the form of metadata objects, properties and relationships; *(1.2)* The amount of metadata objects per use case is unlimited; *(1.3)* Each metadata object can have an arbitrary number of properties; *(1.4)* Metadata objects can exist with or without a corresponding data element; *(1.5)* Metadata objects can be interconnected and *(1.6)* Data elements can be interconnected.

The second requirement denotes the ability to collect metadata on *(2) multiple granular levels*, thus maintain flexibility with regard to the level of detail and allocation of metadata. Through granular levels the model supports heredity of metadata. For example, technical metadata added on a schema level also applies to more granular data elements such as tables, columns, rows and fields.

The metadata model is developed for metadata management in data lakes and should therefore support data lake characteristics. Most metadata is collected on specific data elements, which are organized in zones, thus the model must support *(3) the concept of data lake zones* [10]. This means, metadata should be distinguishable across zones, hereby gaining flexibility in allocating metadata.

Lastly, it should be flexible in the sense that it can *(4) integrate any categorization in the form of labels*, e.g., MEDAL's intra, inter and global labels or Gröger and Hoos' labels API, Core and so on (see Figure 3). This helps to speedily identify the context of the data. It can also be used to check whether all types of metadata are being collected.

These four requirements constitute the new set of general requirements for a generic metadata model in the data lake context.

5 HANDLE - A Generic Metadata Model

We used the requirements given in Section 4 to develop a new model which we present in this section. The new model is called HANDLE, short for "Handling metAdata maNagement in Data LakEs". The new model's intent is to handle all metadata management use cases, general and domain-specific, thereby getting a handle on all metadata. To achieve this HANDLE is defined on a high abstraction level, which gives it the necessary flexibility in assimilating any metadata. Consequently, it does not explicitly specify which metadata needs to be collected or provide decision-support capabilities in this context. The intention is rather to provide a foundation so domain experts such as data scientists can model, collect and also retroactively extend any metadata they require for their specific use cases.

The conceptual metadata model consists of two parts, the core model, illustrated in Figure 6, and three core model extensions, which need to be adapted for each data lake implementation. The core model is a metamodel defining all the elements and the relations required for modeling a metadata management use case and is introduced in Section 5.1. The core model extensions each address the granularity, zone and categorization topics in more detail, according to the requirements 2-4 and are explained in the Sections 5.2-5.4 respectively. All of the models are modeled according to the Crow's Foot Notation.

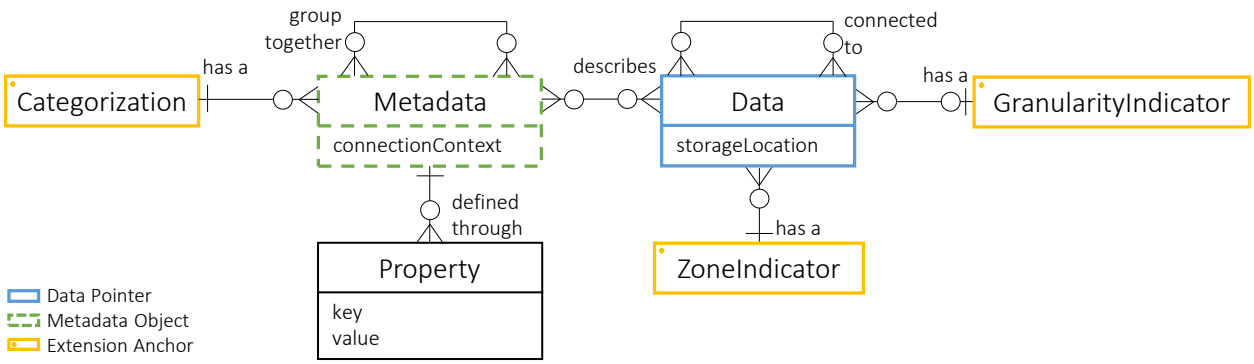

Figure 6. HANDLE's Core Model

5.1 The Core Model

As depicted in Figure 6, one of the core models main entities is the *data* entity, illustrated in solid/blue. In order to avoid storing data redundantly, the data entity represents a pointer to the data in the data lake. The path to the data element is stored in the *storageLocation* attribute. According to requirement 1.6, data elements can be interconnected. For instance, a data element representing a table's row can be connected to the superior data element representing the overall table. The data element has two entities attached to it, the zoneIndicator and the granularityIndicator. They indicate the zone the data is stored in and the level of granularity on which the metadata is collected, as dictated by the requirements 2 and 3. The intended usage of both indicators is explained on the basis of model extensions in the subsequent paragraphs.

The second central entity of the core model is the *metadata* entity, depicted in dashed/green. It is the metadata object specified in requirement 1.1, by way of example it could represent a user who accessed data. The metadata entity is connected to none, one, or many data elements and each data entity may have zero or many metadata entities connected to it, hereby fulfilling requirement 1.4. For instance, the user can access many data elements and data elements can be accessed by many users. An attribute called *connectionContext* describes what information the metadata element contains. For example, the user metadata element may have a connection context called "accessing user". In line with requirement 1.3, the metadata entity can have an arbitrary number of *properties* in the form of key-value pairs, e.g., "name: Hans Müller". According to requirement 1.5, the metadata entity's self-link enables to group zero or more metadata elements together, like the "Access Info" group, as illustrated in Figure 5. Grouping the elements according to some context is helpful when a lot of metadata on the same topic is collected for a data element. As specified through requirement 4, the metadata entity is labeled according to any content-based categorization, represented by the *categorization* entity.

Figure 7 depicts an example instantiation of the core model. In the following, instances of the data and metadata entities are illustrated as circles with the according attributes attached underneath and the categorization, zone and granularityIndicator attached on top like a label. In this example the data element has a storageLocation attribute containing the path to customer data in the lake. The granularityIndicator "Table" implies that the path points to a table. Furthermore, the referenced content is stored in the raw zone, implied through the zoneIndicator "Raw". The data element has one metadata element, which describes the data's content, as declared through the connectionContext attribute. Additionally, the metadata element has the properties "name" and "description", which are instances of the core model's property entity. Lastly, the metadata element is of the type "Business" as specified through the categorization entity.

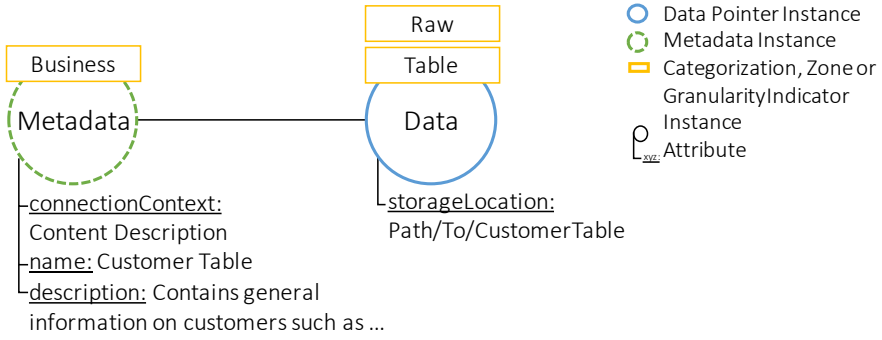

Figure 7. Example Instantiation of the Core Model Entities.

Modeling Metadata According to Utilization: The core model provides the ability to model the same content in various ways. Figure 8, for instance, depicts three options, a), b) and c), for storing metadata on how a data element was accessed and by whom. In option a) all of the information, i.e., the actor and action, are inserted into a single metadata object, pictured in dashed/green, and attached to the according data entity, in solid/blue, for each executed action. This simple version does not require human modeling and facilitates adding new metadata with little effort and in an automated fashion. However, if this metadata is collected to see which users have accessed a data element, then each metadata element would have to be checked individually. This is inefficient and could be sped up by creating a separate actor and action metadata object, as shown in option b). In this model the information is structured for each data entity, but although it is related information, it is still kept separate as in a silo. Hence, the question "what data elements does the user Hans work with" also elicits a high effort. For this the metadata on every possible data entity would have to be loaded to see whether Hans performed an action on it. Therefore, the metadata can be interconnected across data entities as pictured in option c). Interconnecting metadata objects across various data entities is also less redundant as the same information is not stored several times for each data element. In this model, the access metadata is also attached to

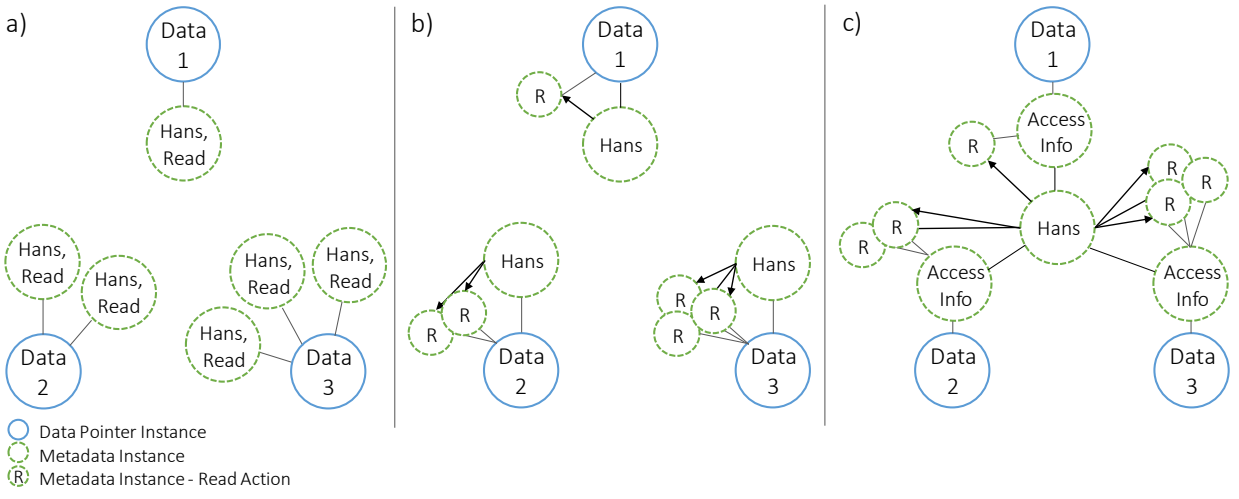

Figure 8. This image shows three modeling variants of the same metadata management use case in which the access to data is recorded. The metadata objects are depicted in dashed/green and the data objects in solid/blue. From left to right the same content is stored according to increasing specifications, based on the metadata's intended usage. Properties, indicators and categorizations are not displayed due to space reasons.

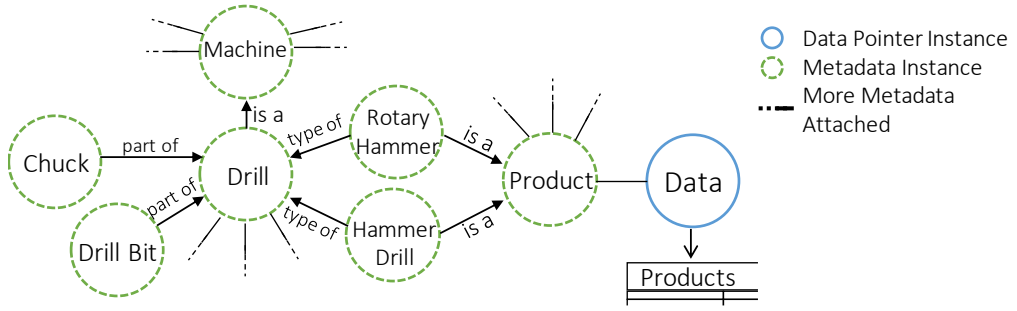

Figure 9. An extract of a knowledge base demonstrating global metadata. When implemented, the "part of", "type of" and "is a" relations are specified through the metadata elements' connectionContext attribute.

an intermediary "Access Info" node in order to group this information and prevent the data entity being swamped by indefinitely increasing access metadata. Through these examples we demonstrate that the core model allows for the metadata to be modeled according to the intended employment, be it the automated acquisition of new metadata or specific queries.

Data-Specific and Global Metadata: The core model supports collecting both metadata which belongs to specific data entities and also metadata which is more universal and can exist without a corresponding data element. In Figure 8 the metadata exemplifies the former, i.e. metadata which only makes sense with an according data element. Other metadata, such as knowledge bases are applicable to a broader scope of data, potentially across the entire data lake and are examples of what Sawadogo et al. call global metadata [26]. These can be represented through the core model by creating and interconnecting several metadata objects which can have, but do not necessarily need a connection to a data element. By way of example, Figure 9 illustrates an extract of a simplified knowledge base. The knowledge base is composed of metadata objects which do not describe a specific dataset in the lake but rather provide a contextual overview of domain knowledge on machines. This domain knowledge applies to data across the whole or a substantial part of the data lake and is therefore an example of global metadata. Nonetheless, it may contain links to matching data in the lake, such as the "Product" table.

5.2 The Granularity Extension

The granularityIndicators have to be adapted according to the intended usage, e.g., for relational structures, as depicted in Figure 10. Thus, the are modeled as an extension to the core model. The granularityIndicator entity enables collecting metadata on different granular levels. These levels are closely tied to some kind of structure in the data. For example, the object, key, value, or key-value

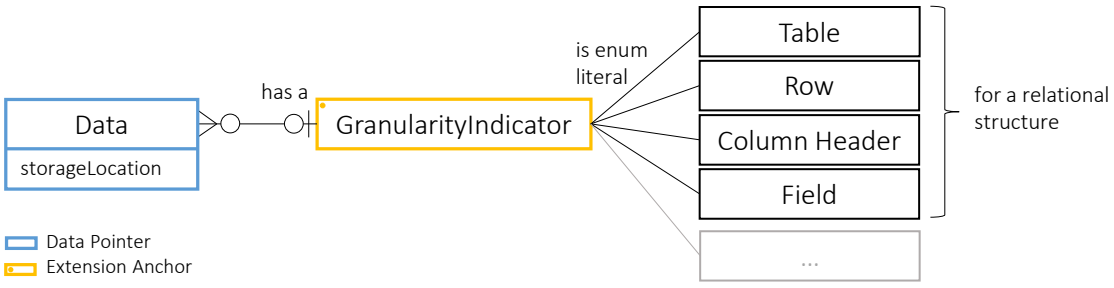

Figure 10. The Granularity Extension to the Core Model

pair instances in a JSON Document may be used as granularity levels. The granularityIndicator is not, however, limited to "structured data." For instance, videos are categorized as "unstructured data" and yet, one may want to collect metadata on single frames of the video. In this case, there would be a video level and a frame level. Domain knowledge provided by domain experts can be helpful for selecting granularity levels, as often it is necessary to understand, e.g., if the metadata refers to the content of a single frame or an entire video.

Figure 10 lists a few enumerations, which can be used to indicate the granular levels of relational data. The "..." indicates that other enumerations may be added as needed. In order to collect metadata on different levels, a corresponding data element must be created that points to that granular instance. So, there may be a set of data elements all referring to the same dataset, simply pointing to more or less specific granular levels. Demonstrating the granularityIndicators defined here, the "Data Pointer" in Figure 5's raw zone would have a label called *Row* and the "Data Pointer" in the trusted zone would have a label called *Table*. There could be other "Data Pointers" in these zones, for instance another pointer to the overall table in the raw zone with the label *Table*.

Metadata Inheritance Through GranularityIndicators: The granularityIndicators enable collecting metadata on various granular levels but it has not yet been considered to what extent metadata can be transferred to other elements. As mentioned in Section 4, the granularityIndicators can facilitate the inheritance of metadata. By our definition, the inheritance of metadata signifies the transferability of metadata attached to data of lower granularity to that of higher granularity. Within relational data, for instance, a table is of a lower granularity and a row of higher granularity. When a user wishes to access a few rows within this table, metadata on access rights are required to ascertain whether this user has the clearance to view this data. In this case it would not make sense to collect the access rights per row, therefore, they are stored on a lower granular level, like the table level. Nonetheless, even though the metadata is attached to the table level, it is also applicable to the according rows and is thereby, heritable. This has the advantage that the metadata can be stored once on a lower granular level and be transfered to higher levels, as opposed to storing it several times for each element in the higher level.

In order to access the inheritable metadata from data of a lower granular level, the hierarchy between the granularityIndicators must be defined. There may be several data entities with various granularityIndicators all pointing to various granular levels of the same dataset without their connection being known. By using the granularityIndicators as depicted in Figure 10, Figure 11

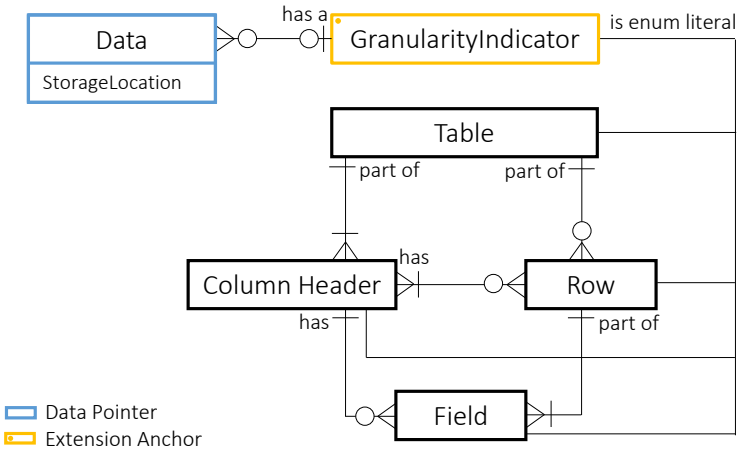

Figure 11. An exemplary definition of the hierarchy among granularityIndicators to facilitate metadata inheritance.

illustrates an exemplary hierarchy between these granularityIndicators. It defines that a *Table* consists of *Rows* and a *Row* consists of one or more *Fields* as these are connected through a *part of* relation. Furthermore, a *Field* and *Row* have either one or one or many *Column Headers* respectively. Having accessed a data entity pointing to a row, the according parent element pointing to the table can be queried through the part of relation and hence, heritable metadata can be found.

5.3 The Zone Extension

Figure 12 illustrates the intended usage of the *zoneIndicator* entity, using the zone model by Zaloni [33]. The zoneIndicator entity is a label on the data entity supplying information on the location of the data element in the data lake's zone architecture. Depending on the zone definition, the data's transformation degree is immediately apparent through it. The different zones are modeled as enumerations for the zoneIndicator. In order to use another kind of architecture, the zone enumerations and their relationships need to be adjusted.

The model illustrates that every data element must have exactly one zoneIndicator, but the indicators may be applied to zero or many data elements. In this model, the *rawZone* entity is designed to be the central zoneIndicator. The reason behind this design decision is that the data is sometimes directly loaded into the raw zone, even though it is the second zone, because the first zone, the transient loading zone, is optional and can be omitted. Furthermore, data loaded into the transient loading zone is only stored temporarily and may not move on into other zones if it does not pass the quality checks. Consequently, if data is stored in any of the other zones, it will have a corresponding data element in the raw zone, however, not necessarily in the transient loading zone. Thus, the raw zone is the most stable reference. The other zoneIndicators, the *transientLoadingZone, trustedZone, refinedZone* and *sandbox*, have a *link* entity, connecting them to the corresponding data element in the raw zone. The information from where the data was imported into the zone is stored within the *importedFrom* property and the corresponding *timestamp* are stored with the link. The importedFrom attribute may contain the name of a zone or the original source. Within Zaloni's zones, the data should first move through the transient loading zone, the raw zone, the trusted zone and lastly, through the refined zone. In the case of the raw zone and sandbox, the data may be loaded directly from the source. The importedFrom attribute enables tracing the data's progress through the zones. As the data may not be moved into the raw zone from the transient loading zone this enumeration can exist without a link to the rawZone element. If it was moved into the raw zone, then it must have a link connecting them.

Figure 13 shows an exemplary instantiation of the the zone extension presented in Figure 12. The four illustrated data elements are all pointers to versions of the same table stored in different zones. The left element is stored in the rawZone and, therefore, points to the original unaltered version of the data. The other data elements are connected to it through a link, illustrated by an

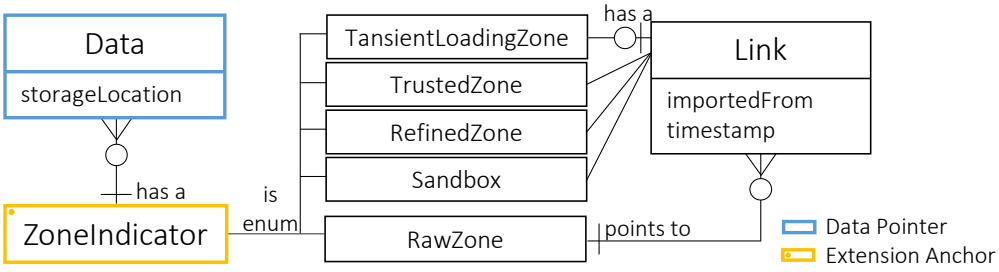

Figure 12. The Zone Extension to the Core Model, Using Zaloni's Zones [33,20].

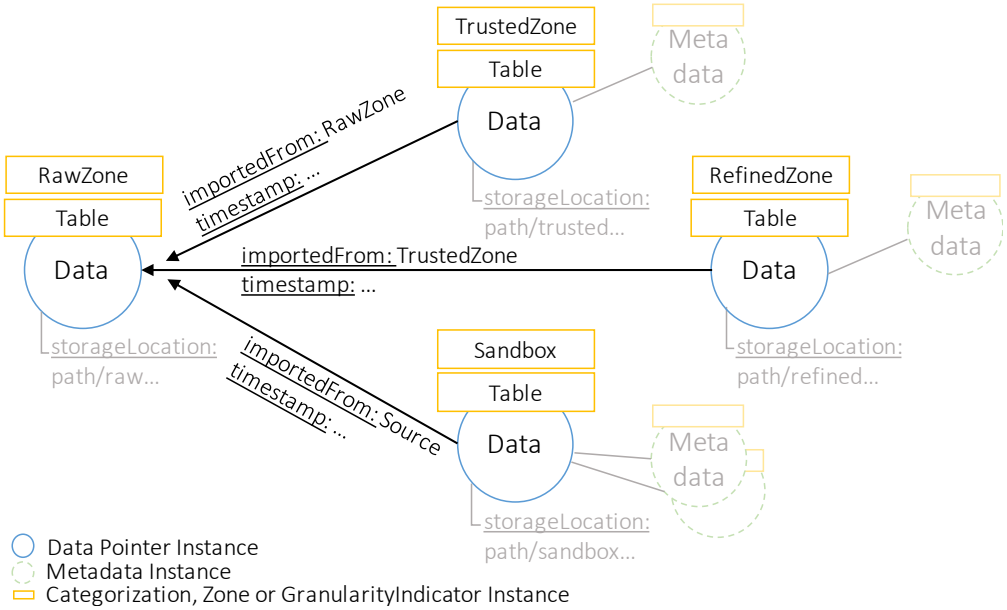

Figure 13. This image depicts an exemplary instantiation of the zone extension presented in Figure 12. It shows a scenario in which a table is stored four times in the data lake within four zones. There is a data entity for each of the four tables pointing to its location. The zoneIndicators and the links illustrate each table's zone together with the information from where and when it was imported. The metadata entities indicate that metadata may be attached to the data entities.

arrow. As depicted, the data has moved from the raw zone into the trusted zone and then into the refined zone. It was also loaded into the sandbox. The importedFrom attribute specifies that the data element in the sandbox was not loaded from any zone in the data lake, but from the source directly.

These zones could also be modeled differently, e.g., such that each zoneIndicator is connected to the zoneIndicator from which the data was imported. In this case, the TrustedZone indicator would have a link to the rawZone indicator and the refinedZone indicator a link to the trustedZone indicator. However, with this alternative design the sandbox indicator might not have any connection to any other indicator when its data was imported from the data source directly, which is why we chose the design as described above as we want to have all instances of the same dataset to be connected.

5.4 The Categorization Extension

Figure 14 illustrates the intended usage of the *categorization* entity, exemplified using the metadata categorization by Gröger and Hoos [11]. Like the zone and granularityIndicator, the *categorization*

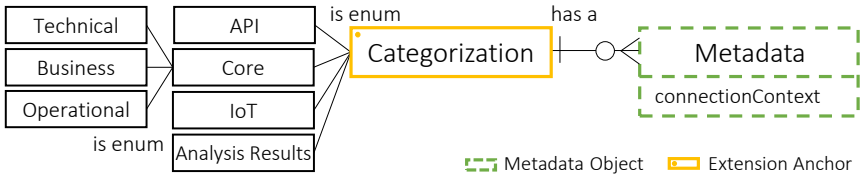

Figure 14. Categorization Extension to Core Model, with the categorization by Gröger and Hoos [11] and subcategories they adopted from [3].

entity is a label assigned according to the metadata element's context. For instance, access information is core metadata and therein operational metadata as defined by [3], and thus a metadata element storing any type of access information will have an operational label. This extension together with the granularity and zone extension as well as the core model add up to be HANDLE.

6 HANDLE Assessment

To asses HANDLE's suitability as a generic metadata model we assess its applicability to a metadata management use case and its implementation aspects. Furthermore, we examine whether it fulfills the requirements specified for a generic metadata model in Section 4 and we compare it to the five metadata models discussed in Section 3.

6.1 HANDLE Demonstration on Access-Use-Case

Figure 15 shows an example instantiation of HANDLE. The depicted model belongs to the access-use-case described in Section 3.2.

As defined through the core model, a *data* instance with zone and granularityIndicator as well as three metadata instances, *action, actor* and *accessInfo*, with the categorization *operational*, are introduced in Figure 15. A data entity has zero or exactly one accessInfo entity. In order to avoid the data element being overloaded by indefinitely increasing access information, all access related nodes are connected to the accessInfo entity as an intermediate node. The accessInfo entity is a way of adding structure. The model suggests that an action element is created for every executed action. It is connected to the involved data's accessInfo element and stored with the time it was performed. The term access covers a variety of actions, such as create, read, update or delete actions. An action is performed by an actor who is connected to one or many actions. For instance, a specific data scientist may repeatedly load data from the customer table. The accessInfo element for the customer table will have one actor element with the data scientist's name and id. This actor element will be connected to *read* actions, one for every time they loaded the data with the according time.

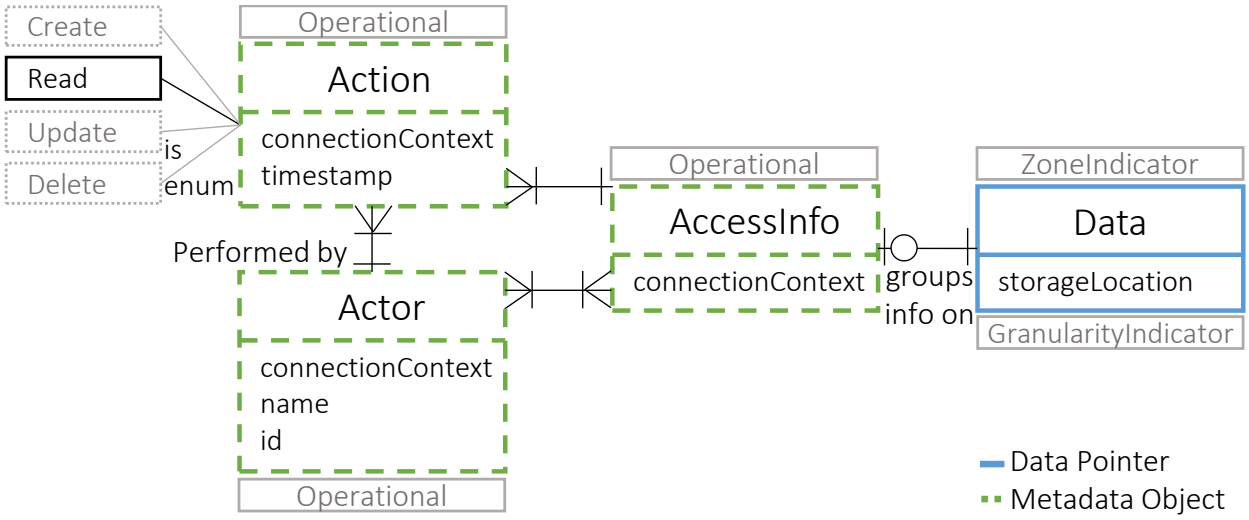

Figure 15. Instance of HANDLE for Access-Use-Case

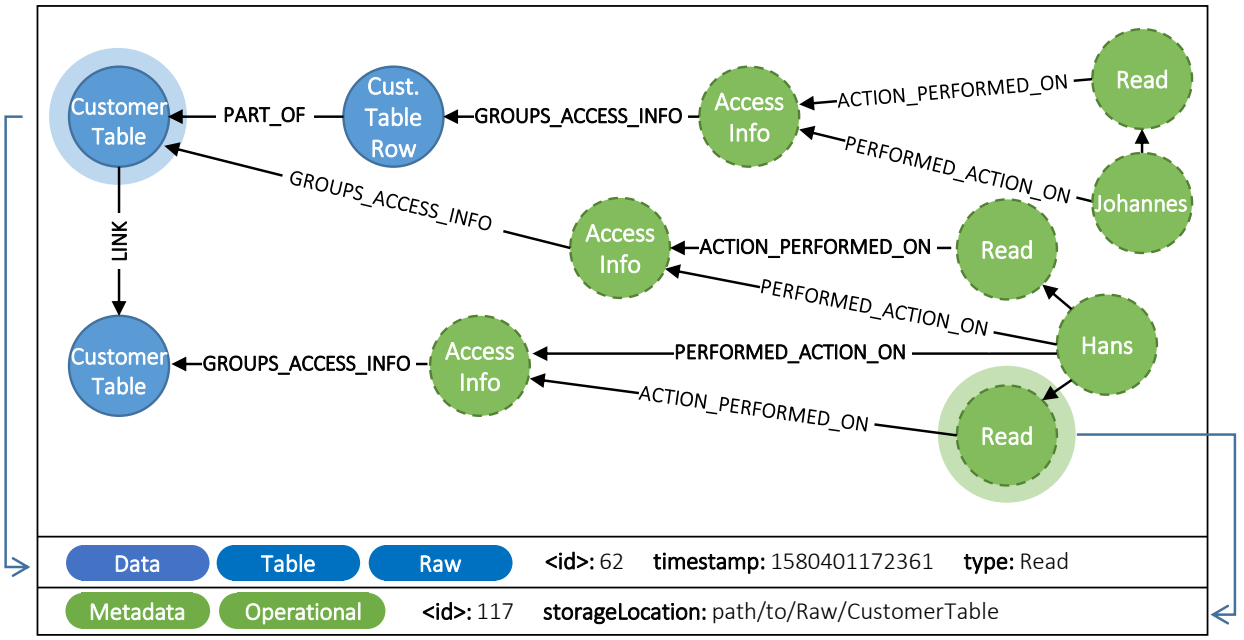

Figure 16. Visualization of the HANDLE access-use-case implementation in Neo4J. The solid/blue and dashed/green nodes represent data and metadata objects respectively. The two highlighted node's labels are depicted on the bottom left, e.g., Data, Table and Raw. The elements' properties are listed next to the labels. The text on the edges is the metadata entity's connectionContext attribute, e.g., actor "Johannes" performed an action and is connected to the data's accessInfo with the connectionContext "performed_action_on".

6.2 Prototypical Implementation

Apart from HANDLE's applicability, we assess its realizability, that is, whether it can be properly implemented. As previously emphasized, flexibility is one of the most important features of the new metadata model. This poses an additional challenge during the implementation as the system components must be able to reflect this flexibility. More specifically, the database of choice must support the aspects of the model which constitute its flexibility. This section shows by way of example that a graph database provides the necessary prerequisites for implementing HANDLE. As a graph database is a NoSQL database it does not require a predefined schema, which makes it more flexible than the traditional relational databases [19]. Also, it is well suited for storing strongly linked data, which is required for many metadata management use cases such as the access-use-case described above, lineage-use-case etc. In the following example, we use the graph database Neo4J[1].

Figure 16 illustrates an implementation of the access model and thus of the core model, as well as aspects of the zone, granularity and categorization extensions. It depicts an extract of a Neo4J graph database and therefore a set of nodes and edges each with labels. Neo4J supports adding properties and labels to both nodes and edges [21]. The labels and properties of one data instance and one metadata instance are displayed at the bottom of the picture, wherein the labels are on the left and properties on the right. The three solid/blue nodes are instantiations of the *data* entity and each have the property *storageLocation* containing the path to the according data element, here the customer table. The *granularityIndicators* introduced in Figure 10 are implemented through labels on the data elements. For example, the highlighted data element on the top left hand side has the label "Table". The solid/blue data element on the top right points to a row and thus

[1] https://neo4j.com/

has the label "Row". The row and table pointers are interconnected according to the hierarchy of granularityIndicators defined in Figure 11, meaning the row pointer is connected to the table pointer through a "part_of" relation. This relation enables the inheritance of metadata as described in Section 5.2. The *zoneIndicators* presented in Section 5.3 are also implemented through a label on the data elements, as exemplified in Figure 13. For instance, Figure 16 lists the label "Raw" for the highlighted data element. The zone extension's *link* entity is implemented through an edge in the graph with the according properties. As can be seen in Figure 16, the two solid/blue nodes on the left are connected through an edge with the label "Link". The link connects the bottom data element to its according instance in the raw zone. The dashed/green nodes are instances of the core model's *metadata* entity. They are also instances of the access-use-case model's metadata entities: *accessInfo*, *actor* and *action*. The metadata object's *connectionContext* is realized as a label on their relations, e.g., the actor elements' "performed_action_on" and accessInfo elements' "groups_access_info" connectionContext describe the relation to the according data object. As can be seen, the actors "Hans" and "Johannes", on the far right in Figure 16, have performed "Read" actions on data elements. "Johannes" read information on a particular customer stored in the raw zone. "Hans" read the entire customer table in both the raw zone in its unpseudonymised state and in another zone, in its pseudonymised state, as indicated by Figure 5. The *categorization* entity is also implemented as a label, e.g., the highlighted "Read" action's "Operational" label can be seen in Figure 16.

6.3 Fulfillment of Requirements

In the following we demonstrate that HANDLE fulfills the requirements for a metadata model listed in Section 4 and the requirements for the access-use-case, as defined in Section 3.2. Furthermore, we discuss whether it is use case and domain independent, and to what extent it fulfills the metadata management requirements.

Requirements for a Generic Metadata Model: To begin with, Requirement (1), enabling flexible modeling, comprises the six sub-requirements (1.1)-(1.6). As prescribed by (1.1), the core model allows the creation of metadata objects with properties. This can be seen in the access-use-case model in Figure 15 through, e.g., the actor entity and its properties name and id. The core model also allows to interconnect metadata objects and data objects, facilitating the wanted relationships in (1.1). As defined per (1.2), the core model does not restrict the amount of metadata objects created and thus, any use case can have an arbitrary number of metadata objects. Equally, metadata properties can be created freely for metadata objects, as required by (1.3). In the access-use-case this is reflected by the three metadata entities action, actor and accessInfo, which have varying properties and can have numerous instantiations. Metadata objects may or may not be connected to a data element, thereby fulfilling (1.4). A metadata entity such as dataLakeDescription may be created for instance, containing general information on a data lake, which need not be connected to any data entity. The self-link of both the metadata and data objects enable the required interconnection of these objects, defined in (1.5) and (1.6). For instance, in the access-use-case the metadata entity actor is connected to another metadata entity called accessInfo which is connected to a data element that may also be connected to other data elements to represent, e.g., a partOf relation. Requirement (2), denoting the support of multiple granular levels, is realized by creating multiple data objects, containing a path to more or less granular elements of a dataset, labeled through the granularityIndicator. If, for example, the granular levels shown in Figure 10 are used, the data element in the access-use-case may point to a table, a row or other level. Requirements (3) and (4), denoting the support of zones and any categorization, are supported through the zoneIndicator and categorization entities, as explained in Section 5. Using Zaloni's zones as shown

in Figure 12 the data element in the access-use-case may for example point to a table in the raw zone as specified by its zoneIndicator. Similarly, the access-use-case's metadata objects carry an *operational* label in Figure 15 according to the categorization shown in Figure 14. In conclusion, HANDLE supports all of the requirements for a generic metadata model as specified in Section 4.

Requirements for the Access-Use-Case: HANDLE also meets the requirements of the access-use-case as described in Section 3.2. According to these the model must be flexible in creating metadata properties for metadata objects to reflect most varied information, the model must support data lake zones and it must support the collection of metadata on various granular levels. These properties are addressed through the model requirements (1.2) and (1.3) on metadata objects and properties and requirements (2) on granular levels and (3) on data lake zones. Given that HANDLE fulfills these, as described in the previous paragraph, it is suited to model the access-use-case which sets it apart from the existent metadata models which are not suited, as shown in Section 3.3. An exemplary instantiation of HANDLE for this use case is given in Section 6.1.

Use Case and Domain Independence: The high abstraction layer in which the metadata object and its properties are defined in HANDLE's core model does not only enable modeling metadata from various use cases, but also from differing domains. As indicated in Figure 9, metadata instances can be created that represent different machines, but in the same way metadata instances can be created that represent proteins and their amino acids. It is irrelevant for the instantiation of the core model whether the metadata originate from the biology or manufacturing domain. It is therefore possible to reflect metadata from different domains and consequently our model is domain independent.

Model Placement in the Metadata Management Requirements: As explained in Section 2 the execution of the data management tasks, data governance, lifecycle management and foundational activities are also required within metadata management. The model we created provides the basis for the required metadata model in the data lifecycle's "Design" step. However, it does not fully satisfy the metadata model as the required instances of HANDLE still need to be designed by domain experts. In this regard HANDLE serves as an enabler for metadata modeling and only satisfies the metadata modeling step in combination with its instances. To which extent the instances of HANDLE are comprehensive must be assessed by domain experts. For example, by comparing the collected metadata with a company-specific list of metadata management use cases or using the categorization label to check if metadata of all specified types are collected. Besides the categorization label, HANDLE does not offer support in assessing the completeness, as the valuation of this will vary between companies and the focus of the model is to create a basis for modeling metadata with which the most diverse use cases can be reflected, as opposed to a guideline which use cases must be reflected.

6.4 Comparison to Existent Models

To further asses HANDLE's generic extent we also compare it to the five selected metadata models. HANDLE can represent the content of all five models through the core model because it is defined on a higher abstraction level. It addresses the use cases in a more general way and can represent any metadata through its abstract entities: *data, metadata* and *property*. This means that metadata stored according to one of the existent models can be transferred and mapped into HANDLE and possibly even combined with metadata stored through yet another model. Besides representing their content, HANDLE adds additional features such as the granularityIndicator, zonIndicator and categorization label. We exemplary demonstrate how HANDLE can represent the content of other models, using two of the existent metadata models that are specified on different levels of abstraction and in varying detail. First, we map HANDLE to the model by Ravat and Zhao as this

model is not abstract, but very concrete and specified in great detail. The second model to which we compare HANDLE is GEMMS as this model is specified on a higher abstraction level giving more flexibility to its instances, similar to HANDLE. By comparing HANDLE to these models we also demonstrate how HANDLE relates to models that are specified differently.

Comparison of HANDLE to the Model by Ravat and Zhao: The mapping of HANDLE to the model by Ravat and Zhao [23] is pictured in Figure 17. The image shows an extract of the model with dashed lines where more elements are originally attached. Most of the elements in Ravat and Zhao's model can be represented by the metadata object and properties, shown in dashed/green. As the location of the data is stored separately in the data object with HANDLE, the attribute "ds_location", which we assume is short for dataset location, was extracted from the "Dataset" entity and is now used equivalently to the storageLocation attribute in the newly added data object. To be conform with HANDLE the connectionContext attribute was added to each metadata object to describe its content or relation to other elements. In the original model by Ravat and Zhao an object called "Relationship" was attached in between the "Datalake_Datasets" self-link. It is used to represent various types of relationships between the "Datalake_Dataset" objects, such as provenance, logical clusters or content similarity [23]. The Relationship-object becomes negligible through the "Datalake_Datasets"'s connectionContext attribute as this is also used to describe interconnections and can therefore also contain, e.g., content similarity. Overall, the content of Ravat and Zhaos model can be represented through HANDLE.

HANDLE's granularityIndicator, zoneIndicator and categorization label can be partially applied to Ravat and Zhao's model. The granularityIndicator can be added to the data object to indicate what it represents, e.g., a table or perhaps video. The relation in between data objects is, however, attached to the "Datalake_Dataset" object in the original model and therefore cannot be represented as intended by HANDLE. The zoneIndicator could also be added to the data object according to Ravat and Zhaos four zones: the raw-data-zone, process-zone, access-zone and governance-zone. Ravat and Zhao only differentiate between *inter-* and *intra-metadata*, which refer

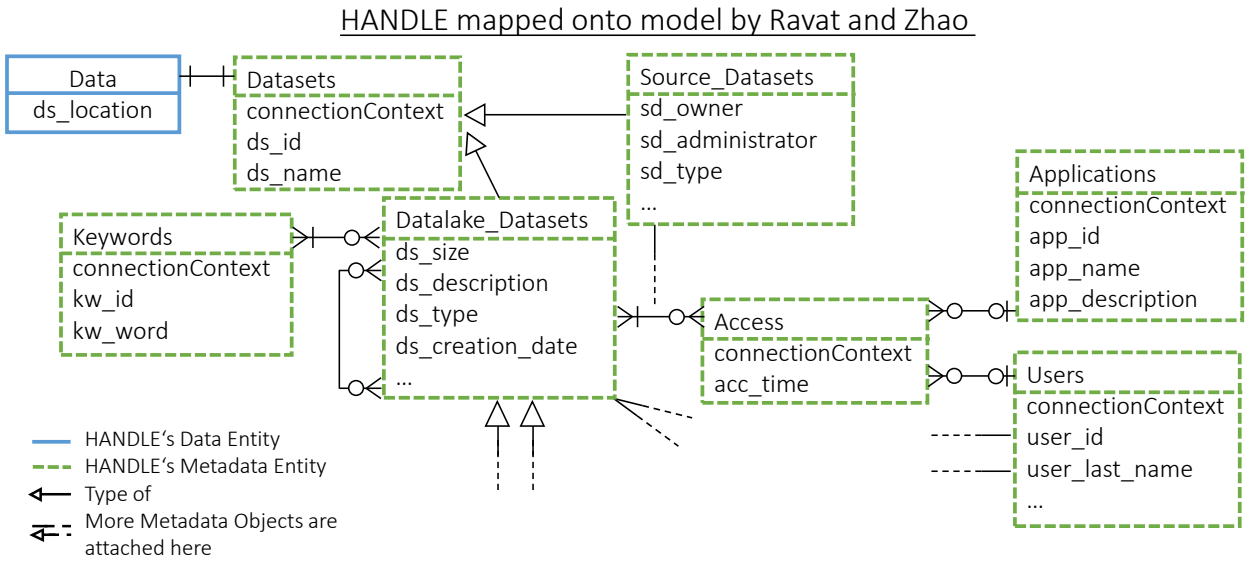

Figure 17. The image illustrates how the model by Ravat and Zhao [23] can be represented through HANDLE. The entities are color matched to the entities in HANDLE's core model. Only an extract is displayed, for the sake of a better overview. The missing elements are indicated by the black dashed lines attached to the metadata entities and can be modeled exactly like the Users, Access and Datalake_Datasets entities.

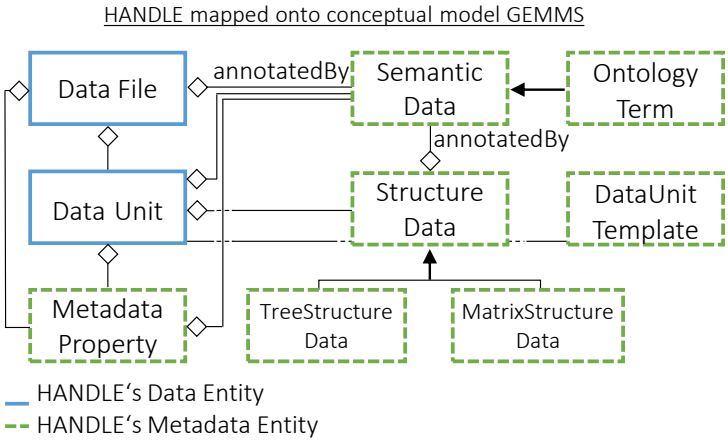

Figure 18. The image depicts the model of GEMMS [22], which' entities are color matched to the entities in HANDLE's core model.

to the object relations and attributes respectively. In this case, each metadata-object would have a categorization called intra-metadata. The inter-metadata categorization would have to be applied to the "Datalake_Datasets"'s relations which is not intended by HANDLE. Therefore, HANDLE's categorization cannot be used in this case. Nonetheless, additional metadata management capabilities can be added by applying HANDLE's zone- and granularityIndicators to the model by Ravat and Zhao.

Comparison of HANDLE to GEMMS: Figure 18 exemplifies how GEMMS' model can be mapped onto HANDLE. The colors indicate that the solid/blue elements are an instance of the core model's data entity and the dashed/green ones instances of the metadata entity. All of GEMMS' entities can be represented through the core model's data, metadata and property entities. In contrast to GEMMS, HANDLE strictly separates data and metadata, therefore the metadata is not stored within the "Data File" or "Data Unit" entities but in the "Metadata Property" nodes.

Furthermore, HANDLE's categorization and granularity topics can be integrated, hereby adding some of HANDLE's features to GEMMS. As can be seen in Figure 19 showing an example instantiation of GEMMS through HANDLE, the "Data File" and "Data Unit" instantiations are labeled

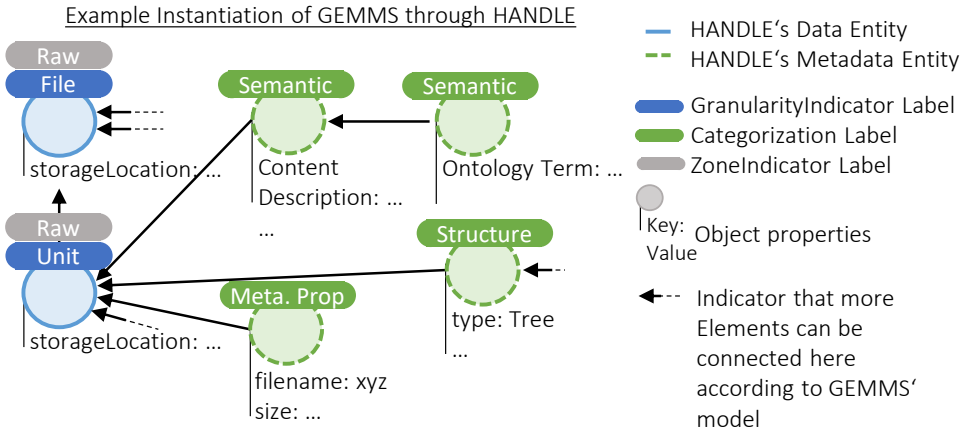

Figure 19. An instantiation of GEMMS through HANDLE is illustrated. The image shows that HANDLE can represent GEMMS content and adds features, e.g., the zoneIndicator, shown on the "File" entity in grey.

with the granularityIndicators “File” and “Unit”. According to GEMMS’ metadata types, the categorization labels are added to the metadata instantiations on the right hand side, “Semantic”, “Structure” and “Metadata Property”. Although GEMMS does not address the zone concept, HANDLE’s zoneIndicator can be attached, as shown through the grey “Raw” label on the “File” entity. Thereby GEMMS is extended and becomes compatible with zones.

The other three models’ content can be represented through HANDLE in a similar fashion, by mapping their main entities onto HANDLE’s data, metadata and property elements. Having demonstrated that HANDLE fulfills the specified requirements, can represent the content of other metadata models, and extends these with features required for metadata management in data lakes, it can be said that HANDLE is more comprehensive and is a generic metadata model which can reflect any metadata management use case and consequently any metadata.

7 Conclusion

In order to exploit the value of data in data lakes, metadata is required. Metadata is a type of data, which provides information on, e.g., other data, processes or systems. As it is a type of data it also needs to be managed. Metadata management is essentially data management for metadata and therefore, involves the data management activities, data governance, lifecycle management and foundational activities like data quality management. One of these activities involves the design of a metadata model, which defines the relation in between data and metadata elements and which metadata can be stored. As metadata is required for the management of all other data types, and throughout each of their management activities, there is a multitude of metadata which needs to be collected. The above mentioned metadata model must therefore be generic, meaning that it should be able to reflect any given metadata management use case and consequently all metadata.

We conducted a literature research and found five comprehensive metadata models which are created for or are applicable to data lakes. For each of these models, the generic extent and the approach through which it was built, were examined. Thereby we found that the models were built based on two general approaches which involve a metadata categorization and/or a list of metadata management features which should be supported. Our assessment of these approaches yields that both are not fully suited for creating a generic metadata model. Consequently, the existent models do not fulfill the required generic extent, as also demonstrated through an exemplary use case in an Industry 4.0 scenario.

A new approach was used to develop a new metadata model for data lakes, called HANDLE. The new model supports the acquisition of metadata on varying granular levels, any metadata categorization, including the acquisition of both metadata which belongs to a specific data element as well as metadata which applies to a broader range of data. HANDLE can also be used to model the same metadata in various ways according to the intended utilization. Most importantly, HANDLE is explicitly created for data lakes and therefore also supports data lake characteristics like data lake zones. Our assessment shows that it is easily applicable to metadata management use cases, can be implemented through a graph database, can reflect the content of existent metadata models and offers additional metadata management features. As the research has demonstrated, it is the most generic metadata model for data lakes up to date. In future, we intend to investigate how HANDLE can be expanded to provide decision-support for domain experts with respect to what metadata needs to be collected for comprehensive metadata management and also whether HANDLE is applicable beyond the scope of data lakes, e.g., to other systems like data warehouses or whether it is applicable across a combination of various data storage systems.

References

1. ISO/IEC 11179-1: Information technology — Metadata registries (MDR) — Part 1: Framework. Tech. rep., International Organization for Standardization (2004)
2. Cao, L.: Data science: A comprehensive overview. ACM Comput. Surv. **50**(3) (2017). https://doi.org/10.1145/3076253
3. DAMA International: DAMA-DMBOK: Data Management Body of Knowledge. Technics Publications (2017)
4. Diamantini, C., Giudice, P.L., Musarella, L., Potena, D., Storti, E., Ursino, D.: A new metadata model to uniformly handle heterogeneous data lake sources. In: Proc of the 22nd European Conference on Advances in Databases and Information Systems (ADBIS 2018). pp. 165–177. https://doi.org/10.1007/978-3-030-00063-9_17
5. Eichler, R., Giebler, C., Gröger, C., Schwarz, H., Mitschang, B.: Handle - a generic metadata model for data lakes. In: Proceedings of the 22nd International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2020). pp. 73–88. https://doi.org/10.1007/978-3-030-59065-9_7
6. Erwin: Metadata Management : The Hero in Unleashing Enterprise Data ' s Value Connect all the pieces of your data. Tech. rep., erwin (2020)
7. GDPR.EU: Art. 15 GDPR - Right of access by the data subject, https://gdpr.eu/article-15-right-of-access/, last accessed 2020-02-28
8. GDPR.EU: What is GDPR, the EU's new data protection law?, https://gdpr.eu/what-is-gdpr/, last accessed 2020-02-28
9. Giebler, C., Gröger, C., Hoos, E., Eichler, R., Schwarz, H., Mitschang, B.: Data lakes auf den grund gegangen. Datenbank-Spektrum pp. 1–13 (2020). https://doi.org/10.1007/s13222-020-00332-0
10. Giebler, C., Gröger, C., Hoos, E., Schwarz, H., Mitschang, B.: Leveraging the Data Lake: Current State and Challenges. In: Proc of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2019). pp. 179–188. https://doi.org/10.1007/978-3-030-27520-4_13
11. Gröger, C., Hoos, E.: Ganzheitliches Metadatenmanagement im Data Lake: Anforderungen, IT-Werkzeuge und Herausforderungen in der Praxis. In: Proc of the 18. Fachtagung für Datenbanksysteme für Business, Technologie und Web (BTW 2019). https://doi.org/10.18420/btw2019-26
12. Hai, R., Geisler, S., Quix, C.: Constance: An intelligent data lake system. In: Proc of the 2016 International Conference on Management of Data (SIGMOD 2016). pp. 2097–2100. https://doi.org/10.1145/2882903.2899389
13. Hai, R., Quix, C., Wang, D.: Relaxed Functional Dependency Discovery in Heterogeneous Data Lakes. In: Proc of the 39th International Conference on Conceptual Modeling (ER 2019). pp. 225–239
14. Halevy, A., Korn, F., Noy, N.F., Olston, C., Polyzotis, N., Roy, S., Whang, S.E.: Managing Google's data lake: an overview of the Goods system. IEEE Data Eng. Bull. **39**(3), 5–14 (2016)
15. Hellerstein, J.M., Sreekanti, V., Gonzalez, J.E., Dalton, J., Dey, A., Nag, S., Ramachandran, K., Arora, S., Bhattacharyya, A., Das, S., Donsky, M., Fierro, G., She, C., Steinbach, C., Subramanian, V., Sun, E.: Ground : A Data Context Service. In: Proc of the 8th Biennial Conference on Innovative Data Systems Research (CIDR 2017)
16. Isuru, S., Plale, B.: Provenance as Essential Infrastructure for Data Lakes. In: Proc of the 6th International Provenance and Annotation Workshop (IPAW 2016). pp. 178–182. https://doi.org/10.1007/978-3-319-40593-3
17. Kandogan, E., Roth, M., Schwarz, P., Hui, J., Terrizzano, I., Christodoulakis, C., Miller, R.J.: LabBook: Metadata-driven social collaborative data analysis. In: Proc of the IEEE International Conference on Big Data (Big Data 2015). pp. 431–440. https://doi.org/10.1109/BigData.2015.7363784
18. Kassner, L., Gröger, C., Königsberger, J., Hoos, E., Kiefer, C., Weber, C., Silcher, S., Mitschang, B.: The Stuttgart IT Architecture for Manufacturing. In: Hammoudi S., Maciaszek L., Missikoff M., Camp O., C.J. (ed.) Enterprise Information Systems (ICEIS 2016). Revised Selected Papers, pp. 53–80. Springer (2017). https://doi.org/10.1007/978-3-319-62386-3_3
19. Kaur, K., Rani, R.: Modeling and querying data in NoSQL databases. In: Proc of the IEEE International Conference on Big Data (Big Data 2013). pp. 1–7. https://doi.org/10.1109/BigData.2013.6691765
20. LaPlante, A., Sharma, B.: Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases. O'Reilly Media (2018)
21. Neo4j: The Definitive Guide to Graph Databases for the RDBMS Developer. Tech. rep. (2016)
22. Quix, C., Hai, R., Vatov, I.: Metadata Extraction and Management in Data Lakes With GEMMS. Complex Systems Informatics and Modeling Quarterly (9), 67–83 (2016). https://doi.org/10.7250/csimq.2016-9.04
23. Ravat, F., Zhao, Y.: Metadata Management for Data Lakes. In: Proc of the 23rd European Conference on Advances in Databases and Information Systems (ADBIS 2019). pp. 37–44. https://doi.org/10.1007/978-3-030-30278-8_5
24. Sawadogo, P., Darmont, J.: On data lake architectures and metadata management. Journal of Intelligent Information Systems (2020). https://doi.org/10.1007/s10844-020-00608-7

25. Sawadogo, P.N., Kibata, T., Darmont, J.: Metadata management for textual documents in data lakes. In: Proc of the 21st International Conference on Enterprise Information Systems (ICEIS 2019). pp. 72–83. https://doi.org/10.5220/0007706300720083
26. Sawadogo, P.N., Scholly, É., Favre, C., Ferey, É., Loudcher, S., Darmont, J.: Metadata Systems for Data Lakes: Models and Features. In: Proc of the 23rd European Conference on Advances in Databases and Information Systems (ADBIS 2019). pp. 440–451. https://doi.org/10.1007/978-3-030-30278-8_43
27. Simoni, G.D., Dayley, A., Edjlali, R.: Magic Quadrant for Metadata Management Solutions (2018)
28. Skluzacek, T.J.: Dredging a data lake: Decentralized metadata extraction. In: Proceedings of the 20th International Middleware Conference Doctoral Symposium (Middleware 2019). p. 51–53. https://doi.org/10.1145/3366624.3368170
29. Spiekermann, M., Tebernum, D., Wenzel, S., Otto, B.: A metadata model for data goods. In: Proc of the Multikonferenz Wirtschaftsinformatik (MKWI 2018). pp. 326–337
30. Theodorou, V., Hai, R., Quix, C.: A Metadata Framework for Data Lagoons. In: Proc of the 23rd European Conference on Advances in Databases and Information Systems (ADBIS 2019). pp. 452–462. https://doi.org/10.1007/978-3-030-30278-8_44
31. Walker, C., Alrehamy, H.: Personal Data Lake with Data Gravity Pull. In: Proc of the 5th International Conference on Big Data and Cloud Computing (BDCloud 2015). pp. 160–167. https://doi.org/10.1109/BDCloud.2015.62
32. Yu, X., Wen, Q.: A view about cloud data security from data life cycle. In: 2010 International Conference on Computational Intelligence and Software Engineering, CiSE 2010. pp. 1–4. https://doi.org/10.1109/CISE.2010.5676895
33. Zaloni: The Data Lake Reference Architecture - Leveraging a Data Reference Architecture to Ensure Data Lake Success. Tech. rep. (2018)
34. Zitron, G., Yarmoluk, D.: Metadata Management as a Strategic Imperative – Towards Data Science. Tech. rep., Octopai (2017), https://towardsdatascience.com/metadata-management-as-a-strategic-imperative-88a16c6ec731