University of Stuttgart
Germany

HANDLE - A Generic Metadata Model for Data Lakes

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

In: Big Data Analytics and Knowledge Discovery (DaWaK) 2020

BIBTEX:
@inproceedings{eichler2020handle,
title={HANDLE-A Generic Metadata Model for Data Lakes},
author={Eichler, Rebecca and Giebler, Corinna and Gr{\"o\}ger,Christoph and Schwarz, Holger and Mitschang, Bernhard},
booktitle={International Conference on Big Data Analytics and Knowledge Discovery (DaWaK) 2020},
pages={73–88},
year={2020},
organization={Springer},
doi={10.1007/978-3-030-59065-9_7}
}

© by Springer Nature
This is the author's version of the work. The final authenticated version is available online at: https://doi.org/10.1007/978-3-030-59065-9_7

HANDLE - A Generic Metadata Model for Data Lakes

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 The substantial increase in generated data induced the development of new concepts such as the data lake. A data lake is a large storage repository designed to enable flexible extraction of the data's value. A key aspect of exploiting data value in data lakes is the collection and management of metadata. To store and handle the 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. In this work, we present HANDLE, a generic metadata model for data lakes, which supports the flexible integration of metadata, data lake zones, metadata on various granular levels, and any metadata categorization. With these capabilities HANDLE enables comprehensive metadata management in data lakes. We show HANDLE's feasibility through the application to an exemplary access-use-case and a prototypical implementation. A comparison with existent models yields that HANDLE can reflect the same information and provides additional capabilities needed for metadata management in data lakes.

Keywords: Metadata Management · Metadata Model · Data Lake.

1 Introduction

With the considerable increase in generated data, new concepts were developed for exploiting the value of this data, one of which is the data lake concept. In this concept an organization's data is incorporated in one big data repository [7]. It is a storage concept designed for data at scale, that integrates data of varying structure, from heterogeneous sources, in its raw format. The focus of the concept is to enable flexible extraction of the data's value for any potential use case.

In order to exploit the data's value, metadata is required [1]. Metadata can be used to document various aspects of the data such as the meaning of its content, information on data quality or security, data lifecycle aspects and so on. Just like any other data, metadata needs to be managed. Metadata management constitutes activities which involve managing an organizations' knowledge on its data [1]. Without this knowledge, data may not be applicable for the intended purpose, e.g., due to a lack of quality or trust.

A central aspect of metadata management is the definition of a metadata model (e.g., [10,15,17]). 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. In order to store all kinds of knowledge on the data to increase its value, a generic metadata model is required. To be generic, a metadata model must reflect any potential metadata management use case of a data lake. This includes standard use cases, e.g., the collection of lineage information, as well as organization-specific use cases, e.g., use cases for the manufacturing domain. It follows that the generic metadata model can represent all metadata regardless of its type.

However, existent metadata models, e.g., [2,16,18], 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 [8,11,21]. 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 introduce a new approach for constructing a generic metadata model by investigating existent models and their shortcomings. (2) Based on this approach, we developed a generic metadata model called HANDLE, short for Handling metAdata maNagement in Data LakEs. (3) 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.

This paper is structured as follows. Related work is introduced and discussed in Section 2. Section 3 specifies the requirements for the new metadata model, which is presented in Section 4, followed by an assessment of it in Section 5. Lastly, the paper is concluded by Section 6.

2 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) [15], for Walker and Alrehamy's personal data lake solution [22], and lastly, the metadata model for the data context service Ground [10]. Many systems both in the research context as well as commercial metadata management systems do not disclose their metadata model and thus we cannot examine their generic extent, e.g., [7,9,12]. Other models exist which are not part of a specific system. However, many of these, also including that by Walker and Alrehamy, only focus on a specific topic and thus, only support a limited set of use cases which makes them non-generic, e.g., [8,11,17,20,21]. Thenceforth these models are not considered here. More general models also created for data lakes include those by Ravat and Zhao [16], Diamantini et al. [2], and lastly, Sawadogo et al.'s model MEtadata for DAta Lakes (MEDAL) [18].

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 2.1 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 2.2 and Section 2.3 shows that it cannot be realized by the existing models, thereby demonstrating that these are not sufficiently generic.

2.1 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 1, 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 2, some of the lists contain high-level and some detailed feature descriptions, making it impossible to combine them. Defining high-level features might not suffice to derive all necessary requirements for a

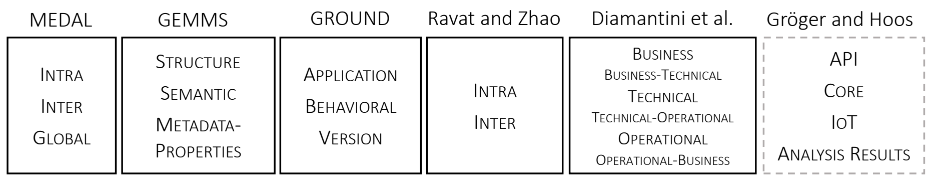
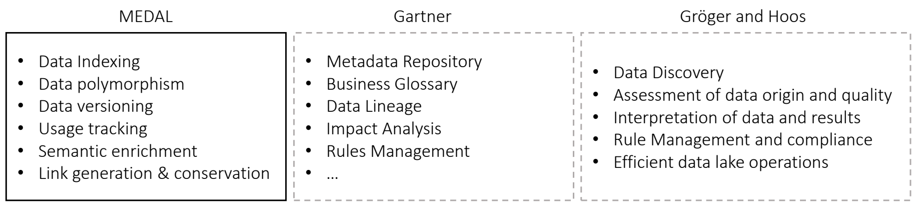


Figure 1. Set of metadata categorizations, the first five belong to the selected metadata models [18,15,10,16,2]. The sixth, in dashes, does not belong to a metadata model [6].



MEDAL

- Data Indexing
- Data polymorphism
- Data versioning
- Usage tracking
- Semantic enrichment
- Link generation & conservation

Gartner

- Metadata Repository
- Business Glossary
- Data Lineage
- Impact Analysis
- Rules Management
- …

Gröger and Hoos

- Data Discovery
- Assessment of data origin and quality
- Interpretation of data and results
- Rule Management and compliance
- Efficient data lake operations

Figure 2. This is a display of three lists of metadata management features. The first belongs with the model MEDAL [18], whereas the other two, in dashes, by Gartner [19], and Gröger and Hoos [6] are created independent of a metadata model.

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 2.

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

2.2 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 [13] 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) [4]. Conformity to the GDPR requires the collection of information on the personal data's use and the users accessing it [3]. 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.

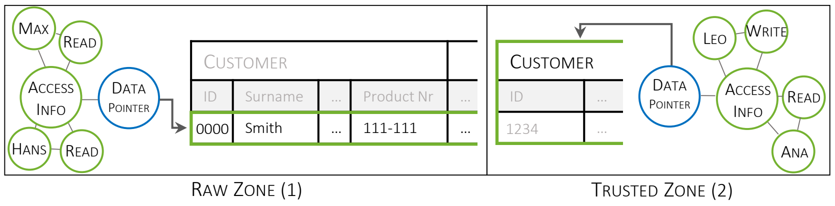


Figure 3. The image displays access metadata collected on varying granular levels of the customer table. The customer table is stored twice, once in each data lake zone. The green circles are metadata objects with properties, e.g., a user object with the name "Max". They belong to the accordingly highlighted part of the table. The blue circles denote a pointer containing the data's storage location. (Colored figure online)

Within our Industry 4.0 scenario, access information is collected amongst other things on a table containing customer data, as depicted in Figure 3. 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 3 illustrates metadata objects, in 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 blue “Data Pointer” object.

Data within data lakes is often organized in so-called zones according to its processing degree [5], e.g., as standardized data in the trusted zone [23]. 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 information and who only saw the customer’s pseudonym. For example, in Figure 3 “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.*

2.3 Assessing the Generic Extent of the Existent Models

Within this section, the five models selected in the beginning of Section 2 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.

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	($\surd$)			($\surd$)	

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 [2,18]. 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.

3 Requirements for a Generic Metadata Model

Section 2 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 reflects a broader scope of use cases. 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* [5]. 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 1). 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.

4 HANDLE - A Generic Metadata Model

We used the requirements given in Section 3 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, thereby getting a handle on all metadata.

The conceptual metadata model consists of two parts, the core model, illustrated in Figure 4, 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. The core model extensions each address the zone, granularity and categorization topics in more detail, according to the requirements 2-4. All of the models are modeled according to the Crow's Foot Notation.

As depicted in Figure 4, one of the core models main entities is the *data* entity, illustrated in 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 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

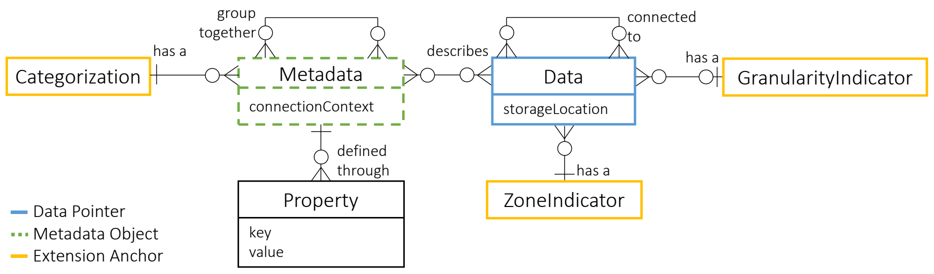


Figure 4. HANDLE's Core Model (Colored figure online)

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 3. 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.

The Granularity Extension: The granularityIndicator enumerations have to be adapted according to the specific use, e.g., for relational structures, as depicted in Figure 5. Thus, it is 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 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

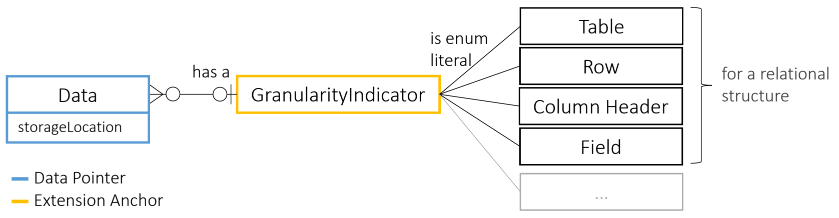


Figure 5. The Granularity Extension to the Core Model (Colored figure online)

case, there would be a video level and a frame level. Domain knowledge 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 5 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 data set, simply pointing to more or less specific granular levels. Demonstrating the granularityIndicators defined here, the "Data Pointer" in Figure 3'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*.

The Zone Extension: Figure 6 illustrates the intended usage of the *zoneIndicator* entity, using the zone model by Zaloni [23]. 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. The *Raw Zone* entity is designed to be the central zoneIndicator, as data is stored in any of the other zones will have a corresponding data element in the raw zone, making the raw zone the most stable reference. The zones depicted on the right hand side 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 as well as the corresponding *timestamp* is stored with the link. The *importedFrom* attribute may contain the name of a zone or the original source. The link and importedFrom attribute enables tracing the data's progress through the zones. By Zaloni's definition, the data may not be moved into the raw zone from the transient loading zone, therefore, this enumeration can exist without a link to the rawZone element [23]. If the data was moved into the raw zone, then it must have a link connecting them.

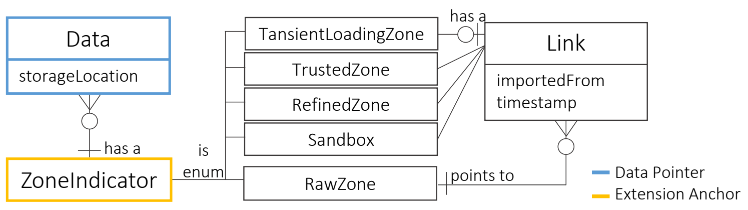


Figure 6. The Zone Extension to the Core Model, Using Zaloni's Zones [23]. (Colored figure online)

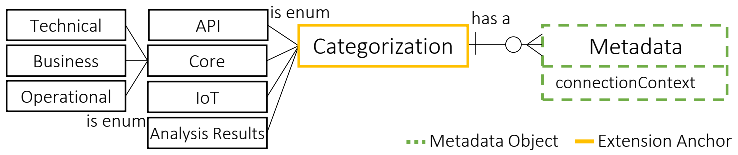


Figure 7. Categorization Extension to Core Model, with the categorization by Gröger and Hoos [6] and subcategories they adopted from [1]. (Colored figure online)

The Categorization Extension: Figure 7 illustrates the intended usage of the *categorization* entity, exemplified using the metadata categorization by Gröger and Hoos [6]. Like the zone and granularityIndicator, the *categorization* 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 [1], 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.

5 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 3 and we compare it to the five metadata models discussed in Section 2.

5.1 HANDLE Demonstration on Access-Use-Case

Figure 8 shows an example instantiation of HANDLE. The depicted model belongs to the access-use-case described in Section 2.2.

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 8. 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.

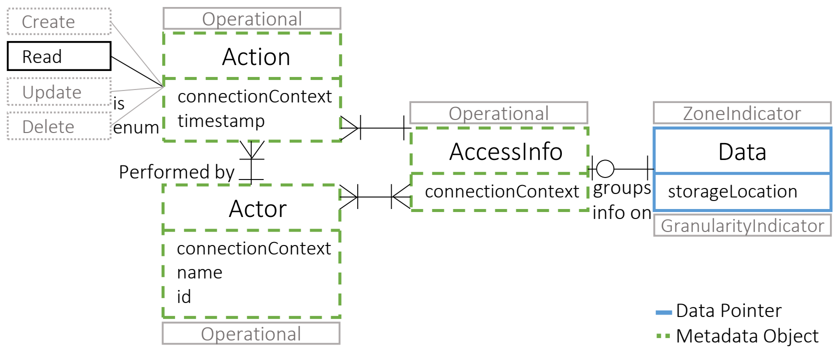


Figure 8. Instance of HANDLE for Access-Use-Case (Colored figure online)

5.2 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 [14]. 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 9 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. The three 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 5 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 blue data element on the top right points to a row and thus has the label "Row". The *zoneIndicators* presented in Figure 6 are also implemented through a label on the data elements. For instance, Figure 9 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 9, the two 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

[1] https://neo4j.com/

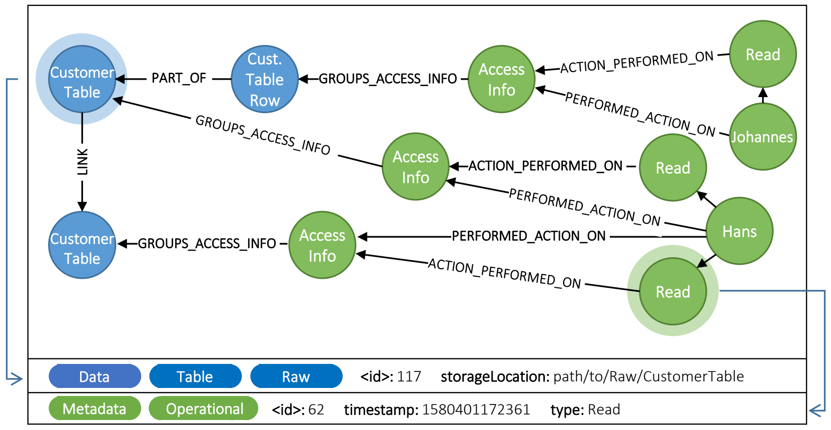


Figure 9. Visualization of the HANDLE access-use-case implementation in Neo4J. The blue and green nodes represent data and metadata objects respectively. The two highlighted node's labels are depicted on the bottom left, e.g., Data, Table and Raw. The elements' properties are listed next to the labels. 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". (Colored figure online)

in the raw zone. The 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 9, 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 3. The *categorization* entity is also implemented as a label, e.g., the highlighted "Read" action's "Operational" label can be seen in Figure 9.

5.3 Fulfillment of Requirements

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. It 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). Metadata objects may or may not be connected to a data element, thereby fulfilling (1.4). The self-link of both the metadata and data objects enable the required interconnection of these objects, defined in (1.5) and (1.6). 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. Requirements (3) and (4), denoting the support of zones and any categorization, are supported through the zoneIndicator and categorization entities, as explained in Section 4. In conclusion, HANDLE supports all of the specified requirements.

5.4 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 GEMMS. Figure 10 exemplifies how GEMMS' model can be mapped onto HANDLE. GEMMS' model is depicted on the left hand side and an example instantiation of it through HANDLE on the right hand side. The colors indicate that the blue elements are an instance of the core model's data entity and the 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,

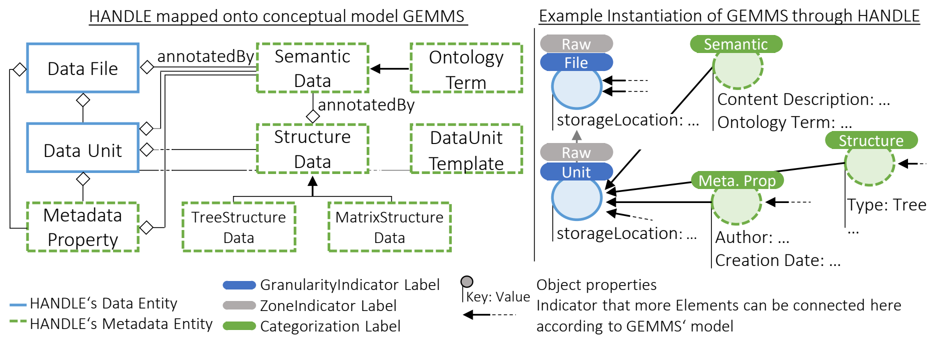


Figure 10. The left side depicts the model of GEMMS [15]. The entities are color matched to the entities in HANDLE's core model. The right side shows an instantiation of GEMMS through HANDLE. The image shows that HANDLE can represent GEMMS content and adds features, e.g., the zoneIndicator, shown on the "File" entity in grey. (Colored figure online)

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 on the right hand side, the “Data File” and “Data Unit” instantiations are labeled 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 four 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.

6 Conclusion

In order to exploit the value of data in data lakes, metadata is required, which in turn needs to be handled through metadata management. One central aspect of metadata management is the design of a metadata model. This metadata model should be generic, meaning it should be able to reflect any given metadata management use case and consequently all metadata.

We selected five comprehensive metadata models and pointed out that the two approaches on which they were built are not suited for creating a generic model. Therefore, 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. 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 whether HANDLE is applicable beyond the scope of data lakes, e.g., in an enterprise-wide federation of data storage systems.

References

1. DAMA International: DAMA-DMBOK: Data Management Body of Knowledge. Technics Publications (2017)
2. Diamantini, C., et al.: 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)

3. 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
4. GDPR.EU: What is GDPR, the EU's new data protection law?, https://gdpr.eu/what-is-gdpr/, last accessed 2020-02-28
5. Giebler, C., et al.: Leveraging the Data Lake: Current State and Challenges. In: Proc of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2019)
6. 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)
7. Hai, R., et al.: Constance: An intelligent data lake system. In: Proc of the 2016 International Conference on Management of Data (SIGMOD 2016)
8. Hai, R., et al.: Relaxed Functional Dependency Discovery in Heterogeneous Data Lakes. In: Proc of the 39th International Conference on Conceptual Modeling (ER 2019)
9. Halevy, A., et al.: Managing Google's data lake: an overview of the Goods system. IEEE Data Eng. Bull. (39), 5–14 (2016)
10. Hellerstein, J.M., et al.: Ground : A Data Context Service. In: Proc of the 8th Biennial Conference on Innovative Data Systems Research (CIDR 2017)
11. Isuru, S., Plale, B.: Provenance as Essential Infrastructure for Data Lakes. In: Proc of the 6th International Provenance and Annotation Workshop (IPAW 2016)
12. Kandogan, E., et al.: LabBook: Metadata-driven social collaborative data analysis. In: Proc of the IEEE International Conference on Big Data (Big Data 2015)
13. Kassner, L., et al.: The Stuttgart IT Architecture for Manufacturing. In: Hammoudi, S., et al. (eds.) Enterprise Information Systems (ICEIS 2016). Revised Selected Papers, pp. 53–80. Springer (2017)
14. Kaur, K., Rani, R.: Modeling and querying data in NoSQL databases. In: Proc of the IEEE International Conference on Big Data (Big Data 2013)
15. Quix, C., et al.: Metadata Extraction and Management in Data Lakes With GEMMS. Complex Systems Informatics and Modeling Quarterly (9), 67–83 (2016)
16. 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)
17. Sawadogo, P.N., et al.: Metadata management for textual documents in data lakes. In: Proc of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)
18. Sawadogo, P.N., et al.: Metadata Systems for Data Lakes: Models and Features. In: Proc of the 23rd European Conference on Advances in Databases and Information Systems (ADBIS 2019)
19. Simoni, G.D., et al.: Magic Quadrant for Metadata Management Solutions (2018)
20. Spiekermann, M., et al.: A metadata model for data goods. In: Proc of the Multikonferenz Wirtschaftsinformatik (MKWI 2018)
21. 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)
22. 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)
23. Zaloni: The Data Lake Reference Architecture - Leveraging a Data Reference Architecture to Ensure Data Lake Success. Tech. rep. (2018)