Component-Based Development of a Metadata Data-Dictionary

Frank Kramer and Bernhard Thalheim

Christian-Albrechts-University Kiel, Computer Science Institute
24098 Kiel, Germany

Abstract. Metadata provide information about data, ease access and querying, support well-planned evolution, and allow to reason on data quality. Moreover, heterogeneous data sources are better to integrate if well-defined metadata are available. Therefore, metadata management becomes a crucial element of modern database systems. We develop a component-approach to metadata management. This approach generalises classical data dictionaries and is based on many-dimensional schemata each dimension of it represents a specific facet of the schema.

Keywords: Conceptual Modeling, Metadata, Component-based Development, Metadata Management, Metadata Repository.

1 Introduction

An important part in the field of data management is the metadata management. Metadata support heterogeneous user groups to get information and assessment over the corresponding data within a system. The management of metadata becomes complicated, if there is a great variability of databases that covers the metadata. Furthermore, there is an evolution in the metadata and the underlying schema over time. Thus, a system is needed that allows a dedicated management of the metadata and that is also safe against evolution of the metadata. Today there exist different approaches to metadata management systems. They can be divided into generic or high specialized approaches. The generic approaches give only suggestions for what is needed for good metadata management systems [1]. Specialized approaches are constructed for special applications such as data warehousing [2] or master data management [3]. Two movements of metadata management can be found often that cover both their own problems. On the one hand, the metadata are integrated into the global schema. Such a schema can become quickly confusing, unreadable and not extensible. On the other hand, the metadata can be disembodied from the application data and worfklow data in an external repository. This leads to a high effort for tending, reading and connecting the metadata with other data within a special context.

Functional approaches for solving problems in large database applications can be found within the field of the *Conceptual Modelling in the Large* (CoMoL). It describes special techniques to model large applications. This covers the structure, functionality, interaction and support for such applications. All described techniques exceed the classical techniques for modelling smaller applications. One facet of CoMol comprises the component-based development of information systems. A component is *a database schema that has an import and an export interface by which it may be connected to other components via a standardized interface technique* [4]. With these components we get a modularization of a schema with all the advantages like loose coupling and abstraction from components on their content. With these components we are able to scale a global schema into a metadata, workflow data and application data dimension. As a result, we reduce the complexity of the whole schema drastically.

This paper presents an approach to metadata management that is based on an internal metadata Data-Dictionary. The dictionary is a generalization of a database data-dictionary. To construct the metadata data-dictionary we define six disjoint categories that can be realized as components. The components build a metadata dimension. The connection between the dimensions is enabled with a concept called harnesses. These harnesses build the metadata data dictionary that corresponds to a metadata repository. As distinct from other repositories, the metadata data-dictionary is not disembodied from the system. Section 2 will give a closer look at creating components. Furthermore, we demonstrate how we can scale data into dimensions based on this components and how the dimensions can be connected together with harnesses. Then, the section 3 will show, how a generalized metadata data-dictionary can be realized. Therefore, we define the six disjoint metadata categories and apply the component-based development on these categories. As a result, we get the metadata data-dictionary. We will show also that such a dictionary meets all requirements of a metadata repository. After that, section 4 will give a short look on related work in the field of metadata management. Finally, a conclusion and a short outlook on future research will be given in section 5.

2 Component-Based Development

In this section, we present a component-based development of database schema. Therefore, we will describe database components as presented in [5] and [4]. After introducing the components, we want to demonstrate how they can be used to dimension a global schema and how the dimensions can be connected together.

2.1 Components in a Nutshell

For our definition of a database component we use the Higher-Order Entity-Relationship Model (HERM) [6]. In HERM a database type is defined as $\mathfrak{S}$ = (Struc, Op, Σ) with a structure Struc, a set of operations Op and a set of static integrity constraints Σ. The structure is defined by a recursive type equality t = B|t $\times \ldots \times$ t |[t]|{t}|l:t over a set of basic data types B, a set of labels L and constructors for tuple (product), set and bag. A database schema S = ($\mathfrak{S}_1, \ldots, \mathfrak{S}_m$, Σ_G) is given by a set of database types $\mathfrak{S}_1, \ldots, \mathfrak{S}_m$ and a set of global integrity constraints Σ_G.

Formally, a component can be described as input-output machines. Every machine gets a set of all database states S^C, a set of input views $I^{\mathfrak{V}}$ and a set of output views $O^{\mathfrak{V}}$. A view can be defined as $\mathfrak{V} = (\mathrm{V}, Op_V)$ with an algebraic expression V on a database schema S and a set of HERM algebra operations Op_V on the view V. The views are used for the collaboration of the components by exchanging data over them. Therefore, an input view of one machine can be connected to an output view of another machine. This data exchange is done by a channel C. The structure of the *channel* is defined by a function $type : C \to \mathfrak{V}$ that maps a channel C on a corresponding view schema V. In general, the input and output sets from a component can be seen as words from a set of words M^* of the underlying database structure.

Thus, a database component is defined as $\mathfrak{K} = (S_{\mathfrak{K}}, I^{\mathfrak{V}}_{\mathfrak{K}}, O^{\mathfrak{V}}_{\mathfrak{K}}, S^C_{\mathfrak{K}}, \Delta_{\mathfrak{K}})$ with a database schema $S_{\mathfrak{K}}$ that describes the database schema of $\mathfrak{K}$, a syntactic interface composed of a set of input views $I^{\mathfrak{V}}_{\mathfrak{K}}$ and output views $O^{\mathfrak{V}}_{\mathfrak{K}}$, a set of all database states $S^C_{\mathfrak{K}}$ and a channel function $\Delta_{\mathfrak{K}} : (S^C_{\mathfrak{K}} \times (O^{\mathfrak{V}}_{\mathfrak{K}} \to M^*)) \to \mathfrak{P}(S^C_{\mathfrak{K}} \times (I^{\mathfrak{V}}_{\mathfrak{K}} \to M^*))$ that connects an output view with a set of input views. To connect two components together they must be free of name conflicts and the input and output views have to be domain-compatible. Assume two components $\mathfrak{K}_1 = (S_1, I^{\mathfrak{V}}_1, O^{\mathfrak{V}}_1, S^C_1, \Delta_1)$ and $\mathfrak{K}_2 = (S_2, I^{\mathfrak{V}}_2, O^{\mathfrak{V}}_2, S^C_2, \Delta_2)$. They are free of name conflicts if the names of their entity, relationship and attribute names within their schema S_1 and S_2 are disjoint. Two channels C_1 from $\mathfrak{K}_1$ and C_2 from $\mathfrak{K}_2$ are domain-compatible, if $dom(type(C_1)) = dom(type(C_2))$. So the output $O^V_1 \in O^{\mathfrak{V}}_1$ of component $\mathfrak{K}_1$ is domain-compatible to input $I^V_2 \in I^{\mathfrak{V}}_2$ of component $\mathfrak{K}_2$ when $dom(type(O^V_1)) \subseteq dom(type(I^V_2))$. For the definition of unification, permutation and renaming of channels together with the introduction of fictitious channels and the parallel composition of channels with feedback we refer back to [4].

2.2 Dimensioning of Data

The modularisation of a database schema with components is then used to scale the schema into dimensions. To do this, the assumption is made that the whole schema is transformed into a component-based schema. After this transformation the data are partitioned into three orthogonal dimensions for application, workflow and metadata, as done in [7]. The application data tier contains all the data that are used by the application. For example, measured data from a research cruise could be seen as application data. The workflow tier contains all data about the structure and process of the workflows that exist within the application. The procedure steps of taking soil samples from the ocean can be such workflow data. The metadata tier contains all metadata that exist within the system. Longitude and latitude of a soil sample can be stored in this tier. To connect the dimensions, we will use a concept, we called harness because the behaviour is similar to wired harnesses in electrical engineering. A detailed description of harnesses is given in section 2.3. Figure 1 shows how the dimensioning can look like.

With the dimensioning of a schema, it is possible to store data based on their origin and purpose. This reduces drastically the complexity of the whole

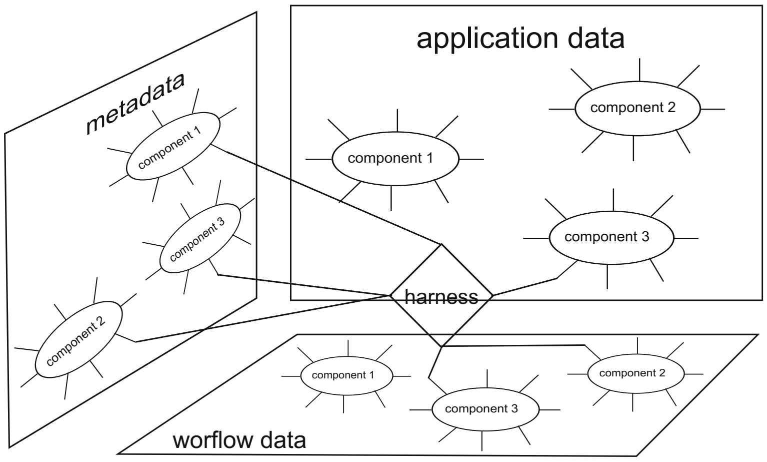


Fig. 1. Data Dimensions

database schema. Because of the dimensioning the data there exist only small components and not a huge global schema. Problems such as saving the same data on different parts of the schema are avoided. Every component can be connected over harnesses to all schema parts where it is needed.

2.3 Harnesses

In this section, we want to present the concept of harnesses. They are based on the work of [4]. A harness is based on a harness skeleton. This is a special form of metaschema architecture. The skeleton consists of a set of components and a set of harnesses that represent the overlapping functions for the components. A n-ary harness skeleton can be defined as a triple $\mathfrak{H} = (\mathcal{K}, \mathcal{L}, \tau)$ with a set of components $\mathcal{K} = \{\mathfrak{K}_1, \ldots, \mathfrak{K}_m\}$, a set of lables $\mathcal{L} = \{L_1, \ldots, L_n\}$ having $n \geq m$ that represent roles of components in the skeleton and a total function $\tau : \mathcal{L} \to \mathfrak{K}$ that assigns a component to its roles. Figure 2 displays an example of a graphical representation of a harness skeleton.

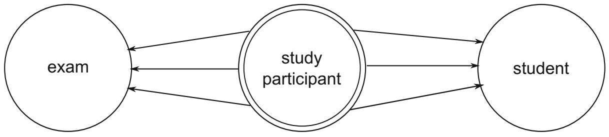


Fig. 2. Graphical harness skeleton

A student and an exam are components. They are represented as single edge circles. The double edge circle represents a harness skeleton that connects the student and the exam to a study participant. Components can be connected to other components in a great variety. In the classical schema design this will lead to a huge and confusing schema because for every introduced subschema a new usage type must be introduced too. To avoid this problem for the component-based design, a filter can be defined for the skeleton. A filter connects the views of the components with the labels of the harness skeleton. Let $\mathfrak{H} = (\mathcal{K}, \mathcal{L}, \tau)$ be a n-ary harness skeleton having m components $\mathfrak{K}_j = (S_j,\ I_j^{\mathfrak{V}}, O_j^{\mathfrak{V}}, S_j^C, \Delta_j)$ with $1 \leq j \leq m$. Furthermore, let $V_1^{\mathfrak{K}_j}, \ldots V_{l_{\mathfrak{K}_j}}^{\mathfrak{K}_j}$ be all input and output views of a component $\mathfrak{K}_j$. A filter $\mathcal{F} = (\mathcal{L}, \iota)$ connects a view from the component $\mathfrak{K}_j$ with a label L_i, if $\iota(L_i) = l$ for $j = \tau(L_i)$ and $l \in \{V_1^{\mathfrak{K}_j}, \ldots V_{l_{\mathfrak{K}_j}}^{\mathfrak{K}_j}\}$ holds. A harness is then defined as $\mathcal{H} = (\mathcal{K}, \mathcal{L}, \iota \circ \tau)$ composed of a harness skeleton $(\mathcal{K}, \mathcal{L}, \tau)$ and a filter $(\mathcal{L}, \iota)$. With this harness we are able to connect components in different dimensions of a dimensioned schema. After the formal construction of components and harnesses for dimensioning data, we want to use these constructs to create our approach of metadata management based on components.

3 Metadata Management

Metadata management covers all functions a system must provide for creation, maintenance and deployment of metadata [2]. Such a management system is normally represented by an independent repository system that is separate from the other system. In this section, we will present a metadata repository that covers all the provided functions of a management system without separating it. Beforehand, we present our approach of such a repository, and we take a quick glance at what kind of data is defined as metadata.

Generally, metadata are defined as *data about data* [1],[8],[3],[2]. In this paper, we want to use a more specialized definition from [1]. Therefore, we first introduce instance data. *Instance data* cover all data that are used as input into a tool, an application, a database or other process engines. All data that describe the format and characteristic of instance data are called *metadata*.

One problem of separating data into instance data and metadata is the fact that the separation depends on the user's point of view on the data. Take as example a relation person = ($\underline{\text{EMail}}$, familyName, firstName, address, fon). If a system developer uses this relation the data within the rows are the instance data he wants to work with. The caption of the columns is the metadata for him because they describe what instance data the developer can find in the columns of a row. If an administrator of the database looks on the same relation, the caption of the columns is the instance data he works with. The data within the rows are not of interest for him. Information about the domain of the columns and the disc space usage are then the metadata for his instance data. As a consequence, instance data and metadata can not be directly separated. They are always in relation to the user's point of view and the context of the user. Thus, in the next step we have to structure metadata in a global context to avoid the user's point of view on the data [2],[1].

3.1 Metadata Categories

In this section, we categorize metadata in a global context to describe the different issues they are used for. For this categorization, we are only interested in functional metadata [9], [2]. These are metadata that are used to interpret application data and to recognize correlations between the data. Technical metadata are not dealt with in this paper as they describe the structure and the behaviour of application data. In general, technical metadata can be found in all modern systems as, for example, the data-dictionary of a database management system.

To divide the functional metadata into categories, we use the fact that metadata are used to describe instance data. This can be represented by the classical W6H-questions: *who*, *what*, *when*, *where*, *why*, *how* and *by what means*. These questions are first mentioned in the classical rhetoric of Hermagoras of Temnos[1] and can be found in the *Zachman Enterprise Architecture Framework* for Information Systems [10]. Therefore, we will use only a single metadata object to answer one of these questions. In the next step, we generate categories using an extended W6H-question set. Every category covers the questions that belong together and that are disjoint to the other questions. After this, we get a set of metadata categories that are shown in Figure 3.

As a result, we get six disjoint categories that cover a set of metadata. Because of answering exactly one question, every metadata can be classified exactly into one of these categories. *Time and Space* covers all relevant metadata about the time and the space for the application data. The *Quality* component contains the metadata for quality information. *Service* metadata gives information about the reason why data is in the system and the method how the data comes into the system. The *Administrative* category covers the information about all administrative information such as rights on the data or roles of the user. The *Structure* consists of metadata that describe the structure of the data it belongs to. *Provenance* covers the history of data from the creation to the deletion of data in the system.

Thus, we have found a way to divide metadata into six global categories that are independent from the user's point of view. Every functional metadata in a system could be assigned to one of these categories. Therefore, a model is needed that can map these categories on an application view. This will build a metamodel as defined in [1] for our categories. For this, we use the component-based development of database schemes from section 2.1. Consequently, every metadata category becomes a single component in the metadata dimension. The advantage of taking the component based approach is that we get all advantages from this approach for every metadata category, for example, the easy extension of a schema and the good understandable schema of the component. The last part missing from our approach is the connection between the metadata components and the corresponding application data components.

[1] Unfortunately, the work of Hermagoras of Temnos is lost. However, the Roman author Cicero refers back to the work of Temnos in his opus "'*De Inventione*"'. Therefore, parts of Temnos' work are still available to the present day.

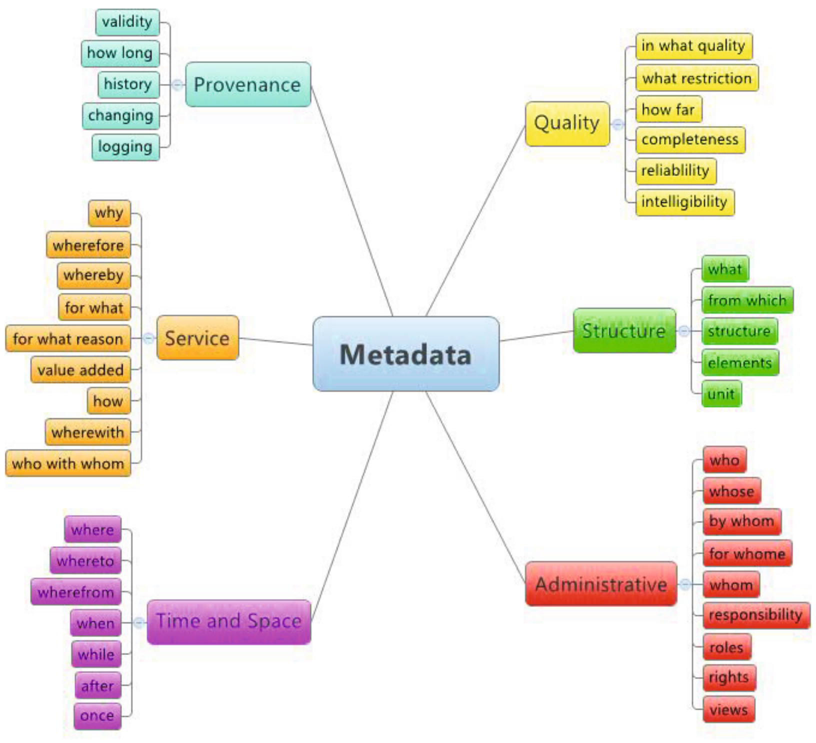


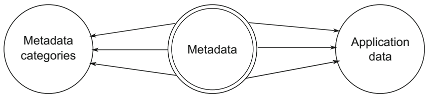
Fig. 3. Metadata categories

3.2 Metadata Data-Dictionary as Repository

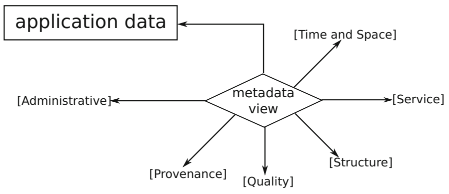
The last step for our metadata management approach is the connection between the metadata components to the components in the other dimensions. In this section, we will only describe the connection of metadata with application data. The connection to the workflow data can be realized in the same way. In 2.3 we have described the concept of harnesses to connect components in different dimensions. We will use these harnesses to connect the metadata categories with the application data. These harnesses are then realized as a metadata data-dictionary that represents an internal metadata repository for a system.

Data-Dictionaries can be found primarily in the area of database management systems (DBMS). They represent a set of system tables that covers different information like the definition of schema objects, logging from actions, comments for tables and rows and much more. For example, take the SYSIBM table from the DBMS DB2. They cover information about the structure of tables, rows, triggers and functions within the DBMS. So a Data-Dictionary contains all needed technical metadata for a DBMS. The great advantage of such a dictionary is the separation of the metadata from the application data of the DBMS [11].

Thereby, the structure of the dictionary is different in most DBMS. For example, the system tables in Oracle can not be read by users of the system. A user only gets access over defined views of the system tables. There is no general schema of a Data-Dictionary. The advantage of covering metadata and application data over relations or views within a DBMS is the possibility, to access the connected data over a query language like SQL. To avoid inconsistency on the data, all tables in the data-dictionary are set to read-only for the user. Changing data in the dictionary is only implicitly possible by using a system query like *CREATE TABLE* or *ALTER TABLE* that changes the data in the dictionary.



(a) Harness skeleton for metadata



(b) Metadata view pattern

Fig. 4. Harness realization

For our approach to use harnesses to connect metadata with application data, we will use similar techniques as used in the design of a Data-Dictionary in a DBMS like DB2 [11]. As described in 3.1, our six metadata categories are represented by components. A harness can be used to connect these metadata components with the application data dimension. Figure 4(a) shows a general harness skeleton that represents this connection. We now map this skeleton on a HERM model that represents it. To find such a model, it is important that every metadata apply only to one category. Therefore, every application data can only have zero till six connections to the metadata components. The number of connections depends on the existing metadata for the application data. Thus, a harness can be mapped to a view that covers all relevant metadata for the description of an application date. The harness will not be used for inserting data directly into the system. Consequently, the harness can be connected to the output views of the components and builds a read-only connection. Metadata modifications are only implicitly possible by using the input views of the corresponding component. Figure 4(b) shows a pattern of a generic realization of a harness into a HERM model that connects application data with the corresponding components from the metadata dimension. Every connection to a metadata category is optional for the relationship type. It is important that such a relationship type is only compiled, if at least one connection to a metadata category exists. Otherwise, such a relationship is dispensable and must not be compiled. So a harness can be mapped on a n-ary relationship type with $n \in \{2, 3, 4, 5, 6, 7\}$ connections. This relationship type represents a metadata view. Such a view has a similar structure, functionality and content as a view in a data-dictionary of a DBMS.

Therefore the quantity of all metadata views in a system builds a metadata data-dictionary. This dictionary covers all metadata that exist for the application data within the system.

Finally, we want to show that our metadata Data-Dictionary represents a metadata repository. In [1], a *metadata repository* is defined as an integrated, virtual area to charge with metadata. Input, access and structure are independent from a special vendor. The repository is used to store metadata and can be used as interface to external metadata. A repository has to meet six requirements:

1. The content of the repository can be connected to every other content within or outside the repository.
2. Input, access and structure are independent from a vendor.
3. The metamodel of the repository is easily extended without effects on the functionality.
4. Every user can search and access the metadata directly without getting irrelevant data.
5. There is a versioning of metadata within the repository.
6. Metadata and the metamodel are protected against unauthorized access.

Our approach of a component-based metadata Data-Dictionary meets all of these six requirements and is a valid metadata repository. The metamodel is based on components. The components are independent from a distributor and can be extended easily. Connections to every data inside and outside our repository can be established by harnesses. Accessing and searching within the metadata Data-Dictionary without irrelevant data is given because every harness in the dictionary can be compiled for a special usage with the needed metadata information. The last two items depend on a good implementation of such a data dictionary. Thus, with a good implementation these requirements are also met.

4 Related Work

The categorization of metadata can be found in different scientific works in literature. In [1], there is a categorization into *specific*, *unique* and *common*. It is based on the assumption that different disjoint user groups can be identified within a system. Specific metadata cover all data that is only created and used by the same user group. If a user group utilizes metadata that is created by another single user group, it is unique metadata. Metadata that is created and used through all user groups in the system is called common metadata. This is an user centric approach that requires the user group identification in a system. Furthermore, there exist metadata that could not be immediately categorized. A categorization that is not user centric can be found in [9]. There, we can also find six different categories of functional metadata; namely *Administration*, *Terminology*, *Context driven*, *Governance*, *Structure* and *Operative*. They should support the terminology management for master data management. All this categories are also answering special WH-questions.

Beside the categorization of metadata, there exist a lot of standards that take a close look at the field of metadata management. Two of the important standards are the *Dublin Core* and the *Common Warehouse Metamodel* (CWA). The *Dublin Core* is standardised *International Organization for Standardization* (ISO) under ISO 1583 [12]. It covers 15 metadata elements that can be used to describe a resource. The purpose of using the *Dublin Core* is the easy detection of resources within a system. Examples for *Dublin Core* elements are title, type and format of a resource. The *Common Warehouse Metamodel* was standardised by the OMG in 2001. It is the prime standard for the field of data warehousing [2]. The CWA is a component-based approach and covers five layers; namely *Object-Model*, *Foundation*, *Resource*, *Analysis* and *Management.* Every layer consists of packages, and every package consists of components that cover a set of metadata elements for the package. There exist other standards for metadata for different levels. There are meta-meta architectures like the *Meta Object Facility* (MOF). Also there are languages for describing metadata like the *Resource Description Framework* (RDF) or describing the interchange of metadata like the *XML Metadata Interchange* (XMI). Furthermore, there are standards for special areas like the *Learning Object Metadata* (IEE LOM), or the *ISO 19115* for geographic information.

There exist a great set of papers about metadata models for highly specialized problems[2]. But there are only some works about generic metadata models. Most of these generic metadata models are models for special areas. An example of such a generic framework can be found in [13]. The framework is based on XML and describes a metadata format for multimedia data that covers general informations on the one hand, such as the title or the author of a special content, and on the other hand special media data metadata such as the GPS location and the resolution of an image. Additionally, the general information can be extended by standards like the *Dublin Core*. An approach for a generic metadata model that covers any kind of metadata can be found in [14]. This approach is based on a combination of the generic Modelling Principles and the architecture of Data Vaults.

5 Conclusion and Outlook

There is a need for a metadata management system that allows a dedicated management of evolutionary metadata. The approach outlined in this paper is based on components which are used to create a set of six metadata components. These components allow a modularization of the global schema with all advantages as, for example, loose coupling. The components make it possible to construct a metadata dimension that lies orthogonally to the application data and workflow data in the global schema. The metadata dimension can be connected to the other dimensions by using a construct called harness. Harnesses can be created

[2] Due to the limitation of this paper we go without citations of these works.

for the special needs of a user group within a system. All harnesses within a system build the metadata Data-Dictionary. The metadata Data-Dictionary is a generalized version of a database Data-Dictionary. Thus, we get an internal metadata repository that allows an evolution safe dedicated metadata management system that reduces drastically the complexity of the global schema without the disembodiment of metadata into an external system.

We have only outlined a general approach on how a good metadata management has to be modelled. In a subsequent step, we must create conceptual models for the six metadata categories. Such a detailed metadata schema that combines all potential aspects for metadata management is under development. Moreover, we must create a system that implements our approach of a metadata Data-Dictionary. Furthermore, such a system must cover some other problems that can be found, for example, in the field of scientific data management [15] or master data management [3]. All these fields need interfaces for the import of metadata from heterogeneus systems. Also the export of data into external and heterogeneous data massive must be possible in such a system. Topical privacy of data is another part such a system must regard. Privacy covers problems such as the property and disclosure of data inside and outside of such a metadata management system. For all this, our approach should be a first step towards such a metadata management system.

References

1. Tannenbaum, A.: Metadata Solutions: Using Metamodels, Repositories, XML, and Enterprise Portals to Generate Information on Demand. Addison Wesley, Reading (2001)
2. Marquardt, J.: Metadatendesign zur Integration von Online Analytical Processing in das Wissensmanagement. Kovač (2008)
3. Berson, A., Dubov, L., Dubov, L.: Master Data Management and Customer Data Integration for a Global Enterprise. illustriert edn. McGraw Hill Professional (2007)
4. Thalheim, B.: Component Development and Construction for Database Design. Data & Knowledge Engineering 54, 77–95 (2005)
5. Schewe, K.D., Thalheim, B.: Component-driven engineering of database applications. In: Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling, APCCM 2006, vol. 53, pp. 105–114 (2006)
6. Thalheim, B.: Entity-Relationship Modeling: Foundations of Database Technology. Springer (2000)
7. Noak, R., Thalheim, B.: Architecturing for Conceptual Modelling in the Large. In: Kiyoki, Y., Tokuda, T., Yoshida, N. (eds.) Proceedings of the 23rd European-Japanese Conference on Information Modelling and Knowledge Bases. Information Modeling and Knowledge Bases XXIII, pp. 29–48. IOS Press (2013)
8. Heinrich, L.J., Stelzer, D.: Informationsmanagement: Grundlagen, Aufgaben, Methoden, vol. 10. Oldenbourg Wissensch. Vlg (2011)
9. Scheuch, R., Gansor, T., Ziller, C.: Master Data Management: Strategie, Organisation, Architektur. Dpunkt. Verlag GmbH (2012)
10. Zachman, J.A. (2008), `http://www.zachmaninternational.com/index.php/the-zachman-framework`

11. Denne, N.: DB2: Theorie und Praxis, vol. 7. DGD-Ed. (2001)
12. Gordon, K., Society, B.C.: Principles of Data Management: Facilitating Information Sharing. British Computer Society (2007)
13. Brut, M., Laborie, S., Manzat, A.M., Sedes, F.: A Generic Metadata Framework for the Indexation and the Management of Distributed Multimedia Contents. In: NTMS, pp. 1–5 (2009)
14. Saratchev, P.: Towards a Generic Metadata Modeling. Yearbook of the Faculty of Computer Science 1(1), 161–174 (2012)
15. Neuroth, H., Strathmann, S., Oßwald, A., Scheffel, R., Klump, J., Ludwig, J.: Langzeitarchivierung von Forschungsdaten: Eine Bestandsaufnahme. Hülsbusch, W (2012)