GEMMS: A Generic and Extensible Metadata Management System for Data Lakes

Christoph Quix[1,2], Rihan Hai[2], Ivan Vatov[2]

[1] Fraunhofer-Institute for Applied Information Technology FIT, Germany
[2] Databases and Information Systems, RWTH Aachen University, Germany
christoph.quix@fit.fraunhofer.de
hai@dbis.rwth-aachen.de, ivan.vatov@rwth-aachen.de

Abstract. The heterogeneity of sources in Big Data systems requires new integration approaches which can handle the large volume of the data as well as its variety. Data lakes have been proposed to reduce the upfront integration costs and to provide more flexibility in integrating and analyzing information. In data lakes, data from the sources is copied in its original structure to a repository; only a syntactic integration is done as data is stored in a common semi-structured format. Metadata plays an important role, as the source data is not loaded into an integrated repository with a unified schema; the data has to come with its own metadata. This paper presents GEMMS, a Generic and Extensible Metadata Management System for data lakes which extracts metadata from the sources and manages the structural and semantical information in an extensible metamodel. The system has been developed with a focus on scientific data management in the life sciences which is often only file-based with limited query functionality. The evaluation shows the usefulness in this domain, but also the flexibility and extensibility of our approach which makes GEMMS also applicable to other domains.

1 Introduction

Data integration has become more dynamic with the introduction of data lakes [8,10]. Instead of loading transformed, cleaned, pre-aggregated data to an integrated repository, data is loaded in its original structure into a central repository, and the cleaning and transformation steps are done within the repository [3]. The advantage is the reduction of integration efforts which have to be spent *before* the repository can be used. Also, this model is more suitable if sources are frequently changing or cannot be completely defined in advance.

Scientific applications (e.g., life sciences) are examples in which flexible data management and integration solutions are required [2]. Data from experiments is collected and processed in various files (e.g., CSV, Excel, proprietary file formats) using a broad range of tools for image and data analysis. There are no predefined schemas, standards are rarely used, and the workflow is documented only (if at all) in a lab notebook in an unstructured form. To avoid repeated experiments for the same substances, or to learn from other similar experiments, an integrated

In: S. España, M. Ivanović, M. Savić (eds.): Proceedings of the CAiSE'16 Forum at the 28th International Conference on Advanced Information Systems Engineering, Ljubljana, Slovenia, 13-17.6.2016, published at http://ceur-ws.org

Date	09/2015					
Autor	John Doe					
Label: Label1						
Mode				Measurement from above		
Emission wavelength start				380	nm	
Emission wavelength end				600	nm	
Emissions wavelength step				2	nm	
Scan count				111		
Spectrum (Em)				280...850: 20 nm		
Spectrum (ex) (Sector 1)				230...315: 5 nm		
Spectrum (ex) (Sector 2)				316...850: 10 nm		
	Temperature: 25.5 °C					
WL	380	382	384	386	388	390
E1	966	224	162	171	206	273
E2	477	240	135	168	148	150
E3	627	235	171	174	232	263
E4	280	160	147	214	252	375
E5	657	245	164	167	157	179
E6	159	97	95	101	150	171

Fig. 1. Spreadsheet Example

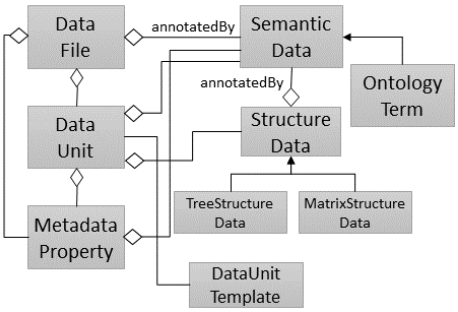

Fig. 2. Conceptual View of the Model

repository would be very beneficial for the scientists as they could explore and analyze the data of previous experiments [9]. Building an integrated repository for a wide range of scientific data is a challenge because of the lack of well-defined schemas and frequently changing requirements. Thus, a metadata repository managing the descriptions of the data sources would be already very helpful.

Fig. 1 shows a typical example for a data file in the life science domain. The spreadsheet has been generated by the control software of some hardware device. It is self-describing and contains metadata about the experiment (author, date, parameter values, etc.) as well as the measured data with its schema (e.g., headers in columns and rows). Other data files may have a different set of metadata properties, various data structures (e.g., table- or tree-like structures instead of two-dimensional matrices), multiple data units (e.g., several sheets within one Excel file), or completely different syntactical formats (e.g., CSV, XML, JSON). Metadata might be encoded inside the file, in the filename, or the directories of the file system. This heterogeneity of managing metadata and data makes it very hard for scientists to search for data efficiently. Usually, keyword queries across the file system are the major method for searching for information. Data management is done in scripts by reading and writing text-based data files.

To support the scientist in her data management activities efficiently and effectively, a system should provide more sophisticated metadata management functionalities which includes especially an interface that allows queries over (at least semi-)structured data and metadata. The metadata should include structural information (i.e., the schema), but also information about the semantics of the metadata and data elements. By using semantic annotations, such ambiguities can be avoided and the semantics of elements can be clearly defined. Especially in the life sciences, ontologies are frequently used to standardize terminologies.

In this paper, we describe the design, implementation and evaluation of a **G**eneric and **E**xtensible **M**etadata **M**anagement **S**ystem (GEMMS) which (i) extracts data and metadata from heterogeneous sources, (ii) stores the metadata in an extensible metamodel, (iii) enables the annotation of the metadata with semantic information, and (iv) provides basic querying support. The system

should be also flexible and extensible, as new types of sources should be easily integrated, which we prove in the evaluation. GEMMS is a major component in the data lake system introduced in [5], which can be used for scientific data in the life science domain, currently being developed in the HUMIT project[3].

The representation of models and mappings in GEMMS is yet less formal than in some model management systems [1], but we plan to include more expressive languages based on our previous work on generic model management [6,7]. Similar with data lakes, another incremental integration approach are *dataspaces* [4], which features human defined mappings in a pay-as-you-go fashion. In contrast to the envisioned dataspace systems, GEMMS focuses on metadata management as one of its core functionalities.

The paper is structured as follows. Sec. 2 introduces our metadata model, while Sec. 3 describes our system and an algorithm for deriving a tree model from a semi-structured data sources. In Sec. 4, we evaluate our system regarding extensibility and performance. Sec. 5 concludes the paper and gives an outlook.

2 Metadata Model

Our motivating example in Fig. 1 illustrated that data sources come with different types of metadata. The descriptive metadata in the header of the file, gives more information about the contents of the source. It is usually just a list of key-value pairs which does not follow a strict model. Values are either simple literals or could have also a complex structure (e.g., value ranges such as '280-850' in the example). We will model this type of metadata as *metadata properties*.

More important for the extraction and integration of data is the structural information of the source, i.e., what is the structure of the raw data contained in the source. In the example from Fig. 1, the raw data is contained in a matrix, but other data structures such as trees, graphs, or simple tables are also possible. Therefore, we must be able to describe the various data structures which might appear in a data source. In the example, we should model the information that the matrix has two dimensions and that $E1, E2, \ldots$ and $380, 382, \ldots$ are values in these dimensions. This description can then later be used for mapping the source data to another data structure or to formulate a query. We model this data as *structure metadata*[4] in our approach.

Metadata properties and structure metadata are important elements to describe a data source, but are only of limited use if we do not understand the names which are used for properties or metadata elements. Furthermore, labels might be understandable for humans, but a system needs to have a more explicit representation of the semantic information. Therefore, our metamodel allows the annotation of metadata with *semantic data*, which link the 'plain' metadata objects to elements from a semantic model (e.g., an ontology term).

Fig. 2 depicts a high-level view over our metadata model. The model elements *Data File* and *Data Unit* have not been discussed so far. In this paper, as we

[3] `http://www.humit.de`

[4] In the following, we refer to this type of metadata as 'structure data' as we do not want to abuse the term metadata.

mainly focus on accommodating files as data sources, we use the *Data File* element as one model component to present the metadata of a file. We are currently working on the extension of our approach to handle also general data sources (e.g., database systems or web services). Thus, in the future, this concept will be generalized to an element *Data Source*.

As the main part of our model, a *Data Unit* represents an independent piece of data, which might carry its own metadata and raw data. A data unit contains most of the relevant metadata information and is flexible enough for other types of data sources as well. The data unit is an abstract entity, containing the structure of the data it contains, plus additional metadata properties. As all other elements, the data unit can be also annotated. Furthermore, for each file type, the scope of a data unit can be defined and which metadata properties are part of a data unit. This information is provided by a *Data Unit Template*. For different file formats, the data unit has of course different semantics. For example, in Excel files, data units represent worksheets; in an XML document, different data units could represent different sub-trees of the whole XML document tree. Two main advantages of applying data units are that they give users flexibility during the data ingestion process, and also provide a level of abstraction above data files. Data files, as well as other data sources in general, can be seen as containers for data units, since the latter are the ones most relevant for the metadata.

With regard to the relationship between *structure data* and *data unit*, the structure data is attached to a data unit, since it carries the raw data, whose schema should be remembered.

As summary, the data model described in this section performs two main tasks: (1) capture the general metadata properties in the form of key-value pairs, as well as structure data to aid in future querying and (2) attach annotations (usually represented as URIs to ontology elements) to metadata elements.

3 System Architecture

We divide the functionalities of GEMMS into three parts: metadata extraction, transformation of the metadata to the metadata model, and metadata storage in a data store. Design and implementation of the system aim at extensibility and flexibility. The high-level design of the system is depicted in Fig. 3. Even each relationship has a 'uses' role, the most highly coupled component is the metadata manager, which orchestrates the whole process. Another self-contained module is the *extractor*, which uses a module for media type file detection and a component parsing files. The components are described in more detailed in the following.

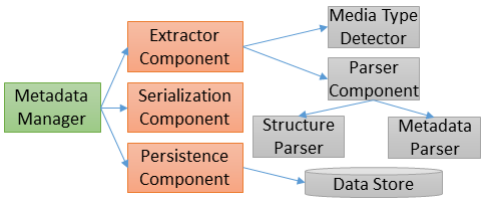

Fig. 3. Overview of the system architecture

The *Metadata Manager* invokes the functions of the other modules and controls the whole ingestion process. It is usually invoked at the arrival of new files, either explicitly by a user using the command-line interface or by a regularly scheduled job. The metadata manager reads its parameters and task from a configuration file and then starts processing the input files.

With the assistance of the *Media Type Detector* and the *Parser Component*, the *Extractor Component* extracts the metadata from files. Given an input file, the *Media Type Detector* detects its format, returns the information to the *Extractor Component*, which instantiates a corresponding *Parser Component*. The media type detector is based to a large degree on Apache Tika[5], a framework for the detection of file types and extraction of metadata and data for a large number of file types. Media type detection will first investigate the file extension, but as this might be too generic (e.g., for XML files), it is possible to refine the detection strategy by specifying byte patterns (which should occur at the beginning of a file) or by providing custom detector classes.

When the type of input file is known, the *Parser Component* can read the inner structure of the file and extract all the needed metadata. Every parser understands the file type and data structure of the file which it is built for, and takes care of specific metadata - either structure data or custom metadata properties. Note that the high expressiveness of some formats, such as XML, implies the existence of multiple parsers for the same data file type, since the *medium* is clear (e.g., XML DOM tree), but the *structure* could be entirely different. The main distinction between extractor and parser components is that the extractor module manages different types of metadata, e.g., structure data or metadata properties, while the parser performs the actual file reading and is specialized in a single type and file structure. The parser uses third-party frameworks working on a lower level than Tika (e.g., Apache POI). The parsers also make use of several algorithms, for example, to detect a matrix structure inside a spreadsheet (as in Fig. 1) or to create an abstract description of a tree structure (i.e., a structure similar to a DTD or an XML Schema).

One important connection point between the data model and system components is the *Data Unit Template*. It is used to define what information should be extracted, and the module using it most actively is the parser. Intuitively, a data unit template gives details about the metadata needed from each data file type (cf. Sec. 2). The parser will use the template to instantiate a corresponding data unit, and then fill this data unit with the metadata extracted from the file. Data unit templates can be more specific than a file type. For example, there is one file type for Excel files, but there can be several data unit templates, each one specifying a different set of metadata properties to be extracted from the spreadsheet (metadata properties in the header of a sheet can be different for each file). For XML documents, the data unit templates contains XPath expressions which specify the location of data units and metadata in the XML file.

The *Persistence Component* accesses the data storage available for GEMMS. The *Serialization Component* performs the transformation between models and storage formats. As serialized objects have to be handled by the storage engine,

[5] `http://tika.apache.org`

Step	SDF	Chembench	File	New
Media Type Registration	7	6	custom-mimetypes.xml	
	1	1	CustomTypes.java	
Data Unit Template	22	0	TabularTxtDataUnitTemplate.java	✓
	0	3	DataUnitTemplateDeserializer.java	
	1	1	DataUnitTemplates.java	
Parser	85	0	SdfPropertyParser.java	✓
Extractor	17	0	SdfExtractor.java	✓
	0	17	ChembenchDescriptorExtractor.java	✓
Mapping to Media Type	3	3	custom-mimetypes.xml	

Table 1. Lines of code needed for each new file type

Media Type	File Count	Extraction Time (s)	Parsing Time (s)
x-2100bioanalyzer+xml	44	10.01	7.46
x-tecan+vnd.openxmlformats..	26	13.03	8.95
x-nanodrop+xml	27	25.83	24.07

Table 2. Performance of Parsers and Extractors

it is closely connected to the persistence component. In our current implementation, we use JSON as serialization format and MongoDB as storage engine.

4 Evaluation

The goal of the evaluation is twofold. On the one hand, we want to show that GEMMS as a framework is actually useful, extensible, and flexible; and that it reduces the effort for metadata management in data lakes. On the other hand, we also evaluated the performance of the metadata extraction components, as it should be possible to apply the system to a large number of files. **Flexibility and Extensibility:** GEMMS has been developed with standard file types (e.g., Excel, CSV, XML) and a few life-science-specific file types in mind. In the evaluation, we analyzed the required steps and efforts for the introduction of new file types. We used the file format SDF[6] which is significantly different from the ones considered earlier; the only commonality is that it is also text-based. The other file type is X-Chembench[7], which is very similar to CSV files. The required steps are shown in the left column of table 1. The 2nd and 3rd column indicate the number of lines of code to implement the required functionality.

The registration of the new media type (file type) is straightforward and just requires a few lines of XML code in the file custom-mimetypes.xml which is used by Tika to recognize file types. The file types have also to be mentioned in one Java class. The definition of the data unit templates requires a little bit more work, as the structure and metadata properties of interest have to be defined. The most expensive part for SDF files is the parser, as these files have

[6] Structured Data Format, `https://en.wikipedia.org/wiki/Chemical_table_file#SDF`

[7] `https://chembench.mml.unc.edu/help-fileformats`

a specific structure that cannot be parsed by one of the existing parsers. For the Chembench file type, the existing CSV parser can be used. This would apply also to XML documents with a custom schema: the existing XML parser could be reused, the data unit template and the extractor just need to provide XPath expressions for the extraction of data units and metadata properties.

After the parser has been defined, the extractor has to be implemented for the new file type. This component integrates the parser with the data unit templates and extracts the required data. Finally, the Tika configuration file needs to be extended with a mapping of the file type to the new extractor class.

Overall, we can see that only very little efforts are required for the framework extension for new data file types. With an increasing number of file types already known by GEMMS, the effort for registering new file types should become smaller, as more code can be reused and the implementation of new parsers is not necessary. By the introduction of new file types, it has further been shown that the designed metadata model is robust enough and had no needs for changes.

Performance Measures: We evaluated the performance of three extractor classes that have been implemented during the development of GEMMS. The files are generated by hardware devices in our life science lab and have proprietary file formats. The tests have been run for the three file types with distinction between metadata properties and structure data for two of the formats. Note that data formats do not correspond directly to the extension of the data file. For example, a single format is based on spreadsheets and the other two are based on XML, but the data in them is structured differently. The first type of XML-based data format encodes its metadata and raw data in a straight-forward tree fashion. XML elements are nested in each other and metadata and raw data values are contained in the leaves. The second type of XML-based data format has a little more peculiar structure. It represents tables, in which the keys of metadata properties are all on the same row, while the values are in the corresponding cells on the row below. Raw data is contained in tables with header columns. Cells are child nodes of the row elements.

For each of the file types, the pair of structure and metadata properties parsers has been run three times in a row in a JUnit test method. The tests have been run on Java SE 1.8.0_45 on a Windows 7 Professional 64-bit Lenovo ThinkPad T440 with 8GB RAM and an Intel Core i5-4210U CPU at 1.7 GHz. The persistence layer of the application is realized with MongoDB 3.0.2. The mean of the three run-time durations is what is shown in Table 2.

As expected, the average durations of the parsing procedures are lower than the extraction procedures. The slower performance is caused by the parsing of the configuration string for the ingestion process and the automatic detection of the extractor suitable for the job.

5 Conclusions and Outlook

In this paper, we proposed the generic and extensible metadata management system GEMMS, designed and implemented as the heart of a data lake [5], which should increase the productivity in analysis and management of heterogeneous

data. Based on a classification of metadata – *structure data*, *metadata properties* and *semantic data* – we derived a generic, extensible and flexible metadata model providing easy accommodation for the new or evolving metadata of various data sources. The framework is also extensible as new types of data sources can be easily integrated as we have shown in the evaluation.

As our focus so far was on extensibility, performance and user interfaces require future work. Although performance is not a major concern in our context, a more scalable processing would be desirable. The querying functionality is yet simple (the user can query for data units annotated with certain ontology terms); an interactive query and exploration interface is one of the next milestones in the HUMIT project. Finally, we also need to consider database systems as data sources; however, we are confident that the required changes or extensions will not break the core system which we have developed so far.

Acknowledgements. This work was supported by the German Federal Ministry of Education and Research (BMBF) under the project HUMIT (`http://humit.de`, FKZ 01IS14007A) and by the Klaus Tschira Stiftung under the miMappa project (`http://dbis.rwth-aachen.de/mi-Mappa/`, project no. 00.263.2015).

References

1. Bernstein, P.A., Halevy, A.Y., Pottinger, R.: A vision for management of complex models. SIGMOD Record 29(4), 55–63 (2000)
2. Boulakia, S.C., Leser, U.: Next generation data integration for life sciences. In: Proc. ICDE. pp. 1366–1369 (2011), `http://dx.doi.org/10.1109/ICDE.2011.5767957`
3. Dayal, U., Castellanos, M., Simitsis, A., Wilkinson, K.: Data integration flows for business intelligence. In: Proc. EDBT. pp. 1–11 (2009), `http://doi.acm.org/10.1145/1516360.1516362`
4. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record 34(4), 27–33 (2005)
5. Hai, R., Geisler, S., Quix, C.: Constance: An intelligent data lake system. In: Proc. SIGMOD. ACM (2016), to appear
6. Kensche, D., Quix, C., Chatti, M.A., Jarke, M.: *GeRoMe*: A generic role based metamodel for model management. Journal on Data Semantics VIII, 82–117 (2007)
7. Kensche, D., Quix, C., Li, X., Li, Y., Jarke, M.: Generic schema mappings for composition and query answering. Data Knowl. Eng. 68(7), 599–621 (2009)
8. Stein, B., Morrison, A.: The enterprise data lake: Better integration and deeper analytics. `http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/assets/pdf/pwc-technology-forecast-data-lakes.pdf` (2014)
9. Stonebraker, M., Bruckner, D., Ilyas, I.F., Beskales, G., Cherniack, M., Zdonik, S.B., Pagan, A., Xu, S.: Data curation at scale: The data tamer system. In: Proc. 6th Conf. on Innovative Data Systems Research (CIDR) (2013), `http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper28.pdf`
10. Terrizzano, I., Schwarz, P.M., Roth, M., Colino, J.E.: Data wrangling: The challenging yourney from the wild to the lake. In: Proc. CIDR (2015), `http://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper2.pdf`