Journal of Data Intelligence, Vol. 3, No. 3 (2022) 366–400

SIMPLIFIED SPECIFICATION OF DATA REQUIREMENTS FOR DEMAND-ACTUATED BIG DATA REFINEMENT

CHRISTOPH STACH
Institute for Parallel and Distributed Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Baden-Württemberg, Germany
christoph.stach@ipvs.uni-stuttgart.de

JULIA BRÄCKER
Institute of Biochemistry and Technical Biochemistry, University of Stuttgart
Allmandring 5B, 70569 Stuttgart, Baden-Württemberg, Germany
julia.braecker@lc.uni-stuttgart.de

REBECCA EICHLER
Institute for Parallel and Distributed Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Baden-Württemberg, Germany
rebecca.eichler@ipvs.uni-stuttgart.de

CORINNA GIEBLER
Institute for Parallel and Distributed Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Baden-Württemberg, Germany
corinna.giebler@ipvs.uni-stuttgart.de

BERNHARD MITSCHANG
Institute for Parallel and Distributed Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Baden-Württemberg, Germany
bernhard.mitschang@ipvs.uni-stuttgart.de

Data have become one of the most valuable resources in modern society. Due to increasing digitalization and the growing prevalence of the Internet of Things, it is possible to capture data on any aspect of today's life. Similar to physical resources, data have to be refined before they can become a profitable asset. However, such data preparation entails completely novel challenges: For instance, data are not consumed when being processed, whereby the volume of available data that needs to be managed increases steadily. Furthermore, the data preparation has to be tailored to the intended use case in order to achieve an optimal outcome. This, however, requires the knowledge of domain experts. Since such experts are typically not IT experts, they need tools that enable them to specify the data requirements of their use cases in a user-friendly manner. The goal of this data preparation is to provide any emerging use case with demand-actuated data.

With this in mind, we designed a tailorable data preparation zone for Data Lakes called BARENTS. It provides a simplified method for domain experts to specify how data must be pre-processed for their use cases, and these data preparation steps are then applied automatically. The data requirements are specified by means of an ontology-based method which is comprehensible to non-IT experts. Data preparation and provisioning are realized resource-efficient by implementing BARENTS as a dedicated zone for Data Lakes. This way, BARENTS is seamlessly embeddable into established Big Data infrastructures.

This article is an extended and revised version of the conference paper "Demand-Driven Data Provisioning in Data Lakes: BARENTS — A Tailorable Data Preparation Zone" by Stach et al. [69]. In comparison to our original conference paper, we take a more detailed look at related work in the paper at hand. The emphasis of this extended and revised version, however, is on strategies to improve the performance of BARENTS and enhance its functionality. To this end, we discuss in-depth implementation details of our prototype and introduce a novel recommender system in BARENTS that assists users in specifying data preparation steps.

Keywords: data pre-processing, data transformation, knowledge modeling, ontology, data management, Data Lakes, zone model, food analysis

1. Introduction

"Data is the new oil." This metaphor, coined by Clive Humby (data science entrepreneur and Chief Data Scientist of Starcount) in the year 2006, is still used today to illustrate the importance of data in a digital society. This analogy also appears to be overly apt. Just as oil was a significant driver for the *Technological Revolution*, data is the key resource of the *Industry 4.0*. Similarities can also be identified with regard to the extraction and usage of data — like oil, data has to be *localized* and *extracted* first. Subsequently, data also needs to be cleansed and processed before it can be stored and provided to users for further exploitation [1].

Peter Sondergaard (Senior Vice President of Gartner) emphasizes the fact that data in general only becomes valuable when it is refined in his 2011 statement that *"information is the oil of the 21st century"*. For instance, datasets may contain inconsistent or inaccurate values, and relevant attributes might not be included in a dataset. Such impurity has to be cleansed in the run-up to an actual analysis. Furthermore, data is typically not available in the form needed for the intended analyses. Besides cleansing, data must therefore also be transformed, pre-processed, and enriched to gain meaningful information from it [43].

Although data shares many characteristics with oil, there are some crucial differences. This becomes particularly evident with regard to the following aspects: While oil is a finite resource, data can be generated at will. Additionally, data is not consumed during processing, i. e., it does not disappear — in terms of storage, concepts for managing *Big Data* are required.

This is further aggravated as data is heterogeneous. Novel tasks emerge frequently, which is why pre-processing needs to be adapted dynamically to the ever-changing challenges. At the same time, refined data is highly specific, i. e., it only reaches its full potential in the use case envisaged through pre-processing. A ready-made one-size-fits-all solution is therefore not an option. It is rather necessary to involve the analyst during data preparation and to adapt it to the respective use case [16].

Moreover, data is not only a driver in industry, but also for, e. g., research and the private sector. A key issue for any data-driven project is rarely a lack of data, but rather the unavailability of the right data in the right form at the right time. John Naisbitt (author in the domain of futures studies) emphasizes that this is a key issue for a digital society with his revised allegory: *"We are drowning in data but starving for information."*

To this end, we introduce *BARENTS*[a], a tailorable data preparation zone for Data Lakes. It enables analysts to model how datasets have to be pre-processed to become meaningful information. This information is assigned to use cases to enable a demand-driven data provisioning. We make the following four contributions:

(a) We introduce an ontology-based method that enables even non-IT experts to **specify the data requirements** of their use cases.
(b) We **manage the data** of the use cases in a resource-efficient manner that addresses the requirements inherent in a Big Data context.
(c) We demonstrate how BARENTS can be **embedded seamlessly** into an established Big Data infrastructure.
(d) We discuss **recommender approaches** that further facilitate the specification of data requirements in BARENTS.

The remainder of this paper is structured as follows: In Section 2, we examine the state of the art with regard to data management. Section 3 identifies requirements towards demand-driven data provisioning. Based on this, we discuss related work in Section 4. Subsequently, we introduce BARENTS and provide details on its implementation in Section 5. We evaluate the performance of our BARENTS prototype in Section 6. Besides a rather basic implementation, we also discuss three realization strategies to improve the performance of BARENTS. Subsequently, in Section 7, we present a feature assessment and discuss enhancements to BARENTS in terms of four recommendation approaches to facilitate the specification of data refinement tasks for non-IT experts. We conclude this paper in Section 8.

2. Data Provisioning Architectures

When it comes to managing and providing large amounts of data in an efficient manner, *Data Warehouses* have long been considered the system architectures of choice. The basic idea is that there is a central relational database, which is considered as the single point of truth and serves as a foundation for any analysis. It has a predefined schema. Thereby business professionals even with little IT know-how are enabled to analyze the contained data with standardized tools (e. g., online analytical processing or reporting tools). The Data Warehouse is populated by external data sources. This entails that data extracted from these sources

[a] Our approach is named after the navigator and explorer Willem Barents — the historic Barents discovered new land in the seas, while our approach explores new insights within a Data Lake.

needs to be transformed first, before it is loaded into the Data Warehouse, in order to match the given schema. In this process, it is inevitable that the extracted raw data is altered and aggregated. This pre-processing and the resulting loss of information limits the analysis potential to a few predefined tasks [32].

Yet, due to the Internet of Things, more flexible solutions without such rigid schemes are needed. Moreover, the inherent transformations cause a significant delay before data is available for analysis. Lastly, the sheer processing of *who*, *what*, *when*, and *where* questions as intended in Data Warehouses is no longer adequate for most use cases [70].

Data Lakes represent such a dynamic system architecture. Similar to Data Warehouses, they provide a central gateway to all data extracted from sources. However, data is stored in an untransformed manner, i. e., in its original format and without a uniform schema. For this, heterogeneous Big Data technologies such as Hadoop are applied. To analyze the raw data, it has to be transformed beforehand, i. e., in addition to the actual analysis, users must also take care of the transformation. This requires extensive IT skills, which is why the target users are data scientists. Yet, they also face the problem of finding all required data in the Data Lake and recognizing the respective data formats on read [24].

(a) Data Warehouse Architecture (b) Data Lake Architecture

Fig. 1. Comparison of the Data Warehouse Architecture and the Data Lake Architecture.

Both architectural models are shown schematically in Figure 1 and their key characteristics are summarized in Table 1. It is evident that demand-driven data provisioning requires a compromise between both approaches combining the simple operability of Data Warehouses with the flexibility of Data Lakes. In the following, we examine the requirements towards such a compromise.

Table 1. Comparison of the Key Characteristics of a Data Warehouse and a Data Lake.

	Data Warehouse	**Data Lake**
Data	• Structured • Aggregated	• Structured & Unstructured • Raw
Storage	• Homogeneous • Schema-on-Write	• Heterogeneous • Schema-on-Read
Usage	• Predefined Tasks • Business Professional	• Support for any Use Case • Data Scientist
	➢ *Simple Data Retrieval*	➢ *Flexible Data Management*

3. Requirement Specification

Based on the work by Grossmann and Rinderle-Ma [23], we derive requirements towards a data management infrastructure, so that demand-driven data provisioning is feasible. We only consider requirements related to data provisioning. In a digital society, these requirements are relevant for any kind of data-intensive task.

R_1 – Adaptable: While in Data Warehouses analyses are known in advance, Data Lakes are supposed to be flexible. Since raw data has to be transformed to be suitable for the intended purpose, i. e., the respective analysis, transformations have to be tailorable. Yet, a simple parameterization of pre-defined transformations is not sufficient; joins between data sources or the application of user-defined functions must be possible as well.

R_2 – Versatile: Similar to the ever-increasing number of use cases, i. e., analyses, new data types and data formats are also being added regularly to a Data Lake. Therefore, an approach towards a demand-driven data provisioning also has to be extensible in terms of new data sources as well as new data types and data formats to reflect this dynamic.

R_3 – Big-Data-Ready: Storing all raw data generates a high data volume. Due to the use-case-dependent transformation required for a demand-driven data provisioning, this data is additionally stored several times in different aggregation states. Therefore, the required disk space should also be restrained to a certain level, e. g., by not keeping all pre-processed data in a materialized manner.

R_4 – Easy-to-Use: Domain experts who have the expertise to specify the requirements for their analyses are usually no IT experts. Data Lakes solve this by requiring data scientists to prepare the data for every analysis. Yet, if no data scientist is available, non-IT experts must be able to tailor the transformations according to their use cases.

R_5 – Resource-Efficient: An approach has to be resource-efficient. On the one hand, the specification of the requirements for the analyses has to be accomplished in a time-saving manner. In particular, the user should not have to reinvent the wheel over and over again for each new use case. On the other hand, the overhead caused by the realization of these specifications, i. e., the actual data transformation and provision, has to be reasonable.

R_6 – Compatible: An approach should be compatible to pre-existing infrastructures, i. e., it has to be embeddable into today's Big Data architectures.

Having identified the requirements towards enabling demand-driven data provisioning, we analyze how related work addresses these challenges in the following.

4. Related Work

Data Lakes have been established as an appropriate data infrastructure for Big Data management [46]. This is largely due to their high flexibility with regard to the provision of data, as outlined in Section 2. On the one hand, this flexibility is evident in the fact that heterogeneous data sources can be integrated to feed the Data Lake with relevant data [45]. On the other hand, Data Lakes can be used as data sources for any kind of use case. Besides raw data, i. e., the unprocessed data from the original data sources, they also provide pre-processed data in different refining states [20]. In addition to basic data cleansing, this data preparation

can even include learning and providing complex machine learning models based on the raw data [68].

While from a conceptual point of view a Data Lake provides the necessary functionality, in practice there are still many issues that need to be resolved for each particular implementation. If not, the valuable data stored in a Data Lake quickly become unusable and thus completely worthless [12]. In particular, with regard to the internal data management, data preparation, and data provision of a Data Lake, many research questions need to be addressed in order to enhance the efficiency and capabilities of a Data Lake [18].

Therefore, it is hardly surprising that a large number of scientific publications can be found in the context of Data Lakes. These papers can essentially be divided into four categories, namely *data partitioning*, *Data Catalogs*, *index structures*, and *navigation assistance* [49]. In addition, there is a recent trend towards the use of *AI-based pre-processing approaches* which prepare raw data for specific use cases [41]. We assess these five concepts in terms of how they facilitate demand-driven data provisioning. Due to the large number of similar approaches, we only discuss representatives for each category.

Data Partitioning. The most intuitive approach to facilitate data discovery is partitioning the data and keeping the amount of data in each partition small. A *Data Puddle* therefore only contains data for a single use case. If the contained data is also transformed, the boundaries to a Data Warehouse become blurred [21]. To support multiple use cases, several Data Puddles can be linked to create a *Data Pond* [29]. Yet, this kind of partitioning causes some problems, e. g., there no longer is a single point of truth and data relevant for multiple use cases must be kept redundant in several Data Puddles. Moreover, similar to a Data Warehouse it must be known in advance which use cases have to be supported. Therefore, *zone models* provide an alternative type of partitioning. Besides a Raw Data Zone, in which the original raw data is held, there are zones which hold transformed versions of the data. Users operate on this pre-processed data. How the data is transformed is defined by the Data Lake [55]. Some models have several zones to provide the data in different processing levels. Users can select the most appropriate zone for their particular use cases [60]. Yet, as the pre-processing is as generic as possible, further transformations are required, which users have to implement and carry out on their own.

Data Catalogs. A *Data Catalog* is a digital inventory of data in the Data Lake, i. e., it contains information about the data source, available attributes, or their data types. Thereby, it facilitates the discovery of data and its usage. While in Data Warehouses a description of all data is possible due to their rigid schemata, this is often not the case in flexible Data Lakes. A thorough and systematic coverage of all data in a Data Lake is still barely addressed in literature [37]. For this purpose, a comprehensive metadata management is required. Sawadogo and Darmont [58] identify two distinct typologies of metadata. *Functional metadata* covers manually generated business metadata, which facilitates the semantic understanding of data, as well as automatically captured operational metadata, which describes the processing of the data, and technical metadata, which contains information about the data formats. For instance, *HANDLE* [13] follows this typology. *Structural metadata* adopts an object-oriented approach by regarding items in Data Lakes as objects with attributes and relations between

them. The *CC Data Lake* [59] takes a structural metadata approach using a knowledge graph. Such semantic knowledge about data sources can also be extracted using a deep learning approach [74], or if there are little to no training data, using a weak supervision approach [38]. However, a Data Catalog only facilitates data discovery, not its demand-driven pre-processing.

Index Structures. While Data Catalogs are designed to improve data discovery by describing the data in more detail, *query-driven discovery* takes the opposite approach. Here, incoming queries are analyzed, and datasets are identified that are similar to the query results. Comprehensive index structures are a key enabler to achieve this. Bogatu et al. [8] leverage hash-based indexes over the features of the data to find similar datasets. By contrast, Zhang and Ives [73] focus on a higher level of granularity. They exploit existing indexes of the source databases of a Data Lake, to recommend additional potentially relevant tables. Yet, similar to Data Catalogs, demand-driven data pre-processing remains the users' responsibility, and they are not supported in this process.

Navigation Assistance. This category summarizes approaches that enable users to browse a Data Lake and comprehend its content. To this end, Nargesian et al. [50] organize the content of a Data Lake using a hierarchical tree structure where the actual data is in the leaves. Starting from a root element, the data is divided into finer and finer categories. By navigating the hierarchy, users find similar data in the respective subtrees. *Data Markets* take this concept a step further. Like in a store for physical goods, data providers offer their data and data consumers can obtain them. This enables market basket analyses and recommendations of useful data based on previous acquisitions or collaborative filtering, i. e., the behavior of comparable users. Fernandez et al. [15] introduce such a Data Market. This requires data providers to be interested in users finding their data and therefore prepare it accordingly. Yet, it can neither be assumed that every data provider invests such effort, nor that an appropriate data offer is available for every demand.

AI-Based Pre-Processing Approaches. The last category of approaches is about using AI techniques to pre-process raw data in a fully automated way in order to make it available for use cases in a tailored form. To this end, Mishra et al. [47] study various ensemble approaches, in which several predefined pre-processing techniques are applied to raw data. All of these approaches have in common that arbitrary combinations of pre-processing techniques are applied to the data in a fully automated manner. For each of the resulting datasets, a machine learning model is computed for the respective use case. The quality of these models (e. g., accuracy or precision) is evaluated. All models with a quality better than a given baseline (e. g., a model trained on the raw data) are pooled into an ensemble. When deployed, all models in the ensemble are evaluated and the weighted average of the results of each model is considered as the overall result. However, the data pre-processing considered here is limited to standard data cleansing tasks, such as handling missing data or removing noise and outliers. Use-case-specific transformations of the raw data are not considered. Chaari Fourati and Ayed [10] present an approach based on federated learning that enables the transformation of data in near real-time. Here, the pre-processing steps are applied as close to the data source as possible. Yet, in order to enable automatic distribution of the transformation tasks, a

model of the infrastructure and the data pre-processing workflow is required. This model can be specified, for instance, using an extended version of BPMN [31]. As the key focus of this approach is on the distribution of the transformation tasks, primarily infrastructure knowledge and less knowledge about the data itself (which is the core competence of the domain experts) is included in the model. Also, information is lost in this approach as sources only submit pre-processed data instead of raw data.

All AI-based approaches face the inherent problem that, apart from use-case-independent standard tasks such as data cleansing tasks, predefined pre-processing steps are required. Consequently, use-case-specific domain knowledge has to be integrated in order to improve automated pre-processing (or to enable it in the first place) [36]. Moreover, since the trained models are usually difficult to understand, subsequent adaptations by the domain expert are hardly possible [39]. Due to the use-case-specific pre-processing tasks, the models also cannot be applied to new use cases without further ado. Therefore, domain experts have to be involved in the training of new models for each and every use case, which is contrary to the fully automated idea behind such AI-based approaches.

Table 2. Mapping of the Investigated Approaches to the Tasks of a Data Discovery Process.

Processing Step	**Discussed in**	**Provided Benefit**
Finding Relevant Data	Eichler et al. [13] Gorelik [21] Inmon [29] Ravat and Zhao [55] Sharma [60]	• Data Partitioning • Metadata
Finding Related Data	Fernandez et al. [15] Langenecker et al. [38] Nargesian et al. [50] Zhang and Ives [73] Zhang et al. [74]	• Index Structures • Relatedness Graphs
Selection of Pre-Processing Steps	Jensen et al. [30] Mishra et al. [47]	• Processing Catalog • Processing History
Application of Pre-Processing Steps	Chaari Fourati and Ayed [10] Kallel et al. [31]	• Abstraction of Processing Steps • Automation via Machine Learning

In addition, all related work requires some kind of *analysis-driven discovery*, providing the essential data for the respective analyses [49]. Without an efficient data provisioning, however, a Data Lake deteriorates substantially [42]. In a highly abstracted view, such a data discovery process consists of two main tasks, namely the retrieval of relevant data and the specification and application of data preparation steps. These two main tasks are further broken down into two subtasks. Regarding data retrieval, it is often not sufficient to merely obtain the main target data, but it usually improves the analysis results considerably if related data can also be identified and used for data preparation. Considering the actual data preparation step, suitable pre-processing operators have to be identified first. To this end, not only domain knowledge is required (e. g., regarding suitable transformations), but also data knowledge (e. g.,

regarding necessary data cleansing tasks). Once these operators have been selected, they have to be applied to the data, which also involves the implementation of the underlying algorithms.

Table 2 lists main representatives of the related work investigated in this paper and maps them to the four tasks of the data discovery process. The table also indicates the key support provided to the user in each step. Here, the disparity in the distribution of research activities is apparent. Most research deals with the first two tasks, i. e., the identification and discovery of relevant and related data. Particularly in the selection and specification of required data pre-processing steps and in the application of these steps, IT-inexperienced users still need more assistance.

Besides the discussed main categories of research approaches, there are many hybrid approaches that combine these concepts. Also, combinations of dynamic Data Lakes with user-friendly Data Warehouses are possible [40]. For instance, Jensen et al. [30] present a programming framework that enables to easily specify extract-transform-load flows to transform data from heterogeneous sources into a processable format. Here, users only need to implement the three individual steps in Python, while the framework handles the execution of the workflow. Thereby, the flows are parallelized as much as possible, which renders the approach suitable for processing Big Data [71]. As shown in Section 2, the extract-transform-load workflow is applied in both, Data Warehouses and Data Lakes. Although the approach by Jensen et al. [30] greatly simplifies the realization of the workflows as a whole, a lot of IT know-how is still required to implement the individual steps.

When considering these five main research directions, the first three categories — namely data partitioning, Data Catalogs, and index structures — are geared solely towards the internal management of data in order to enable a more efficient data retrieval and accessibility from a technical point of view. Only the approaches to assist navigation focuses on human users by attempting to give them a more straightforward gateway to the data sources. While all of this is valuable for data preparation, supporting the actual data pre-processing is only addressed by the AI-based pre-processing approaches.

While the latter sounds very appealing, as an AI might discover new insights regarding data pre-processing and additionally no expensive and rare domain experts are required [28], studies show that full automation is not effective and a trade-off between both trends has to be achieved [5]. That is, comprehensive tool assistance of domain experts in data pre-processing is needed [4].

To this end, we introduce our approach BARENTS in the following section to solve this problem.

5. A Tailorable Data Preparation Zone

Since related work supports users to identify relevant data within the flood of data available in a Data Lake, with BARENTS we take the next logical step by enabling non-IT experts to pre-process this data in a demand-oriented manner. This is crucial as the human and his or her knowledge should always be first-class citizen in the data pre-processing process [3].

For this, we build upon prior work by Giebler et al. [19], which introduces a holistic zone architecture for Data Lakes. This architecture categorizes a Data Lake in two areas: a use-case-independent area and a use-case-dependent area. At the transition point, data

scientists or other IT experts transform the available raw data into refined information. This is where BARENTS comes into play by tailoring the data to the specific use cases for the user.

In the following, we first describe how BARENTS adapts this architecture in Section 5.1. We then discuss in Section 5.2 how domain experts are able to specify transformations in BARENTS in a comprehensible manner. Finally, Section 5.3 outlines how BARENTS is implemented.

5.1. *BARENTS Zone Architecture*

The architecture by Giebler et al. [19] comprises several zones which hold pre-processed data in addition to the raw data. The pre-processing can be highly use-case-specific. Users operate on the refined data. As a result, data can be provided in a demand-driven manner. Yet, profound IT know-how is needed to implement the required transformations, i. e., the effort to support a novel use case is high.

Therefore, in BARENTS we extend this architecture. The result is shown in Figure 2. Here, we differentiate between zones that contain data persistently (depicted in white) and zones that contain data only transiently (depicted in gray).

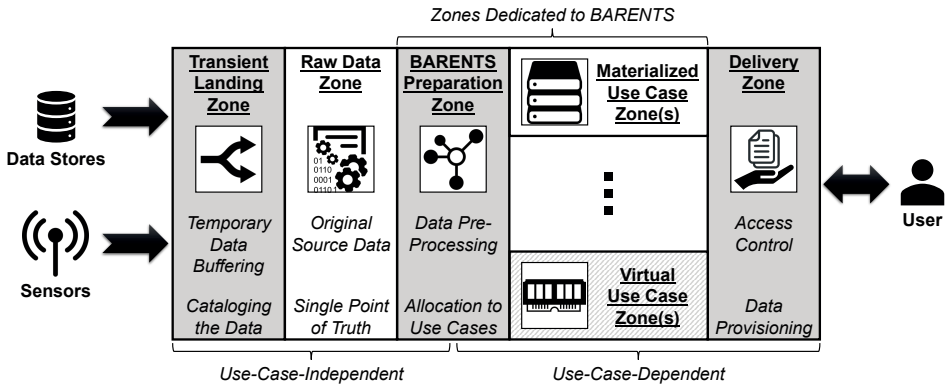


Fig. 2. The BARENTS Zone Architecture.

Data from different source systems enters the Data Lake via the *Transient Landing Zone*. This zone acts as a kind of buffer, since data that is ingested into the Data Lake can accrue in large volumes at a high velocity. If the subsequent zones cannot cope with such high ingestion rates, the Transient Landing Zone can compensate for this by forwarding the incoming data staggered in batches. This zone is supposed to fulfill two additional tasks: On the one hand, it is intended to ensure that the incoming data is authentic, and on the other hand it has to annotate incoming data with metadata that may be relevant for later processing or retrieval. Gritti et al. [22] present a lightweight approach that facilitates both tasks. Here, incoming data packets are signed by the sources using an attribute-based signature. Such a signature guarantees the integrity of the data and provides the necessary metadata via the attributes at the same time.

The unmodified data is then loaded into the *Raw Data Zone*. Since the data here is in an unprocessed state[b], this zone is considered the single point of truth for the Data Lake. It is not intended for users to interact directly with the Raw Data Zone. For this reason, a wide range of storage technologies and systems are used in this zone (e. g., the *Hadoop Distributed File System*[c]), since the main focus is on an efficient data storage and not on an easy data discovery and retrieval.

In order to facilitate data discovery and retrieval, the *BARENTS Preparation Zone* harmonizes the data in the Data Lake. That is, it takes the raw data and pre-processes it based on the demands of the users. As a result, this zone represents the transition from the use-case-independent components of the BARENTS Zone Architecture to its use-case-dependent components. A great deal of insight about the intended use cases — i. e., domain knowledge — is therefore required at this point. Section 5.2 outlines an example of how a domain expert can specify the use-case-specific demands.

The transformed and pre-processed data is stored in one of the *Use Case Zones*. These zones later serve as primary data sources and contain the refined data as demanded by the use cases. The user determines how specifically the data is adapted to a particular use case. S/he is also able to use any technology to store the data in the Use Case Zones. For instance, the *Materialized Use Case Zones* can be implemented as relational databases such as the *Oracle Database*[d] or NoSQL data stores such as the *Apache CouchDB*[e], depending on the requirements of the respective use case.

Since such redundant data storage could be an issue in a Big Data context[f], BARENTS also introduces *Virtual Use Case Zones*. These are, e. g., a strictly in-memory database such as the *Raima Database Manager*[g], a hybrid database such as *Oracle Berkeley DB*[h], or a *Kafka Broker*[i] providing the data via data streams — i. e., the data pre-processed by BARENTS is not persisted but forwarded directly to the user. This enables users to choose the type of data delivery demand-driven for their use cases.

The *Delivery Zone* enables demand-driven data provisioning by referring to the appropriate Use Case Zone. To determine an appropriate Use Case Zone[j], an approach such as the one introduced by Stach et al. [67] can be applied. In this approach, users are identified based on their attributes. Access rights are specified in terms of *public patterns* (i. e., data requirements of an authorized user) and *private patterns* (i. e., insights that must not be disclosed). A utility metric that maximizes the number of detectable public patterns and minimizes the number of disclosed private patterns can be used to select an eligible Use Case Zone.

Orthogonally to these *operational zones*, a Data Lake requires a *Management Zone* that contains, e. g., Data Catalogs, privacy restrictions, or access control policies [60]. Yet, these

[b] Nota bene: "Unprocessed" refers to the actual data — it is, however, enriched by metadata when it is stored in the Raw Data Zone.
[c] see `https://hadoop.apache.org/docs/r1.2.1/hdfs_user_guide.html` (accessed on 20 February 2022)
[d] see `https://www.oracle.com/database/technologies/` (accessed on 20 February 2022)
[e] see `https://couchdb.apache.org/` (accessed on 20 February 2022)
[f] Note that a single dataset in the Raw Data Zone can be included in multiple Use Case Zones.
[g] see `https://raima.com/` (accessed on 20 February 2022)
[h] see `https://www.oracle.com/database/berkeley-db/` (accessed on 20 February 2022)
[i] see `https://kafka.apache.org/` (accessed on 20 February 2022)
[j] In this context, not only the data requirements of the user, but also his or her access rights as well as data privacy requirements have to be taken into account.

aspects are not relevant for this paper and are therefore not shown in the figure for the sake of simplicity.

5.2. *BARENTS Configuration*

To enable domain experts without IT know-how to describe the data requirements of their use cases, we adopt an ontology-based approach in BARENTS. There are many alternative approaches towards the representation of knowledge, e. g., *linguistic knowledge bases*, *expert knowledge bases*, or *cognitive knowledge bases*. However, in our opinion, ontologies are best suited for the configuration of BARENTS, as they are well suited to persist expert knowledge in a machine-processable manner. Since a lot of research work has been done in the area of ontologies, they have reached a high maturity level. Thus, many methods, tools, languages, and systems have been developed in recent years which greatly facilitate the manual creation of ontologies. In addition, machine learning and natural language processing approaches also enable an automatic or semi-automatic creation of ontologies [72]. However, the main benefit of ontologies, which makes them particularly appealing for our purposes, is that they are primarily designed to facilitate the sharing and reuse of knowledge [6]. Thus, the knowledge of domain experts can not only be persisted, but also successively adapted in order to address novel use cases. This constantly reduces the configuration effort for the domain experts since they can leverage the already established knowledge base.

The BARENTS ontology is inspired by the *Data Pyramid* [57], i. e., a model for relationships between *data*, *information*, and *knowledge* [75]. Data is a disjointed collection of facts. It is heterogeneous, unorganized, and not assigned to a particular context. If this data is processed, combined, and assigned to a specific purpose, information is obtained. The preparation steps that are used in this process are commonly validations (i. e., filtering out false or irrelevant data), transformations (i. e., mapping a function to a set of data), and aggregations (i. e., reducing a set of data to a result value). If the obtained information is used to achieve a goal, it is referred to as knowledge. That is, in order to reach the highest level in terms of information value, the information has to be assigned to a specific use case in which it can be profitably used.

Translated to the configuration of BARENTS, at the *data level*, users select the data required for their analyses from the Raw Data Zone and model which data features have to be included in the pre-processing. Data features are modeled as resources in the ontology and data sources are descriptive literals that are linked to these resources via relationships. Additional literals provide further information, e. g., access credentials. The discovery and selection of suitable data sources is supported by the metadata available for the Raw Data Zone.

At the *information level*, the transformations are specified. Initially, all the relevant data features are grouped in a resource. Then, the function which has to be applied to the data, is described in a literal linked to this resource. Additionally, it is modeled which type of transformation is used. BARENTS supports four different types of transformations:

(i) A *filter* operator deals with the elimination or selection of facts from the specified data sets at the data level. For this purpose, a filter function has to be specified that has the following mathematical signature:

$$f : D \to \mathbb{B}$$

This function is applied to a set of data of data type D and maps each of them to a boolean value. All facts selected at the data level where this boolean value evaluates to `false` are filtered out of the set of data.

(ii) A *map* operator deals with the transformation or modification of facts from the specified data sets at the data level. For this purpose, a map function has to be specified that has the following mathematical signature:

$$f : D \to E$$

This function is applied to a set of data of data type D and each fact contained in the set is processed accordingly. In this process, the facts selected at the data level are cast to data type E (which can be identical with data type D).

(iii) A *reduce* operator deals with the aggregation of facts from the specified data sets at the data level. For this purpose, a reduce function has to be specified that has the following mathematical signature:

$$f : E \times D \to E$$

This binary function is applied to a set of data of data type D. All facts contained in the set are combined to a single output value of data type E. Similar to the accumulator in the *von Neumann Architecture*, this binary function takes one fact from the input data set and the current intermediate result — i. e., the data is processed progressively. After processing all facts selected at the data level, the last intermediate result is the result of the reduction.

(iv) Even though many of the pre-processing tasks can be carried out with the three operators presented above, BARENTS also provides support for the use of user-defined functions. For this purpose, the transformation type *procedure* can be used. Although it cannot be assumed that domain experts are able to implement arbitrary transformations independently, implementations for frequently used pre-processing operations can be provided by the IT department in programming libraries that can be integrated and used in BARENTS.

The *knowledge level* is used to model where the results are stored. Here, a literal is used to denote in which Use Case Zone this data sink is located. Similar to the data level, further literals can describe the data sink in more detail, e. g., in terms of access rights.

Sample Use Case Scenario. To demonstrate the configuration of BARENTS, we selected a real-world sample scenario from the field of food chemistry. As food allergies are on the rise in Western countries, it is very important to test food for allergens on a regular basis. In this context, nut seeds are among the most prevalent food allergies which can trigger severe allergic shocks. Hence, in the considered use case, it has to be analyzed, whether a food sample

contains allergens. More precisely, it has to be ascertained whether chocolate samples contain traces of hazelnut or walnut [9].

In the analysis considered in this paper, the samples are examined at the molecular level. That is, the samples are searched for allergenic peptides, i. e., fragments of proteins. Thereby, it is possible to detect even the smallest traces of nuts in the samples. To this end, the samples are analyzed with a *LTQ Orbitrap XL* mass spectrometer[k] paired with an *Accela HPLC* system[l]. The thereby acquired raw data is cross-checked against a protein sequence database[m] to determine whether hazelnut or walnut peptides are contained in the sample. The samples in which a match occurred are marked. To reduce the data volume for subsequent analyses, unmarked samples are then filtered out. The remaining samples, i. e., all chocolates in which traces of nut seeds were detected, are loaded into a peptide analysis software[n] for in-depth analysis [34].

Figure 3 illustrates how a food chemist can specify a configuration for the previously described use case in the BARENTS ontology. However, for the sake of simplicity and clarity, only an excerpt of the complete configuration is shown here. This excerpt contains only the last pre-processing step, in which all irrelevant food samples — i. e., the chocolates that do not contain hazelnut or walnut traces — are filtered out.

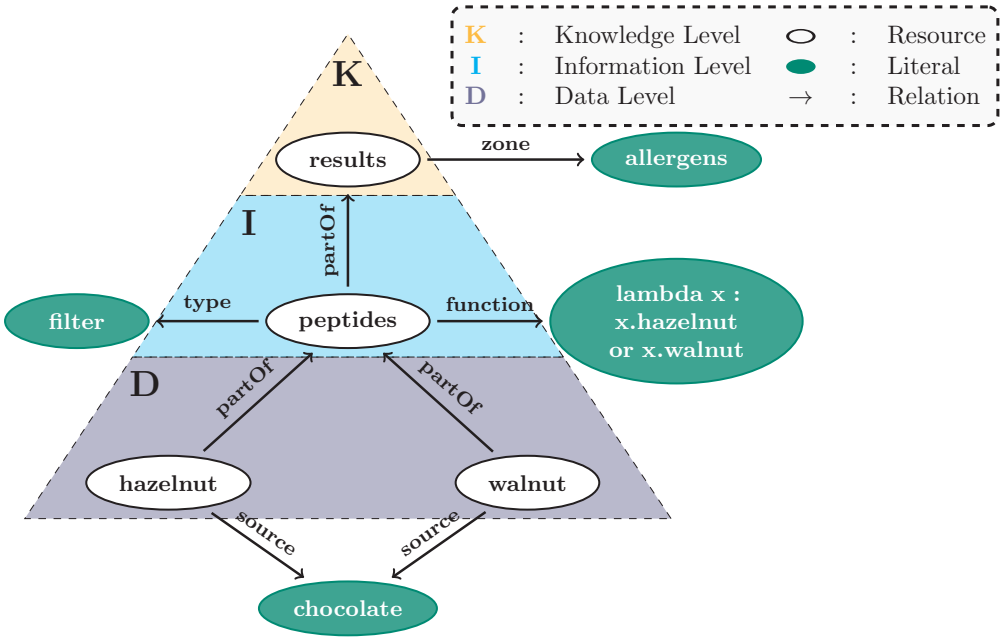


Fig. 3. Excerpt of an Instance of the BARENTS Ontology.

[k] see `https://www.selectscience.net/products/thermo-scientific-ltq-orbitrap-xl?prodID=81295` (accessed on 20 February 2022)
[l] see `https://www.selectscience.net/products/thermo-scientific-accela?prodID=20507` (accessed on 20 February 2022)
[m] see `https://www.uniprot.org/` (accessed on 20 February 2022)
[n] see `https://www.bioinfor.com/peaks-studio/` (accessed on 20 February 2022)

To this end, the domain expert initially selects the database containing the chocolate samples. This is modeled as a literal in the ontology, which is the data source for the specified transformation. From the records contained in it, only the features indicative of hazelnuts or walnuts traces are selected for filtering at the data level. At the information level, a resource is defined that takes these two features as input and applies a filter operator to them. The filter is described as a lambda expression that evaluates to true if one of the two features is present in a sample, i. e., if a corresponding marker is found. Finally, at the knowledge level, it is specified that all samples forwarded by the filter operator have to be inserted into the use case zone "allergens".

Prior to the depicted excerpt, a similarly structured map operator is needed, which assigns a corresponding marker to all samples, for which the comparison with the protein sequence database indicates that they require an in-depth analysis. This also highlights another key feature of our ontology — all transformations can be arbitrarily combined and composed. That is, the outcome of a transformation, i. e., the resources at the knowledge level, can be used as input, i. e., as literals at the data level, in subsequent transformation steps.

Listing 1. RDF/XML Representation of an Instance of the BARENTS Ontology.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dl="http://barents.dl/">
   <rdf:Description rdf:about="http://barents.dl/walnut">
      <dl:layer>Data Layer</dl:layer>
      <dl:source>chocolate</dl:source>
      <dl:partOf rdf:resource="http://barents.dl/peptides"/>
   </rdf:Description>
   <rdf:Description rdf:about="http://barents.dl/hazelnut">
      <dl:layer>Data Layer</dl:layer>
      <dl:source>chocolate</dl:source>
      <dl:partOf rdf:resource="http://barents.dl/peptides"/>
   </rdf:Description>
   <rdf:Description rdf:about="http://barents.dl/peptides">
      <dl:layer>Information Layer</dl:layer>
      <dl:function>lambda x : x.hazelnut or x.walnut</dl:function>
      <dl:type>filter</dl:type>
      <dl:partOf rdf:resource="http://barents.dl/results"/>
   </rdf:Description>
   <rdf:Description rdf:about="http://barents.dl/results">
      <dl:layer>Knowledge Layer</dl:layer>
      <dl:zone>allergens</dl:zone>
   </rdf:Description>
</rdf:RDF>
```

5.3. *BARENTS Implementation*

We use *RDF/XML*[o] for the internal representation of the ontology. RDF is well suited to describe ontology instances in a formal way and for its XML representation, a lot of processing

[o]see `https://www.w3.org/TR/rdf-syntax-grammar/` (accessed on 20 February 2022)

libraries are available. In addition, XML files can be easily shared among users. In Listing 1, the instance of our ontology modeled in Figure 3 is given in this notation.

To enable efficient queries on these RDF/XML files, an equivalent RDF graph has to be generated first. As shown in Figure 4, they are processed in two phases: an initial *preparation phase* and a subsequent *processing phase*.

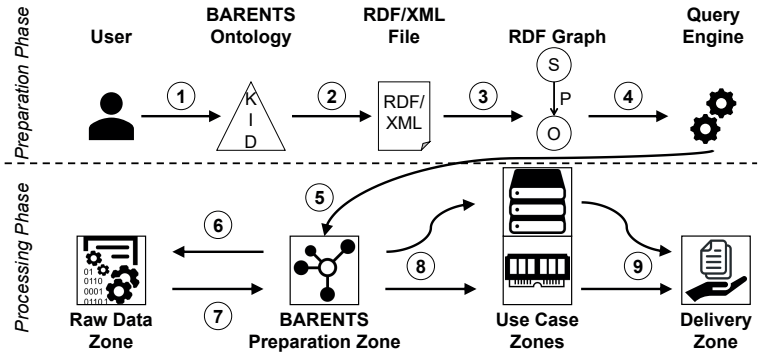


Fig. 4. Two-Phase Configuration and Operation Process of BARENTS.

① First, a domain expert models the pre-processing steps required for his or her use cases, e. g., with the help of a graphical editor. To this end, s/he can make use of existing RDF triples and adopt or extend them. In the example presented in Figure 3, the identified marker peptides could be cross-checked against an external protein sequence database or information emphasizing scripts as presented by Stach et al. [66] could be applied to the data.

② The sum of all triples constitutes the BARENTS ontology. An RDF/XML file is used internally to store it. ③ In order to derive the required pre-processing steps, the XML file is parsed, and the corresponding RDF graph is generated. ④ Efficient query engines are available to traverse such graphs. With them, all sub-graphs can be retrieved which describe a transformation for entities at the data level to entries at the knowledge level.

⑤ After this preparation phase, the derived pre-processing steps are then transferred to the BARENTS Preparation Zone as configuration for the processing phase. ⑥ There, all affected tuples are requested from the Raw Data Zone. ⑦ BARENTS applies the required transformations sequentially to the raw data. Filter, map, and reduce operators are translated into stream operators, while procedures are applied to the datasets using iterators.

⑧ Since the BARENTS Preparation Zone is a transient zone, pre-processed data is forwarded to the respective Use Case Zone specified in the ontology, i. e., either a materialized or a virtual one. ⑨ Users can access the prepared data via the Delivery Zone.

For the data stores at the data and knowledge level, adapters handle the actual access. The information given in the ontology (e. g., access credentials) is used as configurations for these adapters. To support a new data store, BARENTS only needs to be extended by a corresponding adapter. Then, domain experts can use them in their transformation descriptions as sources or sinks without having to worry about technical details.

If needed, a second BARENTS Preparation Zone can be deployed between the Use Case Zones and the Delivery Zone to enable further data pre-processing prior to its release. For

instance, use case data can be filtered or blurred to comply with privacy policies and enable fine-grained data sharing in the process [65].

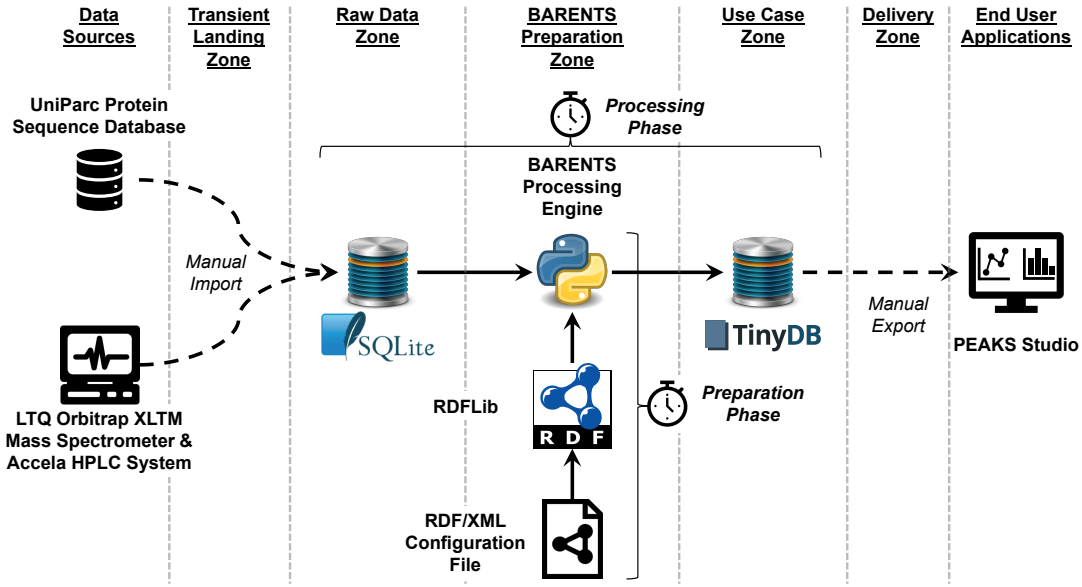


Fig. 5. A Prototypical Implementation of BARENTS used for the Evaluation.

A proof-of-concept implementation for the use case outlined in Section 5.2 is depicted in Figure 5. The databases used in the Raw Data Zone and in the Use Case Zone are only examples that have proven to be suitable for the given use case. Depending on the available infrastructure, other databases and storage technologies can be used as well. One only has to define appropriate connectors for BARENTS that handle the required data accesses. More details on the prototypical implementation of BARENTS as well as an evaluation of our approach are given in the following section.

6. Performance Evaluation

In order to benchmark the performance of BARENTS, we implemented a prototypical test setup. The focus of this prototype is on the BARENTS Preparation Zone since the remaining parts of a Data Lake are not affected by the deployment of BARENTS. Our goal is to analyze the performance overhead of BARENTS in the preparation phase (i. e., reading and parsing the BARENTS configuration and generating the corresponding RDF graph) and in the processing phase (i. e., pre-processing the data from the Raw Data Zone and providing the data in a dedicated Use Case Zone).

For this purpose, we first briefly describe in Section 6.1 the evaluation setup that was used to do this performance evaluation. Then, in Section 6.2, we present the measurement results we obtained for BARENTS and compare them to a baseline that realizes data preparation without BARENTS. Finally, since the prototypical implementation of BARENTS used for this initial performance evaluation very basic, in Section 6.3 we examine three implementation

strategies that can be used to improve the performance of BARENTS. This optimization potential is substantiated by subsequent measurement results. Depending on the intended purpose and available infrastructure, an optimal implementation strategy for BARENTS can be determined based on these findings.

6.1. *Evaluation Setup*

For the performance evaluation of BARENTS, we have deployed a data infrastructure that meets the requirements of the use case described in Section 5.2. This infrastructure is indicated in Figure 5. We manually imported the required data from the involved source systems, i. e., both the UniParc protein sequence database data as well as the LTQ Orbitrap XLTM mass spectrometer and the Accela HPLC system data, into this Raw Data Zone as a preliminary measure. This data import would be handled by the Transient Landing Zone in a real application scenario. For the automatic handling of this so-called *data ingestion*, a variety of off-the-shelf software solutions can be used [27]. For instance, *Apache Sqoop*[p] can be used to transfer bulk data efficiently between a relational database and *Apache Hadoop*[q]. In order to feed streaming data from the metering devices into the Raw Data Zone, e. g., *Apache Flume*[r] can be used [42]. The transfer of the result data from the Use Case Zones to the end user application, i. e., in our use case the analytics software PEAKS Studio, is also handled manually. However, connectors can be provided that allow external applications to automatically access data published in any of the Use Case Zones. To implement such a unifying *serving layers*, the same technologies used for data ingestion can be applied [35]. For more implementation details of these zones, we refer to literature (e. g., the work of Giebler et al. [19]), since they are not relevant for the performance evaluation of BARENTS.

This total abstraction from the actual data sources and data sinks is viable for determining the applicability and performance of BARENTS due to isolation of the underlying zone architecture. Each zone interacts only with its immediate predecessor and successor. That is, from the perspective of the BARENTS Preparation Zone, only the Raw Data Zone and the Use Case Zones exist. All data pre-processing activities — and thus all activities that cause an overhead when using BARENTS — are conducted exclusively between these two zones.

For the implementation of the Raw Data Zone, we opted for a relational database. There are two reasons for this decision: Firstly, all data from the protein sequence database and the two metering devices are structured data, i. e., a schema can be specified for them. Secondly, the use of a relational database is in favor of our baseline, i. e., data preparation without the support of BARENTS, as many of the pre-processing steps can be outsourced to the database, which executes them very efficiently. Also, the amount of processed data can be reduced early, while BARENTS has to process the entire dataset each time. Thus, the overhead determined here represents a worst-case scenario for BARENTS. If a data store without such a comprehensive query interface as provided by relational database systems with SQL is used, the overhead would be considerably smaller. For the sake of simplicity, we use *SQLite DB 3.35.5*[s]. Yet, the actual database does not affect the performance evaluation, since the involved costs

[p] see `https://sqoop.apache.org/` (accessed on 20 February 2022)
[q] see `https://hadoop.apache.org/` (accessed on 20 February 2022)
[r] see `https://flume.apache.org/` (accessed on 20 February 2022)
[s] see `https://www.sqlite.org/` (accessed on 20 February 2022)

apply to both the baseline and BARENTS. Thus, it can be replaced by any other relational database without distorting the results of our evaluation.

When we talk about Big Data infrastructures, we generally assume that the entire Data Lake (including the BARENTS Preparation Zone) is deployed on server systems. In the case of analyses where the results are only relevant for the duration of an experiment, it is also reasonable to run these experiments locally on a desktop computer. As a result, BARENTS is designed not only for high-performance server systems, but also for home computers. Therefore, we deliberately opted to run our performance evaluation on a simple desktop computer. Since the baseline executes the essential pre-processing steps on the relational database, this decision has little impact on the baseline. However, BARENTS would benefit from running on a more powerful system. So, also this decision leads to a worst-case scenario for BARENTS. In accordance with our use case in which the prepared data is only needed temporarily, we opted for a Virtual Use Case Zone. To this end, the document-oriented database *TinyDB 4.5.2*[t] is used. However, it can be substituted with any other data store without affecting the validity of our evaluation. In Table 3 the hardware and software used for the evaluation is given.

Table 3. Setup for our Performance Evaluation.

Hardware		*Software*	
CPU	Intel Core i7-1165G7	**Operating System**	Windows 11
Total Cores	4	**Python Version**	3.10.1
Base Frequency	2.80 GHz	**SQLite Version**	3.35.5
Memory	32 GB DDR4-3200	**TinyDB Version**	4.5.2

Having outlined the evaluation setup, the implementation of the baseline and the BARENTS Preparation Zone as well as the evaluation results are discussed in the following.

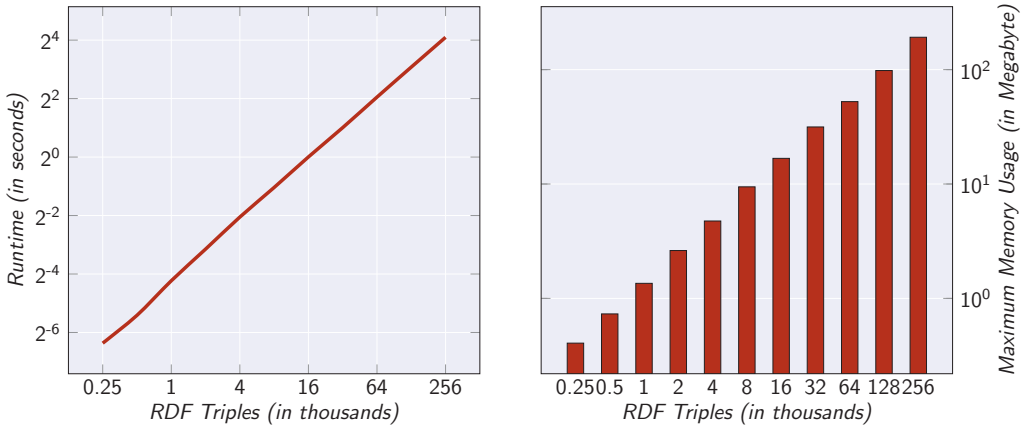
6.2. *Performance Measurement*

BARENTS causes two types of overhead: On the one hand, a configuration must be parsed in the preparation phase to generate an RDF graph. On the other hand, the specified data preparation operations have to be applied to the corresponding data in the processing phase.

Preparation Phase. To determine the overhead, a domain expert modeled the necessary data pre-processing steps as a BARENTS ontology (see Figure 3 and Listing 1, respectively). We then extended this initial configuration with further artificially generated transformation rules. In this way, we incrementally obtained eleven configurations (each time doubling the amount of contained triples) with between 20 and over 21k transformation rules (i. e., between 250 and 256k RDF triples). The RDF/XML configuration file is parsed using *RDFLib 6.1.1*[u] which generates a processable RDF graph. The incurring overhead is exclusive to BARENTS, as no configuration needs to be processed if the data preparation is handled manually. Therefore, there is no baseline against which these costs can be benchmarked.

[t]see `https://github.com/msiemens/tinydb/` (accessed on 20 February 2022)
[u]see `https://rdflib.readthedocs.io/` (accessed on 20 February 2022)

For the performance measurement, each of the eleven RDF/XML files was read ten times and the corresponding RDF graph was built. In each run, we captured not only the runtime (i. e., the time it takes BARENTS to enter the processing phase), but also the memory peak usage (i. e., the required amount of memory to run BARENTS). After each run, we reset everything to exclude distortions due to warm caches. We calculated the median of the ten runs to exclude side effects caused by background processes. The results are shown in Figure 6.



(a) Runtime Overhead

(b) Memory Overhead

Fig. 6. Overhead caused by BARENTS in the Preparation Phase.

It is evident that the overhead caused by BARENTS in the preparation phase — both in terms of runtime as well as memory consumption — is in $\mathcal{O}(n)$. That is, both costs increase linearly with the amount of RDF triples in the RDF/XML file. The total runtime of around 17.1 seconds to parse a configuration of 21k transform rules is fully satisfactory, especially since 21k transform rules can be considered an excessively large configuration. Also, the required memory of approximately 192 MB to process such a configuration is an acceptable overhead even for personal computers, let alone for server systems. In particular, since these are one-time costs, incurring only when a new Use Case Zone has to be created, the overhead caused by BARENTS in the preparation phase can be considered entirely negligible.

Processing Phase. Once the configuration has been loaded, BARENTS can enter the processing phase. In the BARENTS prototype, we use *Pandas 1.3.5*[v] to realize the *BARENTS Processing Engine*. Pandas provides readers for a wide variety of data sources, e. g., databases, spreadsheets, or CSV files. The data imported in this way are internally stored in a *Pandas DataFrame*, a tabular structure with indices on both, rows and columns. Using the `apply` method provided by Pandas, arbitrary functions can be applied to all data in the DataFrame. This way, transformations specified in the BARENTS ontology can be realized in a straightforward manner. The pre-processed data can then be sent to any data sink — i. e., in our case the corresponding Use Case Zone — via writers provided also by Pandas. This Pandas-based implementation of the BARENTS Processing Engine is shown in Figure 7.

[v] see `https://pandas.pydata.org/` (accessed on 20 February 2022)

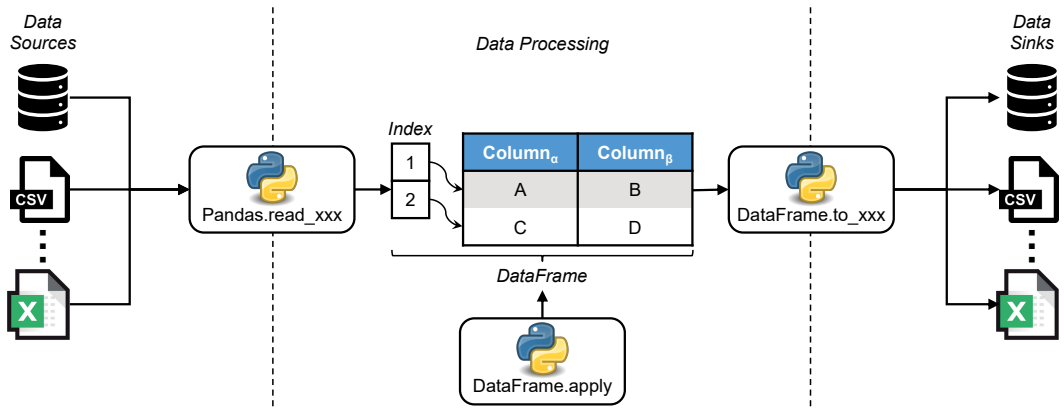


Fig. 7. Prototypical BARENTS Processing Engine Implementation based on Pandas.

For the performance measurement, we populated the Raw Data Zone incrementally with data which have to be pre-processed by BARENTS. Starting with a population of 500 entries, we increased the amount of data tenfold in each of five steps until we obtained 5,000k entries (i. e., approximately 15 GB of data). For the evaluation, we determined how long it takes until all entries in the Raw Data Zone have been processed by BARENTS and are available in the corresponding Use Case Zone. Similar to the preparation phase, we also determined the peak memory usage of BARENTS in the processing phase.

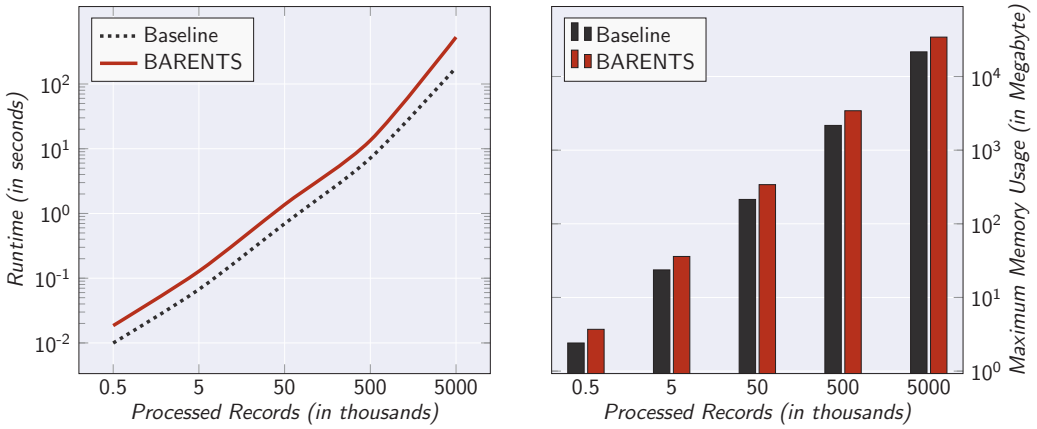
As a baseline, we used a manually created native implementation, similar to the approach a domain expert would have to take in order to be able to pre-process the data without BARENTS. While BARENTS must read and process all data from the Raw Data Zone, in this baseline implementation we use a selection operator to pre-filter the data already in the SQLite DB. This reduces the amount of data to be processed at an early stage. The SQL statement shown in Listing 2 is used for this purpose.

Listing 2. Baseline Implementation used for our Performance Evaluation.

```
SELECT *
FROM chocolate
WHERE hazelnut = 1 OR walnut = 1
```

We also carried out the measurements for the processing phase ten times for each population of the Raw Data Zone. After each run, the SQLite DB and the TinyDB were fully rolled back to exclude distortions due to warm caches. In Figure 8 the medians of each measurement are shown. Again, the median was chosen to exclude side effects caused by background processes.

Compared to the baseline, BARENTS causes an overhead in terms of both, runtime and memory consumption. However, this is to be expected since the baseline implementation performs almost all pre-processing steps in the SQLite DB and thus the amount of data is reduced in advance. Thus, this represents a worst-case scenario for BARENTS. For more complex data preparation tasks, the gap between baseline and BARENTS would be much



(a) Runtime Overhead (b) Memory Overhead

Fig. 8. Overhead caused by BARENTS in the Processing Phase.

smaller — not to mention that IT-inexperienced domain experts would not be able to implement complex transformations by themselves without BARENTS. From this point of view, it is acceptable that the mean runtime of BARENTS is approximately twice as high as the one of the baseline, especially since these costs are one-time costs during data preparation.

Furthermore, it is promising that the overhead of BARENTS also increases linearly in the processing phase. However, it can be seen that the overhead of BARENTS increases significantly from 5000k records onwards. This is due to the implementation, which is based on Pandas. A Pandas DataFrame is permanently kept entirely in main memory, which is why the performance of BARENTS deteriorates for large amounts of data. While this is not much of a problem for server systems, the high memory consumption becomes noticeable on the desktop computer used for evaluation when the data volume reaches about 50% of the available memory.

Although the overall evaluation results look very promising for BARENTS, it has to be noted that Pandas is not designed to process Big Data [51]. Therefore, alternative implementation strategies for the BARENTS Processing Engine are discussed in the following, which are intended to further reduce the caused overhead and thus enable the processing of large amounts of data even on systems with limited resources.

6.3. *Implementation Strategies to Improve the Performance of BARENTS*

In the following, we present three improved implementation strategies for BARENTS, namely *outsourcing the compute-heavy workloads* (see Section 6.3.1), *data partitioning and parallel computations* (see Section 6.3.2), and *deployment of dedicated data adapters* (see Section 6.3.3). For these strategies, we then also determine the corresponding overhead and compare it to the baseline and our basic implementation based on Pandas (see Section 6.3.4). Based on this evaluation, we provide an assessment of which implementation strategy is particularly suitable for which purpose.

6.3.1. *Outsourcing the Compute-Heavy Workloads*

As the comparison with the baseline has shown, the workload in data preparation can be greatly reduced if the data processing operations are outsourced to the underlying data management systems. Such systems can process data much more efficiently due to index structures. Thereby, the data volume can be reduced at an early stage. Our first optimization strategy is therefore to outsource the compute-heavy workloads in BARENTS as well. However, the problem in doing so is that this would require a dedicated processing engine per data source.

To address this problem a transpiler like *Grizzly* [33] can be used. Grizzly maps the Pandas operators to SQL and redirects them to the underlying database system. Grizzly uses lazy evaluation, i. e., the SQL expression is evaluated only when the result set is needed. This way, the SQL expression can be optimized as much as possible, which makes the execution very resource efficient. The processing of Pandas-like commands is outlined in Figure 9.

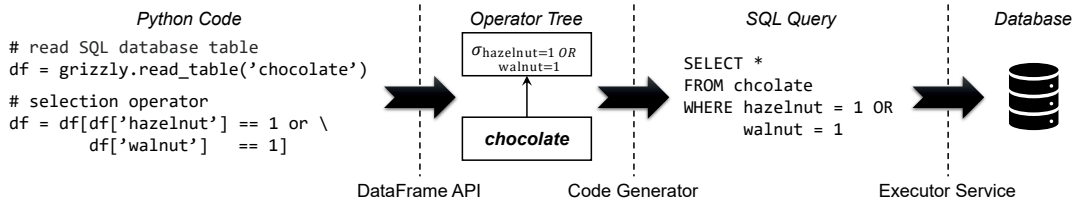


Fig. 9. Processing of Pandas-Like Commands in Grizzly.

Grizzly provides its own DataFrame that can be used similar to a Pandas DataFrame. The DataFrame API translates the Python code into an operator tree representing an equivalent relational algebra expression. This tree is updated and optimized until the result set is needed. Then, a code generator translates the tree into an SQL query that is executed by the database.

Although the Grizzly DataFrame is not fully compatible with Pandas, all operators required by BARENTS are available. For filters and aggregations there are special predefined operators in Grizzly [25]. For all other operations, user-defined functions can be applied [26]. As many Data Lakes use relational database systems with SQL support for data management [58], Grizzly is therefore suitable for the realization of the BARENTS Processing Engine. In the implementation evaluated in Section 6.3.4, we used *Grizzly 0.1.5*[w].

6.3.2. *Data Partitioning and Parallel Computations*

Instead of outsourcing compute-heavy workload, also processing within the BARENTS Preparation Zone can be optimized. The second optimization strategy is therefore to use a parallel computing library for data processing that scales better with Big Data [7]. Parallel processing of compute-heavy tasks has a significant impact on the runtime and memory consumption.

Dask [56] is a library that enables parallel computing for algorithms written in Python by means of dynamic task scheduling. For this purpose, dedicated Big Data data structures are introduced. In the context of BARENTS, the *Dask DataFrame* is particularly relevant. This data structure has an interface similar to the one of a Pandas DataFrame. Yet, all operations

[w]see `https://github.com/dbis-ilm/grizzly/` (accessed on 20 February 2022)

are executed in a parallel and distributed manner. To this end, a single Dask DataFrame consists of multiple Pandas DataFrames. The contained data are partitioned based on their indices, i. e., row-wise. *Modin* [52] is an abstraction layer that handles the distribution of data and tasks to all available CPU cores. Modin can be used for both, computer cluster as well as a single desktop computer. In the back-end, Modin uses, e. g., Dask as a compute engine. In Figure 10 the workflow of Modin is depicted.

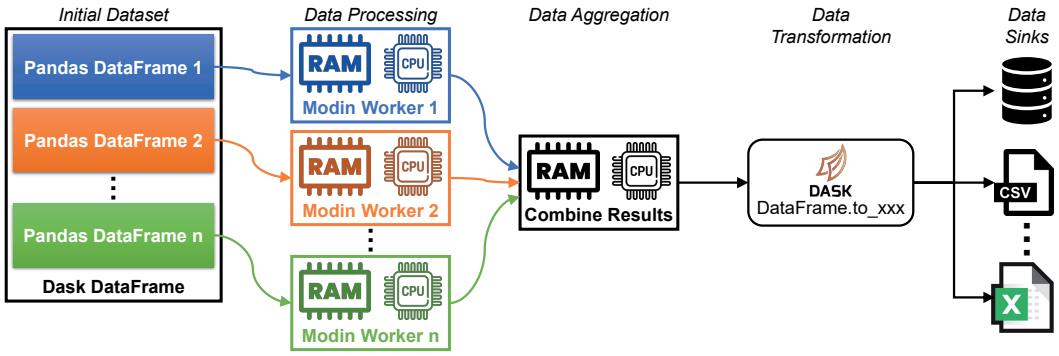


Fig. 10. Data Partitioning and Parallel Data Processing using Modin and Dask.

Modin allocates the DataFrame partitions to *Modin Workers*, which handle the data processing. The number of workers depends on the available CPUs and main memory. Once the processing has been completed, Modin aggregates all partial results into a final result, for which also a worker is required. Just like in Pandas, the outcome can then be sent to arbitrary data sinks via writers. Thus, our initial prototypical implementation of BARENTS can be adapted very easily, since only all Pandas calls have to be redirected to Modin.

In addition to a fully automated distribution of data and tasks, Modin also supports a manual configuration of the workers [53]. This was mandatory for our implementation in the context of this evaluation, since the out-of-the-box configuration resulted in out-of-memory exceptions when processing 5000k records. After a few tweaks, however, it was possible to use four workers, each with two threads, on our test system. In the implementation evaluated in Section 6.3.4, we used *Dask 2021.12.0*[x] and *Modin 0.12.1*[y].

6.3.3. *Deployment of Dedicated Data Adapters*

While all previous implementation approaches use bulk processing to process the data from the Raw Data Zone, this strategy uses stream-like data processing. Thereby, not only high runtime costs for the initial data import can be avoided, but also the memory consumption can be reduced significantly, in particular compared to the approaches where all data in a DataFrame have to be kept in main memory. In a stream-based approach, only a few data objects need to be kept in main memory at any given time, since each object that has been processed is forwarded directly to the data sink. Python supports such a processing mode with the built-in

[x] see `https://docs.dask.org/` (accessed on 20 February 2022)
[y] see `https://modin.readthedocs.io/` (accessed on 20 February 2022)

functions `filter`, `map`, and `reduce` for any iterable data structure. These operators accept lambda expressions to specify how data objects have to be processed. User-defined functions can also be applied to data objects. Thus, all functions that can be specified in the BARENTS ontology are supported. However, while Pandas provides interfaces to any data source and data sink by means of the readers and writers, we need dedicated adapters for this purpose in this implementation strategy. This approach is outlined in Figure 11.

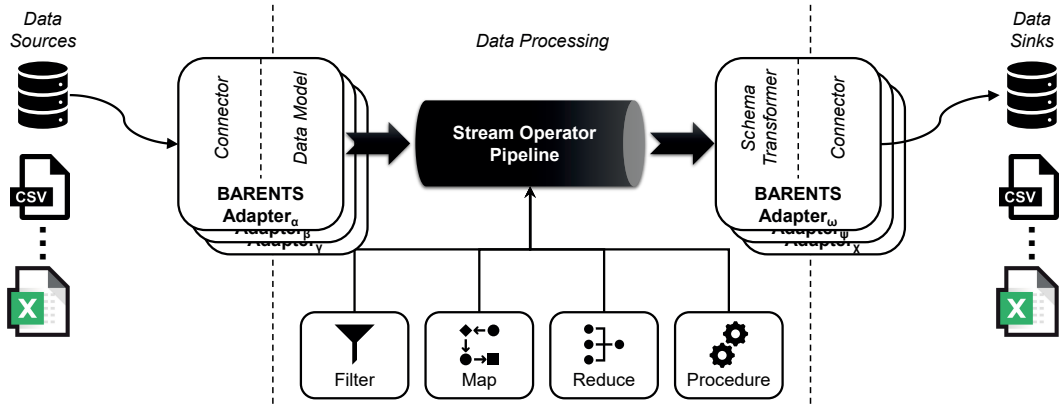


Fig. 11. A Stream-like Implementation of the BARENTS Processing Engine.

The adapters represent a tailored plugin for a data source or a data sink. Thereby, an abstraction is achieved so that the data processing pipeline can operate completely independent of the sources and sinks [62]. In BARENTS, such an adapter consists of two components: a connector that establishes the connection to a given source and a data model that converts the data from the source system into data objects that can be processed by the stream operator pipeline. The adapters for the data sinks have an equivalent design, except that there a schema transformer converts data objects into the data format of the target system. Data models can be easily implemented as *Data Classes* introduced in *Python 3.7*[z]. Listing 3 shows such a data model used for our evaluation.

Listing 3. Data Model for the Use Case Scenario used in the Evaluation.

```
class Chocolate:
    pid: int
    proteins: str
    hazelnut: int
    walnut: int
```

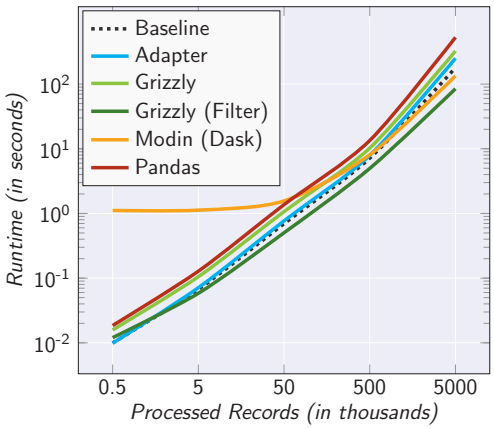
It is striking that only the attributes of the data as well as their data formats have to be specified. Based on this, Data Classes generate various methods, which can be used to convert data objects into other data structures. The information required for the specification of the data model is available in a Data Lake anyway in the form of metadata [14]. Thus,

[z]see `https://docs.python.org/3/library/dataclasses.html` (accessed on 20 February 2022)

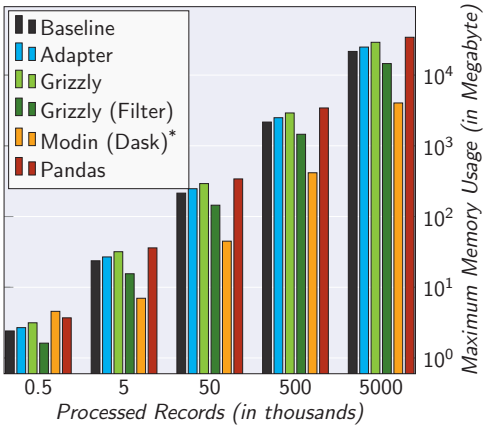
the data models can be generated in a semi-automated manner from these metadata. For efficiency reasons, the adapters can also be used as buffers to bundle several data objects in a micro-batch. Since the optimal batch size depends on the source data, an IT expert should make some tweaks. Since the adapter development can be done detached from the BARENTS ontology specification, this implementation strategy is the only one that can be extended retroactively. Whenever a new data source or data sink is added to the Data Lake, only a corresponding adapter needs to be provided, and the data can be processed instantly [64].

6.3.4. *Evaluation Results*

For these improved implementation strategies, we carried out the performance measurements as described in Section 6.2. In Figure 12, the results are shown in comparison to the original implementation and the baseline. As the preparation phase is not affected by the optimizations and its overhead is negligible anyway, the emphasis here is on the processing phase.



(a) Runtime Overhead caused in Processing Phase



(b) Memory Overhead caused in Processing Phase

		BARENTS				
Records	**Baseline**	*Adapter*	*Grizzly*	*Grizzly (Filter)*	*Modin (Dask)*	*Pandas*
500	0.01 s	0.01 s	0.02 s	0.01 s	1.11 s	0.02 s
5*k*	0.07 s	0.07 s	0.11 s	0.06 s	1.12 s	0.13 s
50*k*	0.70 s	0.77 s	1.07 s	0.50 s	1.56 s	1.37 s
500*k*	7.16 s	7.92 s	10.27 s	4.97 s	8.08 s	13.43 s
5,000*k*	179.344 s	250.23 s	325.25 s	84.87 s	134.17 s	529.21 s

(c) Consolidated Runtime Overhead (Preparation Phase and Processing Phase)

Fig. 12. Compilation of the Evaluation Results for the Discussed BARENTS Implementations.

Overall, all three approaches constitute a significant improvement over the initial prototypical implementation. Each of them reduces both, runtime and memory consumption. However, there are differences between the three strategies:

When using Grizzly (see Section 6.3.1), two cases must be distinguished. If operators are used that are directly supported by the library (in our case the filter), then this approach is the most resource-efficient strategy. Due to the lazy evaluation and the optimizations in the operator tree, even the baseline can be outperformed. However, Grizzly operators cover only a very small part of the required data pre-processing tasks. Therefore, we also implemented our use case as a user-defined function in Grizzly. When using such functions, the performance of Grizzly deteriorates significantly. Nevertheless, this strategy still represents a clear improvement over the initial prototype.

Data processing with Modin using Dask as compute engine (see Section 6.3.2) entails basic costs for the management of the workers as well as the merging of the partial results. Therefore, this approach has a significantly worse runtime than all other approaches when processing very few data. These administrative costs are amortized only for 50k data records and above. For high data volumes, this approach also performs better than the baseline, but not as well as using Grizzly with a filter operator. In terms of memory consumption, this approach seems to be ideal. However, it has to be pointed out that the memory consumption refers to a single worker — i. e., if several workers operate in parallel, the total costs grow significantly.

Table 4. Comparison of the Key Characteristics of the Three Implementation Strategies.

Strategy	**Characteristics**	**Intended Purpose**
Workload Outsourcing	+ highly efficient for inherently supported operators − high costs for user-defined functions − only applicable for SQL-based data sources	The outsourcing of data preparation tasks is particularly efficient for tasks that can be directly mapped to operators of relational algebra. For highly complex and especially user-defined functions, this strategy is less suited.
Parallel Computation	+ most efficient solution for arbitrary data preparation tasks − very high resource requirements	Due to the parallel data processing, this strategy is highly efficient for Big Data. Yet, as a result of the increased resource requirements, this strategy is only suitable for a server-based deployment of BARENTS.
Dedicated Adapters	+ adapters are optimally tailored to the respective source − development effort for the adapters	The adapters represent a good trade-off between processing speed and resource requirements. In addition, the adapters can also provide metadata about the sources, which facilitates the specification of the BARENTS configuration. Yet, the implementation of the adapters causes a development effort. Hence it is not possible to implement adapters for every source.

The adapter-based implementation (see Section 6.3.3) is a good compromise. Even though it clearly lags behind Grizzly with a filter operator and Modin in terms of runtime, it outperforms Grizzly with user-defined functions. In terms of memory consumption, this implementation strategy is the most resource-efficient approach.

Because of the good evaluation results for each of the three approaches, there is a raison d'être for each of the implementation strategies. Table 4 summarizes the advantages and disadvantages of the approaches and gives a recommendation which approach should be adopted for which purpose.

After having evaluated the performance of BARENTS, its functionality is assessed next.

7. Feature Assessment

Lastly, we also have to verify whether BARENTS has all the needed functionality to perform demand-actuated data refinement. To this end, we use the requirements elicited in Section 3. Only if BARENTS satisfies them sufficiently, it represents a significant research contribution.

A data refinement approach must be **adaptable** ($\boldsymbol{R_1}$). Since in a Data Lake environment the required analyses are not known in advance, hard-coded data pre-processing steps are not an option. In BARENTS, all data preparation tasks are specified in the ontology which can be extended or adapted at any time to reflect the data requirements for any analysis. The filter, map, and reduce operators provide comprehensive support for the most common data refinement tasks. In addition, user-defined functions can be implemented by means of the procedure operator to support also complex data transformations.

It is also important that a data refinement approach is **versatile** ($\boldsymbol{R_2}$). That is, it must be possible to deal with arbitrary data sources as well as data sinks. BARENTS supports this due to its zone architecture and the abstraction from sources or sinks. The readers and writers — or adapters, respectively — enable uniform access to any sources and sinks without having to consider any data formats. All data are converted into an internal processing format which is completely independent of the source and target format. Therefore, to support new types of sources and sinks, only appropriate readers, writers, or adapters have to be added.

Moreover, a data refinement approach must be **big-data-ready** ($\boldsymbol{R_3}$), i. e., it must not increase the data load in a Data Lake unnecessarily. The Virtual Use Case Zones ensure that the permanently stored amount of data can be reduced in line with the requirements.

Since a data refinement approach also targets non-IT experts, it has to be **easy-to-use** ($\boldsymbol{R_4}$). While the BARENTS ontology is comprehensible to non-IT experts and enables them to specify transformations without profound IT know-how, the ontology has to be available as an RDF/XML file in order to be processed. In this regard, users need further support. Also, they might reinvent the wheel over and over again, since there is no holistic overview of all the data pre-processing steps already specified in other use cases.

Since during data refinement large amounts of data are processed, an approach also has to be **resource-efficient** ($\boldsymbol{R_5}$). Our evaluation demonstrates that BARENTS is able to process large amounts of data even on a single desktop computer. In particular, the three improved implementation strategies provide resource-efficient data processing options for any purpose.

A data refinement approach must also be **compatible** ($\boldsymbol{R_6}$) with current Big Data infrastructures. Only if an approach can be seamlessly embedded into established infrastructures, it is practically applicable. BARENTS is explicitly geared towards modern Data Lake ar-

chitectures. Furthermore, the abstraction from sources and sinks ensures that BARENTS can also be operated detached from a Data Lake, e. g., locally on a desktop computer. The materialized and virtual storage concepts also enable hybrid batch and stream processing, which is a common data processing principle in Big Data environments [17].

While all requirements towards a demand-actuated big data refinement approach are more than adequately addressed by BARENTS, there is some room for improvement with respect to its usability. In the following, we therefore discuss how usability can be improved.

BARENTS Enhancements to Improve its Usability. There are two aspects in which usability can be improved: First, usability can be improved by providing a **user-friendly editor** for the creation of the RDF/XML files. Second, the specification process itself can be facilitated by providing a **recommender system** to assist users in the work.

Regarding an editor, there is already a plethora of approaches that can not only visualize an ontology in a simple and understandable way, but also allow the collaborative editing of the ontology on a graphical level [2], e. g., *CLONE* [54], *CoModIDE* [61], and *Protégé* [48]. While the specification of the ontology is done graphically, these editors generate processable representations in the background. Since editors such as Protégé can be extended via plugins, it is possible to use them for BARENTS. All that remains to be done is to adapt them so that the editor translates the ontology into a BARENTS-compatible RDF/XML file.

Regarding a recommender system, it is obvious that parts of the BARENTS ontology can be reused for other use cases. The only problem is that users must be made aware of pre-existing transformation specifications. Since the BARENTS ontology is based on an RDF graph, finding relevant patterns that a modeler might be able to reuse in the current context is possible [63]. Also, RDF graphs can be semantically merged to obtain a collective knowledge base [11]. This way, the modeling process not only becomes much more efficient, but the result also becomes better when a user can leverage on other users' expertise. There are four recommender approaches that can be beneficial to BARENTS:

Approach 1: Our initial recommender approach is intended to support the user in choosing an appropriate transformation operator for the respective raw data. It is based on the rule concept of Megdiche et al. [44]. Here, the ontology is not considered at all in making recommendations. Rather, recommendations are based on information about the involved data sources and heuristic rules. For instance, when two different data sources are selected, merging operators are recommended, while filtering, formatting, or aggregation are recommended for a single source. Also, certain types of processing can be excluded depending on the source, such as aggregation when dealing with unstructured data.

For such a recommender approach, only metadata about the sources are required in addition to the applied heuristics. In a Data Lake environment, such information is generally available, e. g., as part of the Data Catalog or the Management Zone, and is therefore available to the BARENTS Preparation Zone. When operating BARENTS locally without a Data Lake, this information can be provided by the data models that need to be defined for any data source in an adapter-based implementation (see Section 6.3.3).

The primary benefit to the user of this approach is to avoid errors, since the recommender can deny the usage of invalid operators and remove unlikely ones from the focus of the modeler, e. g., by positioning them further down in a drop-down menu of an editor.

Approach 2: The second approach also targets at recommending the most suitable data pre-processing steps to the user. While the first approach relies solely on information about the data sources and general heuristics, the second approach makes use of the existing ontology, i. e., domain knowledge. This enables the recommender to point out not only generic transformation operators, but fully specified transformation rules.

For this purpose, the user selects the data sources required for his or her analysis. Then, the recommender traverses the ontology graph and extracts all transformation rules that have already been specified for these sources. For this, only the corresponding *partOf* relations which have the data source as subject need to be gathered. These rules are then presented to the user. This way, s/he is made aware of additional required data pre-processing steps for this data source, and s/he can reuse the ready-made specifications.

By including domain knowledge much better and more specific recommendations can be expected. By pointing to already existing transformations, a user can be made aware if the data s/he is looking for are already available in the required (or sufficiently good) pre-processing stage in a Use Case Zone. A repeated data refinement of these data is therefore superfluous, and an unnecessary overhead can be avoided. Yet, this approach requires a pre-existing BARENTS ontology. Without an ontology for the respective application domain or if a data source is involved that has not been used yet, such a recommender cannot provide any support.

Approach 3: While the first two approaches seek to identify suitable data pre-processing steps for a particular data source, the third approach goes one step further. Here, not isolated data pre-processing steps are recommended, but complex chains of such steps. For data preparation it is necessary to apply a sequence of, e. g., filter, map and reduce operators. In terms of the BARENTS ontology, this means that resources at the knowledge level themselves are also used as subjects of another transformation rule.

In this approach, a user specifies some simple transformation sequences. Based on these sequences, the RDF graph is mined for sub-graphs that contain similar transformations. "Similar" refers in this case to a sub-graph containing the same sequence, but with one additional data pre-processing step. The user can then decide whether this pre-processing step is also relevant for his or her data preparation and can add and adjust it if necessary. Subsequently, the user can recursively search for further recommended pre-processing steps.

Similar to the second approach, the focus here is entirely on leveraging existing domain knowledge. By targeting a specific domain, the recommendations are presumably much more relevant to the users. In addition, this approach is capable of finding complex transformation sequences. The incremental approach that adds one transformation at a time also enables to recommend novel transformation sequences that have not yet been used in this composition. However, this approach can only be applied if a comprehensive ontology with many complex chains of data pre-processing steps is available.

Approach 4: The final approach focuses on the individual experiences and working methods of the domain experts themselves. Similar to the approach of Stach and Steimle [63], a kind of *collaborative filtering* is applied for this purpose. The underlying idea is to learn for different user groups which techniques they prefer to use for data preparation. In other words, clusters of users are identified who have applied similar data pre-processing steps for comparable data refinement tasks. In addition, a personal profile is generated for each user. Based on this profile, a classification can be made to determine the cluster with which the respective

user has the highest similarity. Subsequently, the user only gets recommendations based on transformations that have been specified or applied by other users from this cluster.

It is obvious that this approach operates on a different level of abstraction than the other ones. The aim here is to tailor the recommendations more individually to a user and thus to be able to reduce the number of futile recommendations. For the actual identification of suitable data pre-processing steps in the respective context, underlying approaches such as the three previously discussed are required.

With this approach, a user can be provided with individualized recommendations for transformations that s/he has never specified before. Only the user profile, i. e., the user's preferred approaches regarding data preparation, is taken into account. Yet, it is questionable whether a blending of different data refinement styles would not be more beneficial. Only if a user thinks out of the box, s/he is able to optimize his or her methods. In addition, the problem that a comprehensive ontology is needed is further aggravated by this approach, as only a fraction of an ontology is considered for recommendations and the rest is filtered out.

Preliminary research on these four recommender approaches indicates their theoretical feasibility. In particular, a combination of the second and third approach seems to be suitable for BARENTS, as they are entirely based on the ontology (and thus rely on domain knowledge) and do not require additional data such as user profiles. Yet, the practical applicability of such a recommender has to be proven as part of future work for a comprehensive ontology compiled by several domain experts. Furthermore, it has to be investigated to what extent such a recommender approach can also be applied across domains.

8. Conclusion

Today, data are the most valuable resource. Yet, similar to physical resources, they must first be refined to unleash their full potential — only if data preparation is geared to the intended use case, optimal results can be achieved. This requires a high degree of data and domain knowledge. Since domain experts possess this knowledge but lack the necessary IT skills to implement the respective data refinement tasks, we introduce BARENTS, a tailorable data preparation zone for Data Lakes. BARENTS enables a simplified specification of demand-actuated data refinement tasks. This is achieved by means of four pillar stone:

(a) BARENTS introduces a comprehensible **ontology**. In it, domain experts can collaboratively specify transformations which should be applied to specific source data in the context of a particular use case. By means of the ontology, domain experts can focus on their core skills instead of dealing with the implementation of data refinement tasks.
(b) The **BARENTS Processing Engine** is configured by this ontology. This processing engine is not only capable of parsing the ontology, but also of applying the specified data refinement tasks to data from any source and forwarding the results to any sink. Three different implementation strategies ensure an optimal result for any given use case.
(c) The processing engine is enclosed in the **BARENTS Preparation Zone**, an extension to a reference architecture for Data Lakes. By means of virtual zones, BARENTS also reduces the data overhead in this Big Data infrastructure.
(d) A performance evaluation demonstrates that the overhead caused by BARENTS is reasonable even on desktop computers. A feature assessment reveals that while BARENTS

has all the functionality needed to perform demand-actuated data refinement, there is room for improvement with respect to its usability. To this end, we presented four promising **recommender approaches** that need to be further studied as part of future work.

References

1. W. M. P. van der Aalst (2014), *Data Scientist: The Engineer of the Future*, Proceedings of the 5th International Conference on Interoperability for Enterprise Systems and Applications (I-ESA '14), pp. 13–26.
2. T. Basyuk and A. Vasyliuk (2021), *Approach to a Subject Area Ontology Visualization System Creating*, Proceedings of the 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS '21), pp. 528–540.
3. M. Behringer, P. Hirmer, and B. Mitschang (2018), *A Human-Centered Approach for Interactive Data Processing and Analytics*, Enterprise Information Systems: 19th International Conference, ICEIS 2017, Porto, Portugal, April 26-29, 2017, Revised Selected Papers, S. Hammoudi, M. Śmiałek, O. Camp, and J. Filipe (eds.), Springer (Cham), pp. 498–514, ISBN: 978-3-319-93375-7.
4. M. Behringer, P. Hirmer, M. Fritz, and B. Mitschang (2020), *Empowering Domain Experts to Preprocess Massive Distributed Datasets*, Proceedings of the 23rd International Conference on Business Information Systems (BIS '20), pp. 61–75.
5. T. De Bie, L. De Raedt, J. Hernández-Orallo, H. H. Hoos, P. Smyth, and C. K. I. Williams (2021), *Automating Data Science: Prospects and Challenges*, arXiv preprint arXiv: 2105.05699, pp. 1–19.
6. A. T. Bimba, N. Idris, A. Al-Hunaiyyan, R. B. Mahmud, A. Abdelaziz, S. Khan, and V. Chang (2016), *Towards knowledge modeling and manipulation technologies: A survey*, International Journal of Information Management, Vol. 36, No. 6, Part A, pp. 857–871.
7. S. Böhm and J. Beránek (2020), *Runtime vs Scheduler: Analyzing Dask's Overheads*, Proceedings of the 2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS '20), pp. 1–8.
8. A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou (2020), *Dataset Discovery in Data Lakes*, Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE '20), pp. 709–720.
9. J. Bräcker and J. Brockmeyer (2018), *Characterization and Detection of Food Allergens Using High-Resolution Mass Spectrometry: Current Status and Future Perspective*, Journal of Agricultural and Food Chemistry, Vol. 66, No. 34, pp. 8935–8940.
10. L. Chaari Fourati and S. Ayed (2021), *Federated Learning toward Data Preprocessing: COVID-19 Context*, Proceedings of the 2021 IEEE International Conference on Communications Workshops (ICC Workshops '21), pp. 1–6.
11. D. Collarana, M. Galkin, I. Traverso-Ribón, M. Vidal, C. Lange, and S. Auer (2017), *MINTE: Semantically Integrating RDF Graphs*, Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics (WIMS '17), pp. 22:1–22:11.
12. M. Derakhshannia, C. Gervet, H. Hajj-Hassan, A. Laurent, and A. Martin (2020), *Data Lake Governance: Towards a Systemic and Natural Ecosystem Analogy*, Future Internet, Vol. 12, No. 8, pp. 126:1–126:16.
13. R. Eichler, C. Giebler, C. Gröger, H. Schwarz, and B. Mitschang (2020), *HANDLE–A Generic Metadata Model for Data Lakes*, Proceedings of the 22nd International Conference on Big Data Analytics and Knowledge Discovery (DaWaK '20), pp. 73–88.
14. R. Eichler, C. Giebler, C. Gröger, E. Hoos, H. Schwarz, and B. Mitschang (2021), *Enterprise-Wide Metadata Management: An Industry Case on the Current State and Challenges*, Proceedings of the 24th International Conference on Business Information Systems (BIS '21), pp. 269–279.
15. R. C. Fernandez, P. Subramaniam, and M. J. Franklin (2020), *Data Market Platforms: Trading Data Assets to Solve Data Problems*, Proceedings of the VLDB Endowment, Vol. 13, No. 12, pp. 1933–1947.

16. S. Flender (2019), *Data is not the new oil*, Medium – towards data science.
17. C. Giebler, C. Stach, H. Schwarz, and B. Mitschang (2018), *BRAID — A Hybrid Processing Architecture for Big Data*, Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA '18), pp. 294–301.
18. C. Giebler, C. Gröger, E. Hoos, H. Schwarz, and B. Mitschang (2019), *Leveraging the Data Lake: Current State and Challenges*, Proceedings of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK '19), pp. 179–188.
19. C. Giebler, C. Gröger, E. Hoos, H. Schwarz, and B. Mitschang (2020), *A Zone Reference Model for Enterprise-Grade Data Lake Management*, Proceedings of the 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC '20), pp. 57–66.
20. C. Giebler, C. Gröger, E. Hoos, R. Eichler, H. Schwarz, and B. Mitschang (2021), *The Data Lake Architecture Framework*, Tagungsband der 19. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW '21), pp. 351–370.
21. A. Gorelik (2019), *The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science*, O'Reilly Media (Sebastopol, CA), ISBN: 978-1-491-93155-4.
22. C. Gritti, M. Önen, and R. Molva (2019), *Privacy-Preserving Delegable Authentication in the Internet of Things*, Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (IoT '19), pp. 861–869.
23. W. Grossmann and S. Rinderle-Ma (2015), *Data Provisioning*, Fundamentals of Business Intelligence, S. Hammoudi, M. Śmiałek, O. Camp, and J. Filipe (eds.), Springer (Berlin, Heidelberg), pp. 87–118, ISBN: 978-3-662-46530-1.
24. S. Gupta and V. Venkata Giri (2018), *Practical Enterprise Data Lake Insights: Handle Data-Driven Challenges in an Enterprise Big Data Lake*, Apress (New York, NY), ISBN: 978-1-484-23521-8.
25. S. Hagedorn, S. Kläbe, and K. Sattler (2021), *Putting Pandas in a Box*, Proceedings of the 2021 Annual Conference on Innovative Data Systems Research (CIDR '21), pp. 7:1–7:6.
26. S. Hagedorn, S. Kläbe, and K. Sattler (2021), *Conquering a Panda's weaker self – Fighting laziness with laziness*, Proceedings of the 24th International Conference on Extending Database Technology (EDBT '21), pp. 670–673.
27. O. Herden (2020), *Architectural Patterns for Integrating Data Lakes into Data Warehouse Architectures*, Proceedings of the Eighth International Conference on Big Data Analytics (BDA '20), pp. 12–27.
28. F. Hutter, L. Kotthoff, and J. Vanschoren (2019), *Automated Machine Learning: Methods, Systems, Challenges*, Springer (Cham), ISBN: 978-3-030-05317-8.
29. B. Inmon (2016), *Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump*, 1st Edition, Technics Publications (Denville, NJ), ISBN: 978-1-634-62117-5.
30. S. K. Jensen, C. Thomsen, T. B. Pedersen, and O. Andersen (2021), *pygrametl: A Powerful Programming Framework for Easy Creation and Testing of ETL Flows*, Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVIII: Special Issue In Memory of Univ. Prof. Dr. Roland Wagner, A. Hameurlain and A. M. Tjoa (eds.), Springer (Berlin, Heidelberg), pp. 45–84, ISBN: 978-3-662-63519-3.
31. A. Kallel, M. Rekik, and M. Khemakhem (2021), *IoT-fog-cloud based architecture for smart systems: Prototypes of autism and COVID-19 monitoring systems*, Software: Practice and Experience, Vol. 51, No. 1, pp. 91–116.
32. R. Kimball and M. Ross (2013), *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd Edition, Wiley (Indianapolis, IN), ISBN: 978-1-118-53080-1.
33. S. Kläbe and S. Hagedorn (2021), *Applying Machine Learning Models to Scalable DataFrames with Grizzly*, Tagungsband der 19. Fachtagung für Datenbanksysteme für Business, Technologie und Web (BTW '21), pp. 195–214.
34. R. Korte, J. Bräcker, and J. Brockmeyer (2017), *Gastrointestinal digestion of hazelnut allergens on molecular level: Elucidation of degradation kinetics and resistant immunoactive peptides using mass spectrometry*, Molecular Nutrition & Food Research, Vol. 61, No. 10, pp. 1700130:1–1700130:16.
35. J. Kreps (2014), *Questioning the Lambda Architecture*, O'Reilly Radar.

36. M. Kulmanov, F. Z. Smaili, X. Gao, and R. Hoehndorf (2020), *Semantic similarity and machine learning with ontologies*, Briefings in Bioinformatics, Vol. 22, No. 4, pp. 1–18.
37. C. Labadie, C. Legner, M. Eurich, and M. Fadler (2020), *FAIR Enough? Enhancing the Usage of Enterprise Data with Data Catalogs*, Proceedings of the 2020 IEEE 22nd Conference on Business Informatics (CBI '20), pp. 201–210.
38. S. Langenecker, C. Sturm, C. Schalles, and C. Binnig (2021), *Towards Learned Metadata Extraction for Data Lakes*, Tagungsband der 19. GI-Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW '21), pp. 325–336.
39. P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis (2021), *Explainable AI: A Review of Machine Learning Interpretability Methods*, Entropy, Vol. 23, No. 1, pp. 18:1–18:45.
40. M. R. Llave (2018), *Data lakes in business intelligence: reporting from the trenches*, Procedia Computer Science, Vol. 138, pp. 516–524.
41. J. Luengo, D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera (2020), *Big Data Preprocessing: Enabling Smart Data*, Springer (Cham), ISBN: 978-3-030-39105-8.
42. C. Mathis (2017), *Data Lakes*, Datenbank-Spektrum, Vol. 17, No. 3, pp. 289–293.
43. A. Mavuduru (2020), *Is Data Really the New Oil in the 21st Century?*, Medium.
44. I. Megdiche, F. Ravat, and Y. Zhao (2021), *Metadata Management on Data Processing in Data Lakes*, Proceedings of the 47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '21), pp. 553–562.
45. H. Mehmood, E. Gilman, M. Cortes, P. Kostakos, A. Byrne, K. Valta, S. Tekes, and J. Riekki (2019), *Implementing Big Data Lake for Heterogeneous Data Sources*, Proceedings of the 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW '19), pp. 37–44.
46. N. Miloslavskaya and A. Tolstoy (2016), *Big Data, Fast Data and Data Lake Concepts*, Procedia Computer Science, Vol. 88, pp. 300–305.
47. P. Mishra, A. Biancolillo, J. M. Roger, F. Marini, and D. N. Rutledge (2020), *New data preprocessing trends based on ensemble of multiple preprocessing techniques*, TrAC Trends in Analytical Chemistry, Vol. 132, pp. 116045:1–116045:12.
48. M. A. Musen (2015), *The Protégé Project: A Look Back and a Look Forward*, AI Matters, Vol. 1, No. 4, pp. 4–12.
49. F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena (2019), *Data Lake Management: Challenges and Opportunities*, Proceedings of the VLDB Endowment, Vol. 12, No. 12, pp. 1986–1989.
50. F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost, and R. J. Miller (2020), *Organizing Data Lakes for Navigation*, Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20), pp. 1939–1950.
51. S. Oprea, A. Bâra, and G. D. Ene (2021), *Big Data Solutions for Extracting Load Flexibility Potential and Assessing Benefits*, Proceedings of the 2021 7th International Symposium on Electrical and Electronics Engineering (ISEEE '21), pp. 1–6.
52. D. Petersohn, S. Macke, D. Xin, W. Ma, D. Lee, X. Mo, J. E. Gonzalez, J. M. Hellerstein, A. D. Joseph, and A. Parameswaran (2020), *Towards Scalable Dataframe Systems*, Proceedings of the VLDB Endowment, Vol. 13, No. 12, pp. 2033–2046.
53. D. Petersohn, D. Tang, R. Durrani, A. Melik-Adamyan, J. E. Gonzalez, A. D. Joseph, and A. G. Parameswaran (2022), *Flexible Rule-Based Decomposition and Metadata Independence in Modin: A Parallel Dataframe System*, Proceedings of the VLDB Endowment, Vol. 15, pp. 1–14.
54. A. Preventis and E. G. Petrakis (2021), *CLONE: Collaborative Ontology Editor as a Service in the Cloud*, Procedia Computer Science, Vol. 184, pp. 275–282.
55. F. Ravat and Y. Zhao (2019), *Data Lakes: Trends and Perspectives*, Proceedings of the 30th International Conference on Database and Expert Systems Applications (DEXA '19), pp. 304–313.
56. M. Rocklin (2015), *Dask: Parallel Computation with Blocked algorithms and Task Scheduling*, Proceedings of the 14th Python in Science Conference (SciPy '15), pp. 126–132.
57. J. Rowley (2007), *The wisdom hierarchy: representations of the DIKW hierarchy*, Journal of Information Science, Vol. 33, No. 2, pp. 163–180.

58. P. Sawadogo and J. Darmont (2021), *On data lake architectures and metadata management*, Journal of Intelligent Information Systems, Vol. 56, pp. 97–120.
59. S. Schmid, C. Henson, and T. Tran (2019), *Using Knowledge Graphs to Search an Enterprise Data Lake*, Proceedings of the 16th European Semantic Web Conference (ESWC '19), pp. 262–266.
60. B. Sharma (2018), *Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases*, 2nd Edition, O'Reilly Media (Bejing *et al.*), ISBN: 978-1-492-03299-1.
61. C. Shimizu, K. Hammar, and P. Hitzler (2020), *Modular Graphical Ontology Engineering Evaluated*, Proceedings of the 17th European Semantic Web Conference (ESWC '20), pp. 20–35.
62. C. Stach, F. Steimle, and A. C. Franco da Silva (2017), *TIROL: The Extensible Interconnectivity Layer for mHealth Applications*, Proceedings of the 23rd International Conference on Information and Software Technologies (ICIST '17), pp. 190–202.
63. C. Stach and F. Steimle (2019), *Recommender-based Privacy Requirements Elicitation – EPICUREAN: An Approach to Simplify Privacy Settings in IoT Applications with Respect to the GDPR*, Proceedings of the 34th ACM/SIGAPP Symposium On Applied Computing (PDP '19), pp. 1500–1507.
64. C. Stach, F. Steimle, and B. Mitschang (2019), *How to Realize Device Interoperability and Information Security in mHealth Applications*, Biomedical Engineering Systems and Technologies: 11th International Joint Conference, BIOSTEC 2018, Funchal, Madeira, Portugal, January 19-21, 2018, Revised Selected Papers, A. Cliquet Jr., S. Wiebe, P. Anderson, G. Saggio, R. Zwiggelaar, H. Gamboa, A. Fred, and S. Bermúdez i Badia (eds.), Springer (Cham), pp. 213–237, ISBN: 978-3-030-29195-2.
65. C. Stach, C. Gritti, and B. Mitschang (2020), *Bringing Privacy Control Back to Citizens: DISPEL — A Distributed Privacy Management Platform for the Internet of Things*, Proceedings of the 35th ACM/SIGAPP Symposium On Applied Computing (PDP '20), pp. 1272–1279.
66. C. Stach, J. Bräcker, R. Eichler, C. Giebler, and C. Gritti (2020), *How to Provide High-Utility Time Series Data in a Privacy-Aware Manner: A VAULT to Manage Time Series Data*, International Journal on Advances in Security, Vol. 13, No. 3 & 4, pp. 88–108.
67. C. Stach, C. Gritti, D. Przytarski, and B. Mitschang (2020), *Trustworthy, Secure, and Privacy-aware Food Monitoring Enabled by Blockchains and the IoT*, Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom '20), pp. 50:1–50:4.
68. C. Stach, C. Giebler, M. Wagner, C. Weber, and B. Mitschang (2020), *AMNESIA: A Technical Solution towards GDPR-compliant Machine Learning*, Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP '20), pp. 21–32.
69. C. Stach, J. Bräcker, R. Eichler, C. Giebler, and B. Mitschang (2021), *Demand-Driven Data Provisioning in Data Lakes: BARENTS — A Tailorable Data Preparation Zone*, Proceedings of the 23rd International Conference on Information Integration and Web Intelligence (iiWAS '21), pp. 191–202.
70. M. Stonebraker and U. Cetintemel (2005), *"One Size Fits All": An Idea Whose Time Has Come and Gone*, Proceedings of the 21st International Conference on Data Engineering (ICDE '05), pp. 2–11.
71. C. Thomsen and T. B. Pedersen (2011), *Easy and Effective Parallel Programmable ETL*, Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP (DOLAP '11), pp. 37–44.
72. W. Yun, X. Zhang, Z. Li, H. Liu, and M. Han (2021), *Knowledge modeling: A survey of processes and techniques*, International Journal of Intelligent Systems, Vol. 36, No. 4, pp. 1686–1720.
73. Y. Zhang and Z. G. Ives (2020), *Finding Related Tables in Data Lakes for Interactive Data Science*, Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20), pp. 1951–1966.
74. D. Zhang, M. Hulsebos, Y. Suhara, ˙ Demiralp, J. Li, and W. Tan (2020), *Sato: Contextual Semantic Type Detection in Tables*, Proceedings of the VLDB Endowment, Vol. 13, No. 12, pp. 1835–1848.
75. C. Zins (2007), *Conceptual approaches for defining data, information, and knowledge*, Journal of the American Society for Information Science and Technology, Vol. 58, No. 4, pp. 479–493.