Datenbank-Spektrum (2023) 23:117–122
https://doi.org/10.1007/s13222-023-00450-5

SCHWERPUNKTBEITRAG

A Systematic Approach to Consuming Data in Complex Data Management Landscapes Using Data Consumption Patterns

Corinna Giebler[1] · **Eva Hoos**[1]

Received: 31 January 2023 / Accepted: 15 June 2023 / Published online: 30 June 2023

Abstract
Through data analytics, enterprises can exploit the value their data hold. However, there are still various challenges to be solved, one of them being how to consume data in heterogeneous data management landscapes. To address this challenge, we developed a systematic, hierarchical approach to data consumption, including six data consumption patterns, which is presented in this paper. Each of the six patterns can be associated with multiple implementation patterns that detail its technical realization. We report the application of these patterns in a real-world practical scenario and discuss the benefits of applying the data consumption patterns.

Keywords Data Consumption · Data Management · Practical Experiences

1 Introduction

Advanced data analytics [1] is a key factor in enterprises' transformation to gain competitive advantage [2, 3]. For this reason, enterprises invest in various data analytics use cases, from reporting and dashboarding to AI use cases. Data for these use cases are collected from a variety of source systems and often analyzed in specifically developed data analytics solutions. This results in a heterogeneous data management and analytics landscape built on a wide variety of technologies [4], owned by various teams across business units: from enterprise resource planning (ERP) systems under the governance of source system teams to on-premise and cloud-based data management platforms run by a central IT department to various data analytics tools. Figure 1 depicts the interactions in such a heterogeneous landscape, based on a practical implementation at a globally active manufacturer. This landscape includes a hybrid data lake, i.e., a data lake both on-premise and in the cloud. Specific providers and technologies were omitted, as the approaches discussed in this paper are independent of the underlying technological setup.

✉ Corinna Giebler
Corinna.Giebler2@de.bosch.com

✉ Eva Hoos
Eva.Hoos@de.bosch.com

[1] Robert Bosch GmbH, Stuttgart, Germany

Data originating from ERP source systems are first transferred into an on-premise enterprise data lake owned and operated by central IT (1), where they are available for a wide variety of use cases. As these ERP data are of relational nature, the data lake uses relational storage for their management. From there, they are extracted and moved to the cloud-based part of the data lake (2), where they are stored in a file storage and enriched with data from a cloud-native source from a different cloud provider (3) that belongs to a non-IT business unit. Finally, these combined data are made available cloud-based business application (4) before they are forwarded to the actual data analytics tools (5), both maintained by another business team. This combination of data management environments can be found for various use cases across our enterprise. Due to the variety of involved systems, we deem it a fitting example of a heterogeneous landscape.

This complex combination of participants arises from the enterprise setup, where next to a central IT department, different business units manage their own IT systems to tailor them to their needs. Various technologies are involved in this example, from relational databases to two different cloud providers. Especially cloud providers tend to offer a wide variety of tools and services for data exchange, such as pipelining, and visualization tools, which increases the complexity of setting up such a data pipeline. In fact, delivering data to consumers poses a major challenges to leveraging data's value [4]. Coordination between different parties (source system owners, central IT providers of e.g.,

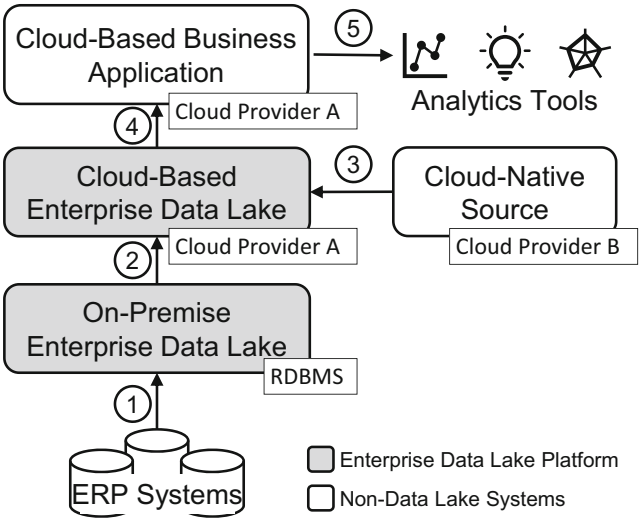

Fig. 1 Interactions in a heterogeneous data management and analytics landscape (example from practice)

the data lake, and business unit IT) is complex and time consuming, and much time is spent on developing fitting interfaces. While there are different approaches to make data available, a systematic comparison is missing, leaving it unclear when to resort to which approach. We confirmed this situation by exchanging observations with several data analytics experts in our organization, namely developers, technology consultants, and platform and data architects, all with a wide variety of experiences in different use cases and technologies. It showed that different technologies are used to implement the same approach to bring data to consumers, e.g., providing data via virtualized tables in relational databases, Azure Synapse, AWS Redshift and many more.

In this paper, we focus on the challenge of data consumption (receiving data from a providing person or system). A systematic approach to data consumption can improve coordination between involved parties. To this end, we make the following contributions:

- We investigate various approaches to data consumption both in literature and in practice to identify categories and recurring patterns (Sect. 2).
- From this, we derive a systematic classification in the form of generally applicable and reusable data consumption patterns. Each pattern comes with various implementation patterns, which describe the realization with specific technologies (Sect. 3).
- We share our experiences from the use of the data consumption patterns in a real-world industry case and show how they improve coordination between teams and lower the effort needed by IT developers (Sect. 4).

2 Related Work

To identify approaches to data consumption, we conducted a literature review focused on data consumption and data exchange.

To address the challenge of enabling data analytics on data from various sources, many enterprises adopted centralized data management platforms such as data warehouses or data lakes [5, 6]. In these concepts, the centralized platform ingests data from all sources of the enterprise, and offers them for consumption through a unified interface [7, 8]. However, the data lake often represents only one node of a data management and analytics landscape [9]. Thus, data consumption must be considered beyond the scope of the data lake.

Beyond data warehouses and data lakes, there are various works available in literature on the challenge of data consumption [10–12]. However, to the best of our knowledge, there is no systematic comparison available. Instead, the focus lies on implementations for given applications rather than on a systematic overview over data consumption approaches.

From the context of information integration stems the distinction of *materialization*, where data are physically moved from the source into the consuming system; and *virtualization*, where data remain only in the source but queries and request are forwarded to the source from the consuming system [13]. While this provides a general categorization of data consumption approaches, it is not sufficient to realize complex data exchanges like in Fig. 1, as the used technologies provide a wide array of materialization and virtualization techniques (e.g., virtualization as tables vs. virtualization as files). Thus, a more granular classification is required.

3 Data Consumption Patterns

In addition to scientific literature, we investigated data consumption in our already existing data management and analytics landscape. We explored how data consumption is handled in different data analytics platforms across the enterprise, e.g., the enterprise data lake, self-service sandbox environments, and existing implementations on the cloud. However, there was no systematic approach to data consumption available within the organization.

To address this gap, we present the *data consumption patterns*, which represent a systematic categorization of data consumption approaches. Data consumption occurs when one person or system (consumer) receives any amount of data from another person or system (provider). The provider is not necessarily the producer of the data but can also be a data distributor such as a data lake. The

exchange of data can either be carried out via a push from the provider's side or a pull from the consumer's side. For simplicity, we see data consumption as interaction between one consumer and one provider. More complex pipelines can be divided into multiple such pairings, e.g., from a source system to the data lake and from the data lake to an analytics tool.

3.1 Pattern Derivation

To derive the data consumption patterns, we examined implementations for data consumption from both literature and practical observation. We then grouped the different implementations based on their characteristics. For example, one group contained implementations in which data was physically transferred and then kept in sync with the source through a variety of means, while another focused on accessing data directly using different query languages. These groups were then abstracted from the specific tooling to generate more abstract and generally applicable patterns.

Our systematic approach to this topic should include the following to tackle the challenges mentioned in Sect. 1: (I) a categorization of approaches to create a common understanding of offerings and implementations between consumer and provider, (II) characteristics of approaches to allow their comparison, and (III) a way to assign abstract patterns to specific tools and implementations for reusability of technical realizations. To reflect these parts, our data consumption patterns are part of a hierarchy as depicted by the different columns in Fig. 2. Each consumption pattern belongs to a specific category (tackling I) and is associated with one or more implementation patterns, reflecting specific tools and implementations (see III) including tooling, interaction between tools, interfaces etc. When making use of defined implementation patterns, developers can draw on technical setups that have already been implemented, tested, and are proven to work. Furthermore, new implementation patterns can be created whenever new technologies are used to implement a consumption pattern, and then be reused across the organization. Characteristics of patterns (see II) are reflected in the consumption pattern itself on a higher abstraction level, and in the implementation patterns on a more granular level, e.g., batching/micro-batching/streaming.

We identified two categories of data consumption patterns in literature and our daily work: *data transfer*, where data are physically moved from provider to consumer, and *data access,* where the provider enables the consumer to read and potentially write data but does not duplicate them to the consumer's system. Each of these categories contains three data consumption patterns, resulting in six data consumption patterns overall, which are detailed in the following subsections. These categories and their associated consumption patterns are depicted in Fig. 2.

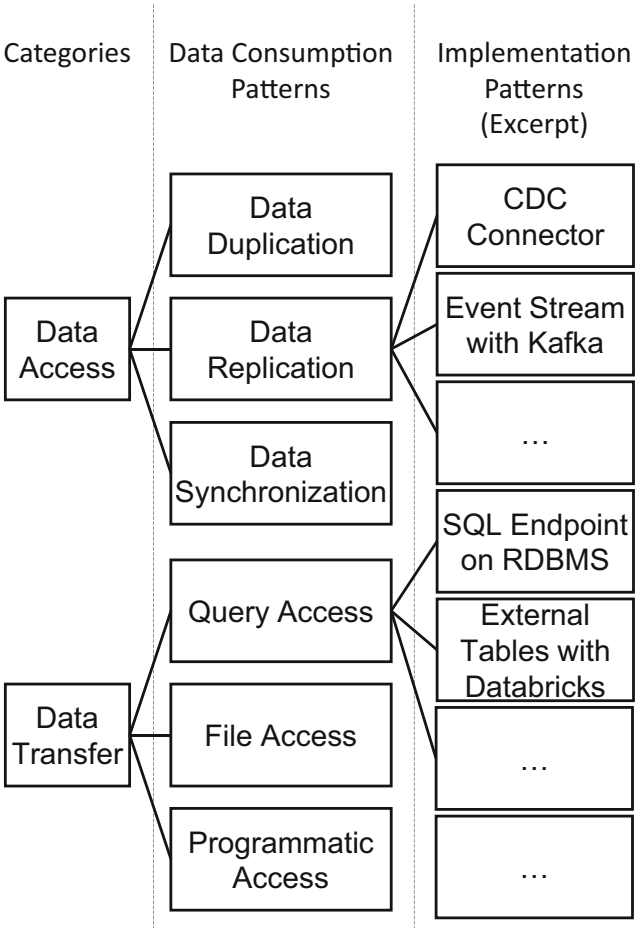

Fig. 2 An overview over of the data consumption patterns and an excerpt of associated implementation patterns

While the two categories are very similar to materialization and virtualization respectively, they are not the same. Rather, materialization and virtualization can be seen as possible results of data access/data transfer: data access can mean providing a view (virtualization) or providing an endpoint from which data can then be pulled physically (resulting in materialization). Data transfer however always has materialization as a result. To facilitate the decision for a specific data consumption pattern, we created a set of guiding questions to first identify the need for data access or data transfer respectively: What environment should the data be available in, complexity of the transformations required, do consumers want to create their own pipeline or receive a centrally maintained pipeline. After that, more detailed questions lead to the specific pattern, e.g., preferred mode of access or need for regular updates on the data.

We furthermore defined a variety of implementation patterns to support developers. An excerpt of these implementation patterns is given both in Fig. 2 and in the following

descriptions of the data consumption patterns. Please note that the examples given are not exhaustive, as data consumption patterns can be implemented using a vast variety of technologies and methods. The specific selection of a suited implementation pattern depends on the technologies used and various boundaries, e.g., security or compliance regulations.

3.2 Data Transfer Patterns

The three data transfer patterns differ in the way that changes in the data are communicated:

1. *Data Duplication*: This data consumption pattern describes a one-time data transfer from provider to consumer. Data are not kept up to date. Exemplary implementation patterns: Spark[1] batch job, ETL job in ETL tool.
2. *Data Replication*: In this data consumption pattern, data are transferred from provider to consumer, similarly to the data duplication pattern. After this initial transfer, changes made on the providers' data are forwarded to the consumer. This forwarding can take place as a continuous stream or in batches. Exemplary implementation patterns: Change data capture (CDC) connector, event stream with Kafka[2].
3. *Data Synchronization*: Again, data are transferred from the provider to the consumer initially in this consumption pattern. After this, data are kept in sync on both provider and consumer side, meaning that changes made in the consumer's dataset are sent to the provider and vice versa. This exchange can take place via a continuous stream or in batches. Exemplary implementation pattern: Bi-directional CDC pipeline.

Using a data transfer pattern results in a materialization of data on the consumer's side. Thus, data transfer patterns should be used if the consumer can persist data in their own storage, complex transformations are required on the data, and the query time should be minimized. It is important to note that even with data replication/synchronization, data between provider and consumer may be inconsistent for short periods of time.

3.3 Data Access Patterns

For the data access patterns, the distinction is based on how the data access is carried out:

1. *Query Access*: Data are accessed using a query language, such as SQL, MongoDB[3] Query Language (MQL), Neo4Js Cypher[4] etc. This consumption pattern is typically found in relational database management systems (RDBMS), where users execute their SQL query directly on the system to retrieve data. Exemplary implementation patterns: SQL endpoint on RDBMS, external tables in databricks[5].
2. *File Access*: Access is provided to a file and data are retrieved in a file format, e.g., csv, pdf or parquet. An example for this consumption pattern is a shared folder on a network, where users can use a file browser to access the stored data. Exemplary implementation patterns: Virtual file explorer, read access to files in HDFS[6].
3. *Programmatic Access*: In this consumption pattern, data are accessed using program code via APIs. Note that if the API is used with a query language (e.g., SQL with Open Database Connectivity ODBC) or returns a file, it is considered part of Query Access or File Access respectively. This pattern occurs when e.g., REST-APIs are used to retrieve data from a web endpoint in a message format (e.g., JSON).

For one, data access patterns can be used to realize data virtualization. In this case they should be used when requests on the data do not require extensive transformations, and when data always should be up-to-date. However, data access patterns can also be used to enable a data transfer from the consumer's side. In this case, the provider gives the consumer access to the data, who then pulls the data for a materialized approach.

4 Pattern Application and Experience Report

To evaluate the patterns defined in the previous section, we applied them in the data management and analytics landscape depicted in Fig. 1. To do so, we divided the path of the data into pairs of consumer and provider, e.g., ERP systems (provider) and on-premise enterprise data lake (consumer) etc. After this, we identified where materialization is required, i.e., where data have to be physically moved. Data should be materialized in both parts of the enterprise data lake to minimize load on the source and reduce latencies. Additionally, data should be materialized in the cloud-based business application to allow the responsible business unit to perform complex transformations on the data without im-

[1] https://spark.apache.org/.

[2] https://kafka.apache.org/.

[3] https://www.mongodb.com/.

[4] https://neo4j.com/developer/cypher/.

[5] https://www.databricks.com/.

[6] https://hadoop.apache.org/.

pacting the central IT systems. The analytics tools require no materialization, as they have small internal storage and no complex transformations are required here.

With these conditions in mind, we assigned data consumption patterns to each pairing of provider and consumer from Fig. 1, e.g., pairing (1): ERP systems and on-premise enterprise data lake. The following paragraphs give an overview over the patterns used, thus showing how the complexity of the landscape can be addressed by the data consumption patterns.

To ensure the actuality of data, changes from the ERP source should be transferred into the enterprise data lake (1). However, no transfer of changes should take place from the data lake to the source to not compromise the operational system. Thus, the *data replication* pattern is used.

As data from the ERP sources are stored in a relational storage on the on-premise enterprise data lake, offering data via SQL is a standard approach. However, data should be materialized in the cloud-based enterprise data lake to reduce latencies and load on the on-premise infrastructure. Thus, a combination of patterns is used for (2): The on-premise enterprise data lake implements the *query access* pattern via SQL, while the cloud-based part uses this access to implement the *data replication* pattern.

For the connection between the cloud-based enterprise data lake and the cloud-native source (3), the *data replication* pattern is used to keep data up-to-date. Note that data in the cloud-native source should not be changed with data from the data lake, ruling out the *data synchronization* pattern.

In pairing (4), data should be made available in the cloud application of a specific business unit. On the way, data should already be transformed for the use in the application. As multiple different applications access data from the cloud-based enterprise data lake for various use cases, the requirements towards data consumption also differ. Overall, five different use cases were implemented for pairing (4), based on different data consumption patterns. In three of them, the business team cloud application was given access to files in the file-based storage, using the *file access* pattern. The business team then implemented the *data duplication* pattern using this access. In the remaining two, the *data replication* pattern was implemented via a pipeline by the team of the cloud-based enterprise data lake.

(5) The data analytics tools used have the ability to consume data via SQL. Thus, the consumption pattern *query access* is used.

We presented and discussed the data consumption patterns with various business teams, developer teams, and architects. We were able to assign existing implementations to our derived patterns with no need for additional patterns.

Furthermore, the usage of data consumption patterns improved the communication between the teams of different systems, as reported by the product owner of the cloud-based Enterprise Data Lake. Before, discussions typically focused on specific tools rather than the actual approach. The architect of the business application confirmed that with the data consumption patterns, a common understanding as to how data should be exchanged could be achieved, including the constraints and implications of the different approaches. This allowed him and his team to make more informed decisions for their application setup. In the cases where a specific set of tools was already in use, the data consumption patterns allowed us to classify the options these tools provide and to extend them with further implementations where needed (e.g., (2), where query access was provided but data transfer was required).

Finally, the data consumption patterns improved the implementation efficiency for developers, as they not only specified how data should be made available and transported, but also gave detailed development information through the implementation patterns.

5 Conclusion and Outlook

Getting data to the right consumers poses a challenge in heterogeneous data management and analytics landscapes. In this work, we presented data consumption patterns, a systematic approach to consuming data. Overall, we identified six data consumption patterns (Query Access, File Access, Programmatic Access, Data Duplication, Data Replication, Data Synchronization) that each belong to one of two categories (data access vs. data transfer). Furthermore, each data consumption pattern comes with a set of implementation patterns that allow for the physical realization of the pattern. We applied the data consumption patterns in our work at a large, world-wide active manufacturer. It showed that the data consumption patterns foster a common understanding between data providers and data consumers, and increase efficiency during implementation.

Future work will on the decision support when it comes to implementation patterns, i.e., which tools to use to realize a specific data consumption pattern for a specific use case.

Acknowledgements The authors would like to thank our colleagues Matthias Czolk, Jana Ickenroth, and Simon Rehker for their work and support in this topic. We also thank our supervisors and the DAST@CI program at Bosch for enabling this research.

Conflict of interest C. Giebler and E. Hoos are employees at Robert Bosch GmbH. E. Hoos declares that D. Nicklas contributed to her PhD thesis as second reviewer.

References

1. Bose R (2009) Advanced analytics: opportunities and challenges. Ind Manag Data Syst (idms) 109:155–172
2. Pearson T, Wegener R (2013) Big data: The organizational challenge. Bain & Company. http://2016.leagueofleadingladies.com/wp-content/uploads/2014/06/BAIN_BRIEF_Big_Data_The_organizational_challenge.pdf
3. Santos MY, Sá JO e, Andrade C, Lima FV, Costa E, Costa C, et al. A Big Data system supporting Bosch Braga Industry 4.0 strategy. International Journal of Information Management 2017;37:750–60.
4. Gröger C (2021) There is no AI without data. Commun ACM 64:98–108
5. Madsen M (2015) How to Build an Enterprise Data Lake: Important Considerations before Jumping. Third Nature Inc
6. Mathis C (2017) Data Lakes. Datenbank Spektrum 17:289–293
7. Hai R, Quix C, Zhou C (2018) Query Rewriting for Heterogeneous Data Lakes. Proceedings of the 22nd European Conference on Advances in Databases and Information Systems. ADBIS, vol 2018
8. Mami MN, Graux D, Scerri S, Jabeen H, Auer S, Lehmann J (2019) Uniform Access to Multiform Data Lakes using Semantic Technologies. Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services. iiWAS, vol 2019
9. Dehghani Z. How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh. MartinFowlerCom 2019. https://martinfowler.com/articles/data-monolith-to-mesh.html. Accessed 12 June 2023.
10. Cunha F, Villas L, Boukerche A, Maia G, Viana A, Mini RAF et al (2016) Data communication in VANETs: Protocols, applications and challenges. Ad Hoc Netw 44:90–103
11. Ham J, Koo Y, Lee J-N (2019) Provision and usage of open government data: strategic transformation paths. IMDS 119:1841–1858
12. Thoben K-D, Lewandowski M (2016) Information and Data Provision of Operational Data for the Improvement of Product Development. In: Bouras A, Eynard B, Foufou S, Thoben K-D (eds) Product Lifecycle Management in the Era of Internet of Things, vol 467. Springer, Cham, pp 3–12
13. Leser U, Naumann F (2007) Informationsintegration – Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen. dpunkt.verlag, GmbH