UDC 004.6, 004.8

Y. Vlasiuk, V. Onyshchenko

AUTOMATED DETECTION OF THE DATA PRODUCT CONSUMERS IN DATA MESH

Abstract: Data product became one of the core principles based on which Data Mesh architecture is defined. Being a main and recommended unit for collaboration between domains in a mesh, data product plays the role of communication contract between the components. Distributed nature and high scale of Data Mesh might lead to significant and uncontrolled growth of data product usage by various consumers. This article analyzes existing approaches and tools for detecting data product consumers in Data Mesh and proposes alternative automated approach. Component diagram and execution algorithm are designed as part of the research and presented in the article.

Keywords: Artificial Intelligence, Data Mesh, data product, data lineage.

Introduction

Data Mesh is a new data architecture pattern for designing distributed data platforms at scale to support rapidly growing demand from organizations towards transforming their raw data into valuable business asset. This architecture pattern is very significant so that requires also a shift of organizational paradigm towards developing and managing data-driven products, which focuses on distributing ownership and governance of data across the organization, rather than centralizing it in a single team or department.

Data Mesh should be built based on four main principles: domain-oriented decentralized data ownership and architecture, self-serve data platform, and federated computational governance, data as a product [1]. Inês Araújo Machado, Carlos Costa and Maribel Yasmina Santos give comprehensive explanation for each of principles while mentioning that Data Mesh applies “Product Thinking” concept to the data, which forces organizations to treat their consumable data as ready-to-use products. Authors provide well-aimed concept that as data is considered as product, consumers of this data should be treated as customers so that their needs in terms of data quality, availability and accuracy should be met [2].

However, before designing and building approaches that will provide abilities to meet data quality requirements for all consumers by tracking and executing data validation checks, resolving conflicting and contradicting requirements, aligning them with platform-wide rules, it’s critical firstly to identify what components of Data Mesh can consume data products and afterwards catalog and track all consumers of data products in Data Mesh platform. Section 2 presents analysis of Data Mesh components which are consuming data products. Section 3 contains analysis of existing approaches and tools for data lineage and data consumers tracking in Data Mesh. Section 4 describes proposed solution for automated data product consumers identification based on metadata with components diagram and execution algorithm. Finally, section 5 discusses pros and cons of proposed approaches and concludes with some remarks.

Data product consumers in Data Mesh

Setup of data product consumers highly depend on domain-data archetypes applied in Data Mesh. There are three of them: source-aligned domain data, aggregate domain data and consumer-aligned domain data [3]. Last archetype exactly dictate that consumers of the data product should be domain centric and thus, data products should be represented to fit a particular use case of a particular consumer. Zhamak calls this “fit-to-purpose” domain data.

Data product consumers can be represented by various forms and configurations, but their main characteristic is a need to access and use data products to do their job. Some of the common components that can consume data products in a Data Mesh architecture include:

1. Data applications
2. Data services
3. Data pipelines
4. Reports and business intelligence products
5. Machine learning models and algorithms
6. Data Products

Data applications are software applications that are designed to consume and process data. These applications can be developed and managed by individual domains and can consume data products from other domains or from external sources. Data applications can be built using microservices architecture, programming languages like Java, Python, C# and others. They can read data from data products and expose it via API, visualize on UI, export to files.

Main aim of the data services is to provide access to data products. They are usually built for cross-platform data sharing, support integrations with external services and providers. Data services can also be built to serve as a data product sharing mechanism inside the Data Mesh across the domains. Technological stack is similar to the stack, used for data applications.

Data pipelines are automated workflows that are designed to process data. These pipelines can consume data products from other domains or from external sources and can transform or enrich the data as it flows through the pipeline. Technological stack available for building data pipelines is very broad and includes SQL-based transformations, utilization of distributed data processing frameworks like Spark, Databricks, proprietary platforms with own domain-specific language (DSL) like Talend, Informatica and others.

Reports and business intelligence (BI) products are produced by software tools that are designed to analyze data. These tools can consume data products from other domains or from external sources and can provide insights into the data that are relevant to the specific use case or application. This is usually a final consumer of the data product before presenting the data to the end-user. Variety of business intelligence tools is also big while absence of a single standard (like ANSI SQL) leads to very broad technological stack and implementation approaches: some tools allow to define data model per BI report like Microstrategy, other like PowerBI contains powerful data model engine that allows to define re-usable models and enhance those with built-in data transformations. DSL and data model definition are significantly different across platforms.

Machine learning (ML) models are software products that are trained on data and used to make predictions or classifications. This use case considers consuming existing data products to train an ML model and then deploying the ML model as a data product to make inferences and generate new data. ML model can consume data products on different steps of own lifecycle: on training step where model will be using data products from various domains to pass the training and on inference step where model will be using existing data products to handle incoming requests. Corresponding processes like data drift detection and model concept drift detection can potentially also be an example of data product consumers.

Finally, data product can be also a consumer of a set of other data products. If we review curated dataset in a data mart area of the data warehouse as a data product, it can be represented as SQL view (materialized or runtime) component which in turn can use other data products (curated tables, data marts, data models).

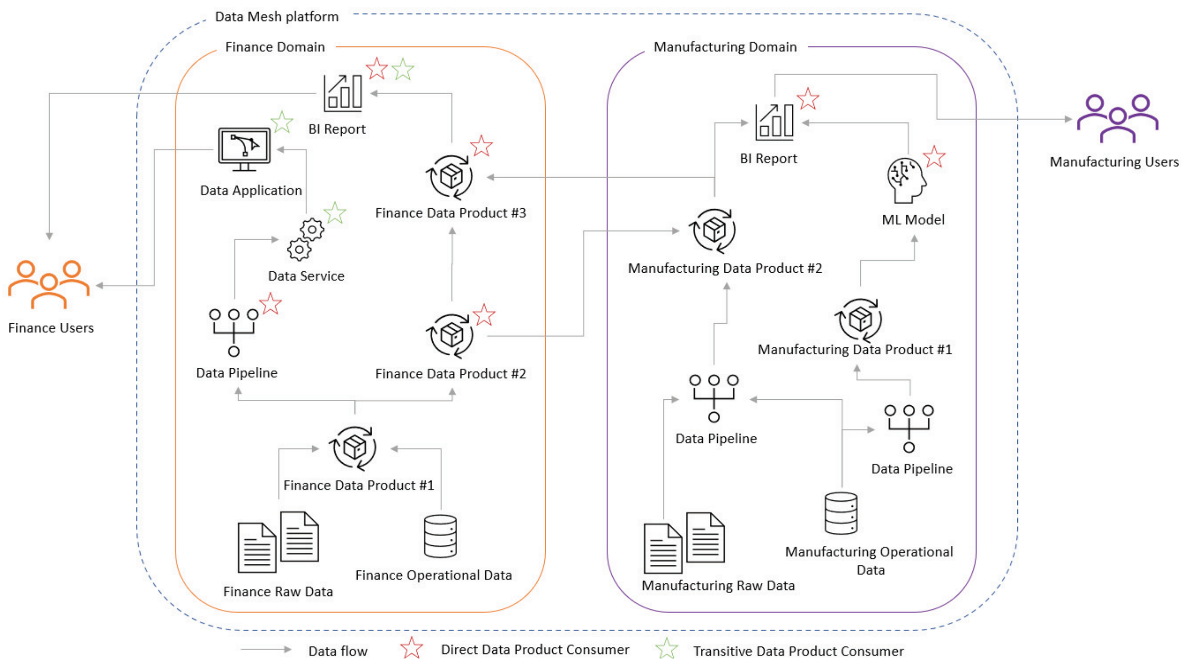

Fig. 1. Data Product consumers in Data Mesh

As we see, variety of components in data platform, built using Data Mesh architecture, is significant. The bigger platform grows, more domains are added – the bigger number of data product consumers becomes. However, the main complexity is in the fact the data product consumers can be chained into complex data flows. It can start with data product, which is built using set of other data products, continue with data pipeline which uses initial data product, then split into BI object that triggers the pipeline and data service that executes the pipeline on a regular basis. There might be multiple levels of nesting until the final destination of data product. To manage the complexity of large data flows, they are represented in visual form of data lineage graphs where data consumers are represented.

Data lineage and data consumers tracking in Data Mesh

The process of data lineage and its technical implementation options started to be researched for the database management domain. Masaya Yamada, Hiroyuki Kitagawa, Toshiyuki Amagasa and Akiyoshi Matono describe lineage as the process of “tracing the source data from which the analysis results are derived” [4]. The main idea of data lineage is to track and, in some cases, visualize the flow of transforming data from the source state, through all intermediate transformations and states, up to target state, consumable by the end user. When data platforms, like data fabric or data factory, started to become widely used, data lineage process was adopted to the new environment. The goal remained the same, but implementation was modified as now data flow became distributed, having data to be moved from one component of the platform to another. Data is moving from data warehouse (DWH) layer to ETL (extract-transform-load) layer for data cleansing and enrichment. After that data is returned to DWH layer where can be transferred multiple times inside the platform between various components until it finally reaches the target state.

When we are analyzing data lineage in Data Mesh platform, we are adding another dimension of complexity like it happened during switch from database management platforms to data platforms. In Data Mesh, data can flow and can be transferred not only between the components of data platform (from DWH to ETL and Reporting) but also between domains. Data consumer can be located in a same domain where data is located or in totally different domain, consuming data product built out of source data. Data as Product principle of Data Mesh is aiming to standardize cross-domain dataflow. Marian Siwiak, Sven Balnojan, Jacek Majchrzak describe data product as data contract between domains in Data Mesh and provides an example of defining requirements for the data contract from consumer and producer sides [5]. Building a Data Mesh platform, we should follow the rule of data contracts: raw data shouldn’t leave the borders of source domain, only data product can be a unit of data that can be exchanged between the platform components across various domains.Approaches of building data lineage in Data Mesh can be divided in three categories:

- Event-driven lineage

- API-based lineage
- Metadata-based lineage

Event-driven lineage heavily relies on event-driven architectures, where data events are used to trigger data processing and analysis. By capturing metadata about each event, it is possible to build a complete picture of the data lineage. This approach can be adopted during the initial stage of building components in Data Mesh platform as each consumer of the data should report an event of data consumption into some central storage. Afterwards, centralized event analyzer should be collecting all events in historical order and build the complete schema of data flow. The drawback of this approach is difficulty in adoption for existing, established Data Mesh where introducing changes to all data consumers is problematic as they are placed in various business domains owned by different teams.

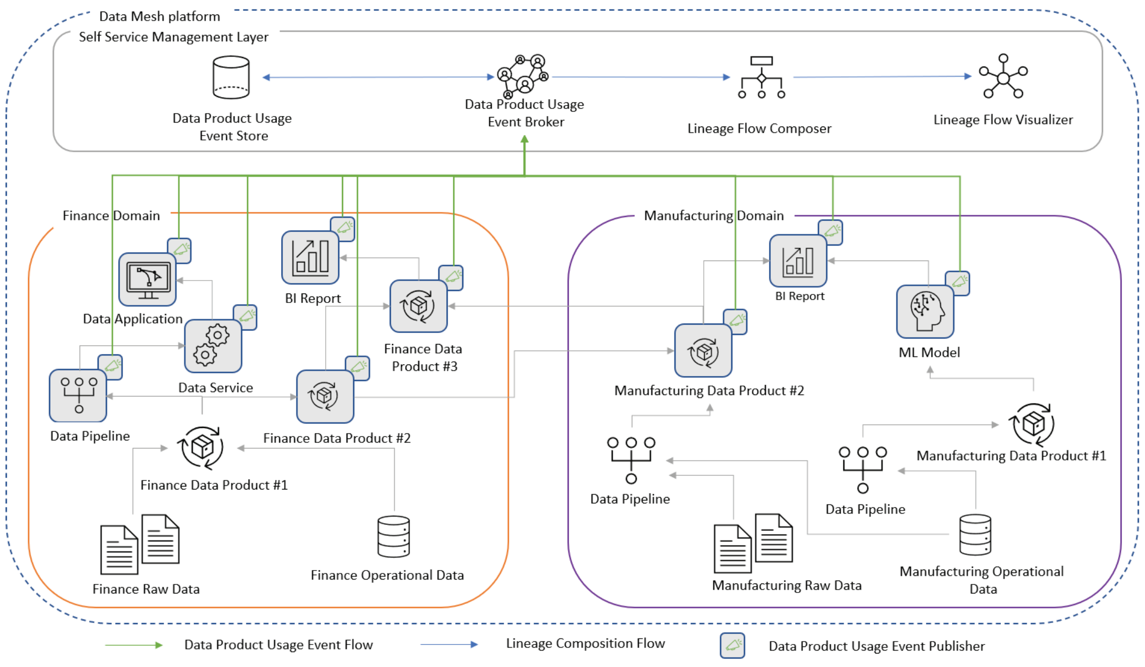

Fig. 2. Event-driven lineage in Data Mesh

API-based lineage is built on top of Data Mesh recommendation to use of standard APIs for integrating data between domains. By capturing metadata about each API request and response, it is possible to build a complete picture of the data lineage. This approach is similar to events-based lineage but doesn’t require to modify each consumer of the data to notify about the event of using particular data product. All API requests for data product consumption could go through one proxy component which will record the fact of consumption. This solution has own drawbacks: necessity to have all data product consumption via API and proxy component as single point of failure. First drawback is dictated by the fact that quite frequently data products are consumed via non-API approaches: via direct connection to databases through

JDBC/ODBC protocol, via reading files from file systems, like HDFS. To adopt these usage patterns to API-based approach, separate wrapper components should be created that encapsulate logic of data product consumption and expose API interface. These wrappers will be executed, and corresponding API execution will be tracked by proxy component. Having centralized proxy component for all API-based data product consumption, puts significant availability and performance requirements for this component of the platform. Autoscaling, distribution across availability zones, disaster recovery should be supported for it to make sure cross-domains communication is reliable and stable.

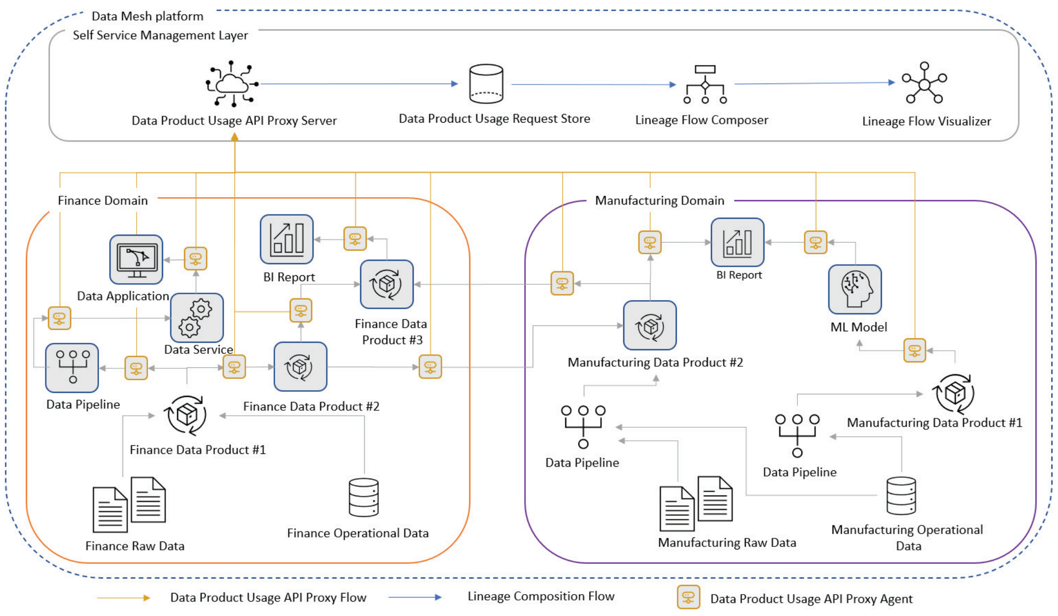

Fig. 3. API-driven lineage in Data Mesh

Metadata-based lineage stands on recommendation of Data Mesh platform to use a centralized metadata management system, which tracks the metadata for all data domains. This system can be used to capture data lineage information, including the data sources, transformations, and destinations. Many widely used commercial and open-source tools already added support of Data Mesh architecture into their feature list. Datakin and Apache Atlas as open-source tools allow to build data catalog, serve as metadata management and governance tool. Both tools automatically collect metadata out of data consumers but their set of supported platforms is limited: Atlas focuses on Hadoop-based platforms while Datakin targets cloud providers and doesn’t have wide support of traditional data warehouses like Oracle or MS SQL Server. Commercial tools like Collibra or Alation has wider variety of supported platforms however follows “push model” when metadata about consumers and data products which they are using, should be send to the tools for them to be able to build data lineage flows. It will require changes to data consumers which makes usage of this platform in existing Data Mesh platforms problematic. Proposed solution of automated metadata-based consumer detection is collecting metadata of the platforms which are consuming data products in Data Mesh platform automatically. It doesn’t require to modify existing data consumers at all to emit event or pass the metadata to centralized platform. Approach is based on metadata extraction, parsing and identification of data product which is used by the data consumer.

Automated metadata-based approach for data product consumers identification

In section 2 it was defined six types of data product consumers. Proposed approach will be focusing on automated identification of data pipelines, reports and business intelligence objects and data products consumer types leaving other three consumer types as out of scope for this article. Main component of the solution is a standalone software application that consists of multiple modules and performs following set of activities to identify data product consumers:

- Identify metadata extraction type.
- Test the connection to the data product consumer platform.
- Extract the metadata from the consumer platform.
- Identify data model of the consumer platform.
- Parse retrieved metadata according to identified data model.
- Identify usage occurrence of data objects.
- Map identified data objects to registered data products; filter out data objects with raw domain-specific data.
- Combine list of data products, their consumers across all domains of Data Mesh.
- Identify transitive data product consumption via intermediate data products usage.

Software architecture of the standalone software application consists of following main modules:

- Metadata extractor
- Application provisioner and configurator
- Analyzer
- Dependency builder
- Monitoring dispatcher
- Management console

Initially application is determining what approach of metadata extraction is supported by the platform that represents data product consumer. The most widely used are three:

1. JDBC/ODBC-based protocol connection to the data product consumer platform

2. Metadata artefacts extraction
3. HTTP-based API connection

JDBC/ODBC protocol is used to connect to data warehouse platform which contain data objects which use data products. Metadata artefacts extraction is used to export metadata information from data product consumer platform into text-based formats and files, like csv, json, xml, sql and other. This approach highly depends on capabilities that platform support as export functionality should be supported natively. Finally, HTTP-based API connection can be used as an alternative approach to metadata artefacts extraction. Some ETL and BI platforms exposes API that allows to read information about their structure, objects, components, dependencies, retrieve executable code for further analysis.

Once connection option is determined, verification of connection is performed to ensure that network between the application and data product consumer platform can be established. Connection verification stage is platform specific as different RDBMS and ETL platforms are using different ports for connection: 1521 for Oracle, 1433 for MS SQL Server, etc. Inbound TCP traffic should be enabled on instance where corresponding platform is hosted.

Connection verification step is pre-requisite for the start of metadata extraction step. In scope of it, application is reading available metadata about the data product consumer platform. Metadata extraction is a highly variable operation so extractor components should be pluggable into the overall architecture, giving an ability to easily create extractor for any new platform which needs to be supported. Metadata extraction from the platforms, compliant with ANSI SQL, is performed from “information_schema” [6]. It allows to get the information about tables, views, stored procedures, and functions including their dependencies and executable code. Information schema of MS SQL Server contains table “sql_expression_dependencies” that allows to retrieve all tables, views and other objects which are read, created or modified by particular database object. Below SQL query will return consumed objects by “Warehouse.StockMovement_Post” stored procedure of open source “World Wide Importers” database:

```
USE WideWorldImporters
GO
SELECT referencing_entity_name, referencing_id, referenced_entity_name, referenced_schema_name, referenced_entity_name, referenced_id, referenced_minor_id, is_caller_dependent
FROM sys.sql_expression_dependencies
WHERE referencing_entity_name = 'Warehouse.StockMovement_Post';
```

Metadata extraction for non-ANSI SQL platform is performed based on metadata artefacts extraction or via API extraction. This type of extraction is applied to ETL and Reporting platforms. Majority of them export their structure and executable as artefacts as files or folders. For example, SQL Server Integration Services (SSIS) platform allows to export structure of ETL pipelines as a set of .dtsx files which have xml structure. These files contain definition of executables which are run as part of the pipeline. Executables in case of SSIS are mostly written in SQL, however options with code in C# are also possible. Such SQL scripts retrieved from exported .dtsx file as metadata artefact, will be referencing particular data products and raw data objects. To determine which exact data products are consumed, executable should be parsed and analyzed.

API-based extraction can be used in parallel or as alternative to artefacts-based extraction. For example, PowerBI business analysis platform provides an option to export its metadata as .pbix files as well as get metadata information about reports, visualizations, data models and data objects via API. Combining these two approaches, it’s possible to build complete data model of data consumer as a PowerBI report including utilized data products.

To analyze what data products are used by each consumer, corresponding metadata information that includes executable code snippets should be parsed. For this purpose, libraries and frameworks based on formal grammar notation also known as Backus-Naur form (BNF) are used. Executables extracted from metadata are described as a set of production rules, where each rule defines a non-terminal symbol (a syntactic category) in terms of other non-terminal and/or terminal symbols (the building blocks of the language). The non-terminal symbols represent syntactic categories such as expressions, statements, declarations, and so on, while the terminal symbols represent the actual tokens or symbols that appear in the language. Python libraries like PLY, ANTLR, Lark and other provides high-level abstraction on top of terminal symbols of BNF, giving an easy access to parsed tokens and classifying them. Results of syntax parsing of executables will be represented as list of consumed objects. Having a full list of objects with their types extracted from the “information_schema” gives an ability to look up objects from executable in the full list and identify which data products are consumed by the platform, ETL, BI or DWH in case data product is using other data products for creation.

At this stage, data consumers are already identified for the set of data products. However, only explicit consumption, when platform directly uses data product, is identified. Very frequently, data products are used transitively: one data product is used by another, which is referred by data transformation logic, which is executed from the final data product exposed as business intelligence report. To trace such indirect dependencies, proposed solution includes dependency builder component. This component encapsulate logic to build a chain of dependencies starting from defined data product. Graphs building and traversing algorithms can be used for this purpose. Depth-First Search algorithm explores the nodes of the graph in depth-first order, starting from a given node, and builds the graph as it goes. Python libraries like NetworkX, igraph encapsulate the logic of working with the algorithm providing high-level abstraction on top of it. Result of such analysis will be a data model which represents the graph of transitive dependencies between data consumers and data products as presented on Fig. 4.

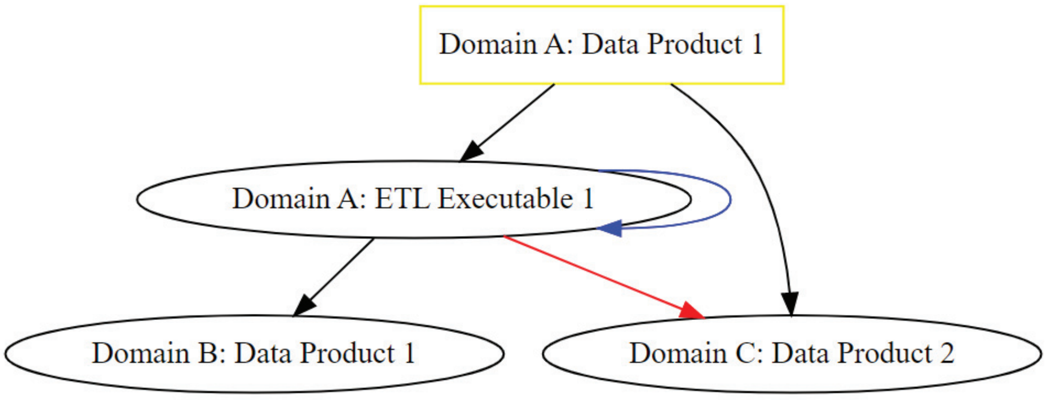

Fig. 4. Transitive dependencies between data products and data product consumers in Data Mesh

Conclusions

Automated method of data product consumers detection in Data Mesh platform is proposed as a result of this research. Evaluated method is based on metadata extraction from data product consumers. It has number of benefits comparing to event-based or API based methods of data product consumers detection. In particular, proposed method doesn't require to modify existing data product consumers so can be adopted for existing platforms without changing verified and tested components. At the same time, metadata extraction doesn't require introducing intermediate API Proxy Agents and API Proxy Server which can be a single point of failure in data flow execution, can slow down the process and requires significant operational effort. Metadata extraction method can be added to existing platform without changing operating data flow that supports critical business processes.

At the same time, proposed method has own drawbacks. It doesn't cover data service, data application and ML consumers as metadata is usually absent for them or can not be extracted and processed easily and with enough level of predictability. High dependency on artefacts provided by data product consumers is another drawback of proposed approach. In case, particular metadata artefact is missing or doesn't contain information about consumed objects, data product consumers identification will become difficult.

In further research, these drawbacks will be investigated deeply and resolution options will be presented.

REFERENCES

1. *Dehghani Z.* How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh. 2019. / Retrieved from: https://martinfowler.com/articles/data-monolith-to-mesh.html

2. *Inês Araújo Machado, Carlos Costa, Maribel Yasmina Santos*. Data Mesh: Concepts and Principles of a Paradigm Shift in Data Architectures, 2021. / Retrieved from https://reader.elsevier.com/reader/sd/pii/S1877050921022365?token=D16A531B8281668BA35608A988D318A68D6213FFDB2F7AE66002018933044CF03F2C5322EE58DD8DA7BD62ED11BBF6BC&originRegion=eu-west-1&originCreation=20221111101328

3. *Zhamak Dehghani*. Data Mesh: Delivering Data-Driven Value at Scale. / March 2022. ISBN: 9781492092391

4. *Yamada, M., Kitagawa, H., Amagasa, T. et al.* Augmented lineage: traceability of data analysis including complex UDF processing. / The VLDB Journal (2022). https://doi.org/10.1007/ s00778-022-00769-7

5. *Marian Siwiak, Sven Balnojan, Jacek Majchrzak.* Data Mesh in Action.

6. *Melton, Jim; Simon, Alan R.* Metadata, Repositories and The INFORMATION_SCHEMA. Understanding the New SQL: A Complete Guide. / The Morgan Kaufmann series in data management systems, ISSN 1046-1698. Morgan Kaufmann

7. *Piethein Strengholt* Data Management at Scale: Modern Data Architecture with Data Mesh and Data Fabric / April 2023. ISBN: 9781098138868