How to build, implement and leverage a successful data architecture using data mesh architecture pattern – A view from BCG and Teradata Corporation.

Authors:

Robbert Naastepad (Teradata)

Jakub Fila (BCG Platinion)

Jens Mueller (BCG Platinion)

Vincent von Hof (BCG Platinion)

Table of Contents

1. Data Mesh—a new approach building on existing technology foundations

Data mesh is a frequently discussed topic as it is perceived as game changer in data architectures. However, it is often misunderstood as a concept as well as are its consequences.

What is data mesh then? It is a shift in architectural and business usage paradigm that reorients data architecture towards business specific to a company.

It is, however, no remedy for every issue; nor a silver bullet to solve any fundamental data problem. Let's start the article with an honest view into what Data Mesh really is as well as what it is not.

Data mesh IS: A way of client-centric organization of data and data interfaces for efficient and business aligned consumption and monetization of the data. It serves business domains with relevant, timely and high-quality data views/perspectives exposed as business understood data products/services. It shifts the paradigm of designing data architecture to business derived and business oriented. It is also a way of decomposing a highly entangled/coupled monolithic data architecture into a domain-oriented architecture, delivering on the premise of so called self-service. Decentralization and federated governance are concepts at the center of this architectural style enabling its usability and adaptability.

Data mesh IS NOT: A replacement for data warehouse, data lake or lake house. It is not a replacement for properly designed data integration nor is it a remedy for poor data management at back-end/core systems. It is not a magic key to self-organize the data in the organization that has no proper data organization and lineage between its systems and repositories.

Data mesh SHOULD NOT: Be confused with Data Fabric. Although both are data management architectures, a data mesh produces data products specific to (business)domains. A data fabric produces many data artifacts of which a data product can be one. Where a data fabric is a more technical focused architecture, data mesh tends to lean more towards the organization of data management and is an information architecture. The data fabric center is metadata, that of a data mesh is the domain.

1.1 Principles of data mesh

In a canonical design of data mesh, five principles need to be highlighted:

- Domain orientation
- Data as a product
- Self-service
- Federated governance
- Agility
- Each of those plays a vital role in building a value of data mesh solution as data mesh is more a business philosophy than pure technical concept.

1.1.1 Domain orientation

Data are organized across domains with clear domain ownership spanning from operational sources, via [optional] centralized data to dispositive data consumption. The examples might be party/client/partner domain or deposits domain in the banking area. Each defines its main business objects, data describing them and ways the data are served also engaging direct owners into the process of definition of the domain.

1.1.2 Data as a product

Data cataloguing should allow discovery of the data by their consumers and stakeholders. APIs that operate on self-service premise ensure interoperability of the data products. Declaration of data quality is done in data delivery agreements. Data products are simply a concept and implementation of data entities and services enabling them. They typically form a group of business meaningful data objects being served and operated. As an example, product: Contracts in the service provider company would enable all the business relevant information on the contractual agreements with company's clients thus allowing insight into the data as well as potentially manipulating the data. The products can be built as raw business semantics on the data or built upon other, more fundamental products.

The products are typically delivered and developed further using agile approach with data ownership and product team established for each of them.

1.1.3 Self-service

Data-product teams typically spin up technical components needed to extract, load, transform, store, publish and expose the data. The data are accessible by means of data/service contracts and the services providing them operate at scale. Data consumers only need to know the contract and endpoint to be able to establish the client.

In existing data architectures, technical components are often set in stone and managed by a centralized team. The interaction with that team is a quintessential requirement. Data owners must await their support before they can proceed. In self-service architectures the data owners can shape the products themselves deciding on how the data are served and consumed.

1.1.4 Federated governance

The data are harmonized in centralized fashion using data catalogues, metadata management solutions for common discovery, consumption and interoperability. Data-product teams have significant freedom (not breaking central principles though) to build data assets allocated in domains in the most efficient way matched to the specificity of the domain. Computational governance is in place to enforce the data delivery agreements.

1.1.5 Agility

Canonical agile methodology and organization combined with a set of data-specific capabilities provide a tooling for data mesh build. DataOps provides a framework for dev-test-deploy of various data assets in automated and scalable way. Obviously agile methodologies proved to be useful for other, older styles, however data mesh directly benefits from it as self-service and domain orientation are best aligned with them.

1.2 Architectural concepts that support data mesh

Data Mesh as a concept is fresh, but elements of the idea have existed long before and so has the tooling. It is the assembly of the concepts that reorient data architecture towards business services and support business domains in the most relevant way. The applicable concepts to be explained in the later part of the article:

- Domain-driven design
- Microservice architecture
- API driven integration
- Resource-oriented architecture

1.2.1 Domain-driven design

Software design consultant and author Eric Evans defines domain driven design as the most fundamental underlying paradigm that supports modelling and implementation of business services grouped by specific business areas (called domains). For the data mesh, the notion of a data domain is very often used to reflect the data specificity of the architecture, nevertheless it is useful to maintain a holistic view on the domain as a such.

1.2.2 Microservice architecture

A way of decomposing IT systems into loosely coupled parts that support very specific business services as well as can be extended and scaled independently of the other chunks. Microservice architecture allows to specialize service implementation towards defined business services and optimize its design.

Although data mesh pattern would typically benefit from usage of microservice architecture, it is not a must and other application patterns can be considered and still work well. The main issue is to make them work with data mesh.

1.2.3 API driven integration

Integration pattern that connects the application using high level APIs that are typically implemented as RESTful services using http protocol. Quality APIs are usually self-descriptive, cover a specific business area and expose it as a service, are manageable and allow for loose couplings between the applications/systems.

In fact various types of APIs and API usage can be imagined for data mesh (see further) and best fit for purpose should always be used. RESTFful API is, however one of one of the most versatile ways of communicating with the data/service platform.

1.2.4 Resource oriented architecture

Exposure of resources rather that behavioral APIs/services. A single resource (e.g. current account) exposes all the allowed operations on it such as read, modification, creation of a new one etc. Resource is typically well defined and implements a clear and relevant business term.

Combination of the above mentioned four paradigms allows for defining and creating data domains, supporting them with properly designed and implemented microservices as well as expose/consume the data by well defined, business meaningful services. All of the paradigms have been known for some time while data mesh concept is just a glue that allows them to cooperate to provide business view of the data.

Obviously other choices such as event publishing are also viable and can be implemented depending on the needs and chosen patterns.

2. Data mesh vs. data warehouse vs. data lake

Data mesh is a functional and behavioral paradigm settling data consumers in the center of the architecture. It can be implemented in several ways utilizing various concepts for organizing source data as well as using various integration and enablement patterns.

Each of the architecture styles mentioned addresses some business problems.

Data warehouses have been here for many years, serving mostly as structured sources of structural (but not always and only) data, provider of the data for reporting, consolidating history, sometimes general ledgers, etc. Their premise was centralizing the data to one source of facts about the organization.

Data lakes have been brought to life to enable capabilities of mass processing, storage and categorization of the unstructured, high volume, velocity, and veracity of data. If the data warehouse pattern was typically schema at write, data lake is schema at read—making it high volume with fast availability, but using lots of user's attention and awareness.

Data warehouses and lakes are often combined. A "lake house" is a build offering the capabilities of both. Although seasoned, they are not obsolete.

Data mesh, on the other hand, is all about self-service, and decoupling enterprise data management and consumption into simpler and less entangled data products. Data mesh is not a direct replacement of centralized data source such as data warehouse or data lake.

There are patterns in place that allow seamless collaboration between data mesh and centralized data stores representing a single source of facts. In such a case data mesh would typically be produced by a consolidated storage layer. Moreover, there are architectural patterns and data models that allow for organization of single source of facts so that it automatically organizes the data into domains and enables easy exposure of data mesh as a prevalent consumption pattern. There are three types of domains:

Source-oriented domain (source domain):

- Sourced from enterprise core applications
- Facts and reality of business
- Immutable timed events / Historical snapshots

- Change less frequently
- Permanently captured

Consumer oriented domain (consumer domain):

- Sourced from data products from source-oriented domain(s) or integration domain(s)
- Fit for consumer purpose
- Aggregation / Projection / Transformation
- Change often
- Can be recreated

Integration domain:

- Sourced from data products from source-oriented domain(s) or consumer domain(s)
- Integration over domains
- Granularity specific to consumer domains requirements
- Change specific to consumer domains requirements
- Can be recreated per source domain

3. Ways of organizing and sourcing distributed data mesh

Distributed data mesh is not a fixed pattern that can be implemented in one canonical way. There are at least two dimensions across which we can make decisions on its sourcing:

- Ways of organizing data (schemas) and allocating/collocating them
- Ways of sourcing the data for distributed data mesh domains

3.1 Ways of organizing schemas for the distributed data mesh

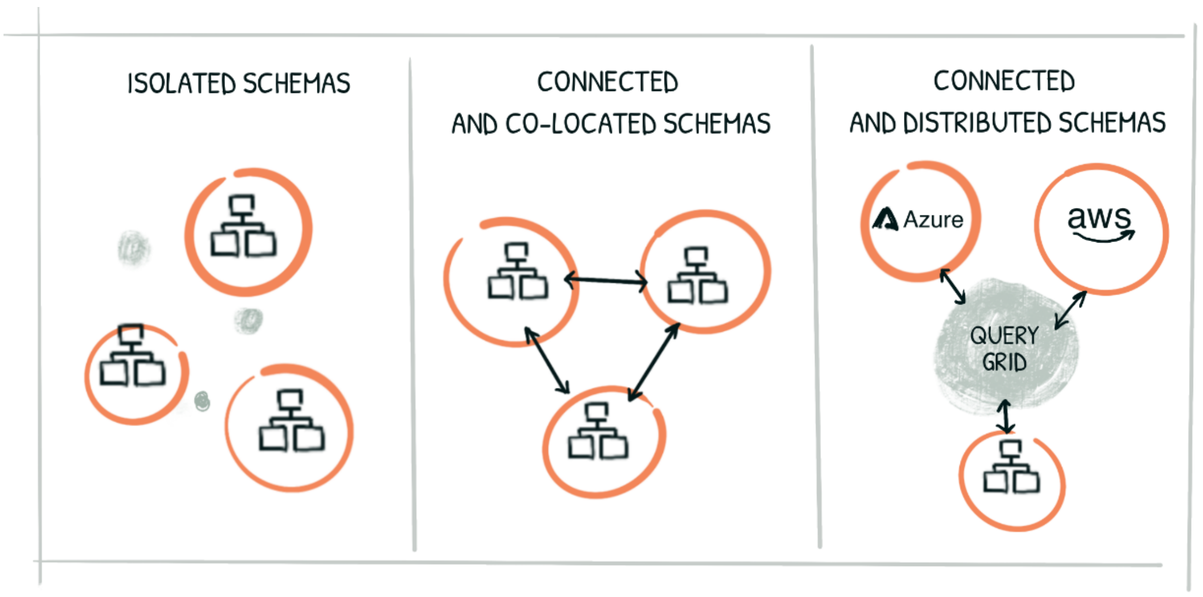
Federating the development of complex data products does not automatically imply the federation of their deployment. In fact, a spectrum of deployment options is available to organizations deploying data mesh solutions. Different strategies are associated with fundamentally different engineering trade-offs, so it is important that organizations frame these choices correctly and are intentional about their decisions.

In general terms, there are three different strategies for deploying schemas within a data mesh as defined by the vendors such as Teradata:

1. Isolation
2. Co-location
3. Connection

These are not mutually exclusive, and many real-world implementations use a combination of these approaches, especially in large data estates.

If environments such as Teradata Vantage are used, the play is between deployment of centralized image(s) to host collocated domains, host individual domains or use Vantage as a data platform gateway to virtualized data from other platforms.

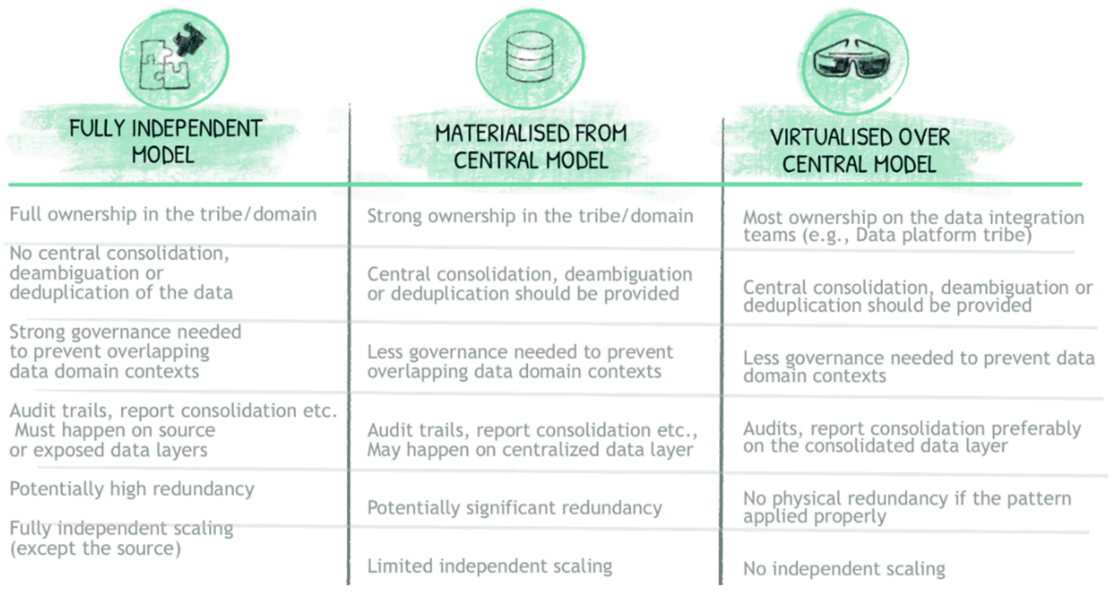

ISOLATED SCHEMAS
CONNECTED AND CO-LOCATED SCHEMAS
CONNECTED AND DISTRIBUTED SCHEMAS
Azure
aws
QUERY GRID

3.2 Ways of sourcing/building the data mesh

Among several patterns to build data mesh three deserve a closer look at them:

- Fully independent model in which each domain is sourced independently
- Centralized sourcing with materialization of the domain
- Centralized sourcing with virtualization of the domain

Of course, variations mixing the model can and certainly will be built and used.



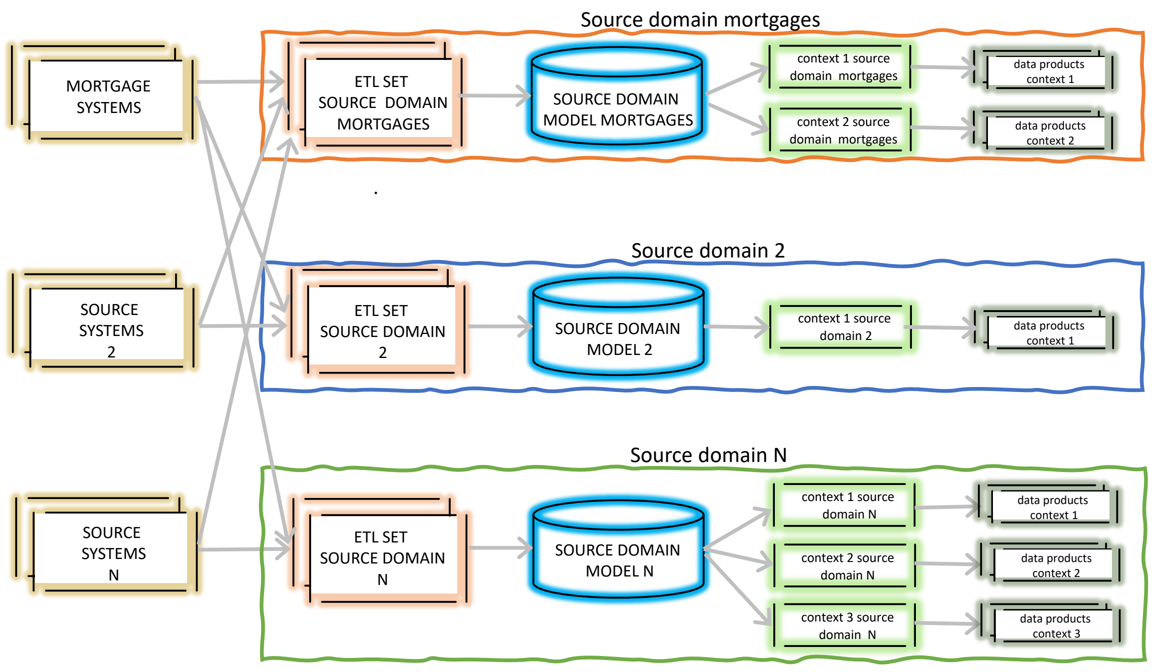
FULLY INDEPENDENT MODEL	MATERIALISED FROM CENTRAL MODEL	VIRTUALISED OVER CENTRAL MODEL
Full ownership in the tribe/domain	Strong ownership in the tribe/domain	Most ownership on the data integration teams (e.g., Data platform tribe)
No central consolidation, deambiguation or deduplication of the data	Central consolidation, deambiguation or deduplication should be provided	Central consolidation, deambiguation or deduplication should be provided
Strong governance needed to prevent overlapping data domain contexts	Less governance needed to prevent overlapping data domain contexts	Less governance needed to prevent data domain contexts
Audit trails, report consolidation etc. Must happen on source or exposed data layers	Audit trails, report consolidation etc., May happen on centralized data layer	Audits, report consolidation preferably on the consolidated data layer
Potentially high redundancy	Potentially significant redundancy	No physical redundancy if the pattern applied properly
Fully independent scaling (except the source)	Limited independent scaling	No independent scaling

Of course, the models can be combined to maximize the benefits of data mesh. The condition for successful cohabitation of the patterns is consistent federated governance spanned on entire mesh.

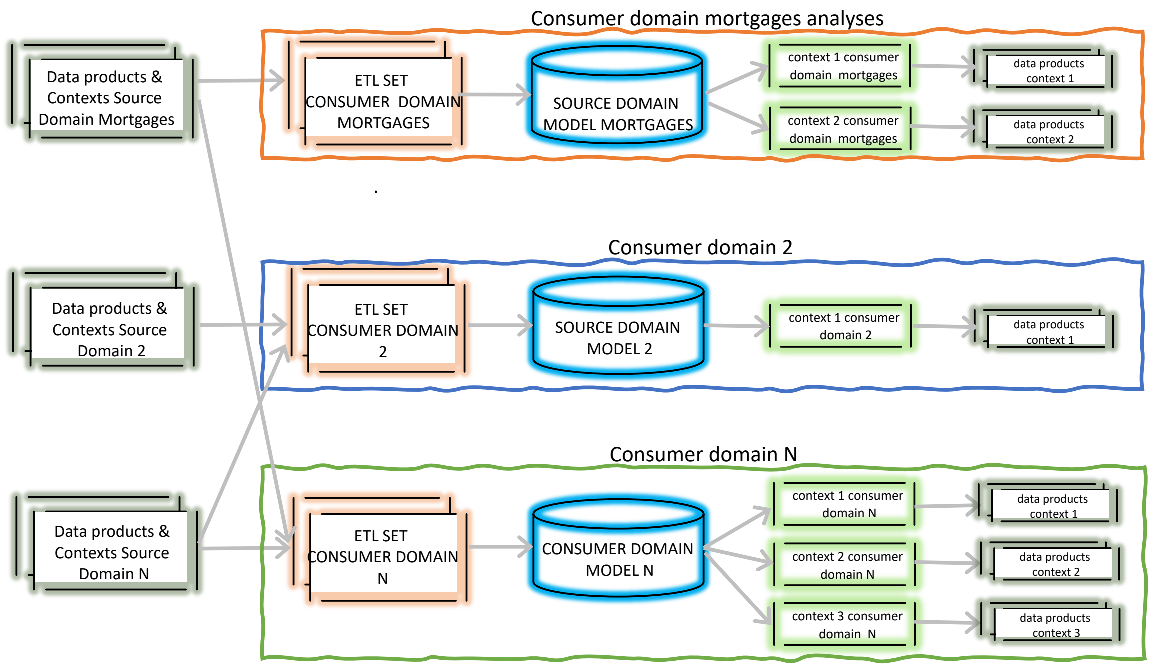
The Teradata Vantage platform can be used as a technology to serve any data architecture and is not only capable of implementing each of these 3 models, but also works in its various parts providing sources, serve as centralized repository as well as implement domain repositories delivering data within its bounded context.

3.3 Fully independent model

Each of the domains is logically separated from its sourcing to consumption. Still the interoperability is satisfied by coherent data delivery agreements.



The source domain exposes data products to the downstream domains in the form of APIs, databases and streams that are immutable for everything outside of the domain. The source domain can produce other data products too.


Consumer domain mortgages analyses
Data products & Contexts Source Domain Mortgages
ETL SET CONSUMER DOMAIN MORTGAGES
SOURCE DOMAIN MODEL MORTGAGES
context 1 consumer domain mortgages
context 2 consumer domain mortgages
data products context 1
data products context 2
Consumer domain 2
Data products & Contexts Source Domain 2
ETL SET CONSUMER DOMAIN 2
SOURCE DOMAIN MODEL 2
context 1 consumer domain 2
data products context 1
Consumer domain N
Data products & Contexts Source Domain N
ETL SET CONSUMER DOMAIN N
CONSUMER DOMAIN MODEL N
context 1 consumer domain N
context 2 consumer domain N
context 3 consumer domain N
data products context 1
data products context 2
data products context 3

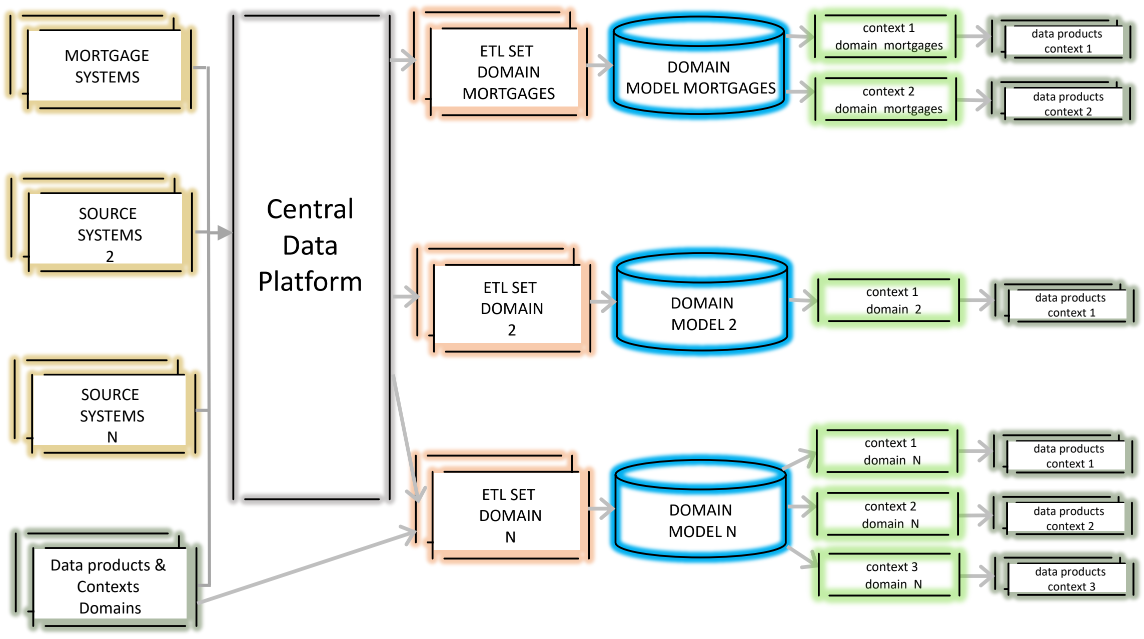
Source domains and consuming domains are well aligned, for example, a mortgage system source domain and mortgage analyses consuming domain. There may be two different teams but they're aligned on business processes, with one team providing data in a consistent manner, one team consuming it.

The integration domain is a special kind of consuming domain. It is less aligned with just one source domain but is generally sourced from more source domains then consumer domains are.

3.4 Sourcing from the central data platform with materialization of the domain

Domains are sourced from the central data platform and domains are typically materialized. Less command is left in the hands of data-product team allocated in the domain. The central domain platform uses the data lake (nowadays mostly stored on native objects stores), the lake house and data warehouse patterns, whichever is fit for purpose to store data. The data in it will be aligned on a domain basis, for example in dedicated buckets/accounts/schemas. Domains get a slice of the technology stack.

The concepts of source-, integration- and consumer domains are not strong here; however it is still possible for domains to consume data products directly from other domains. Integration domains could exist, but most of the time integration is done on the central data platform. Creating a separate integration domain must be done using good common sense.

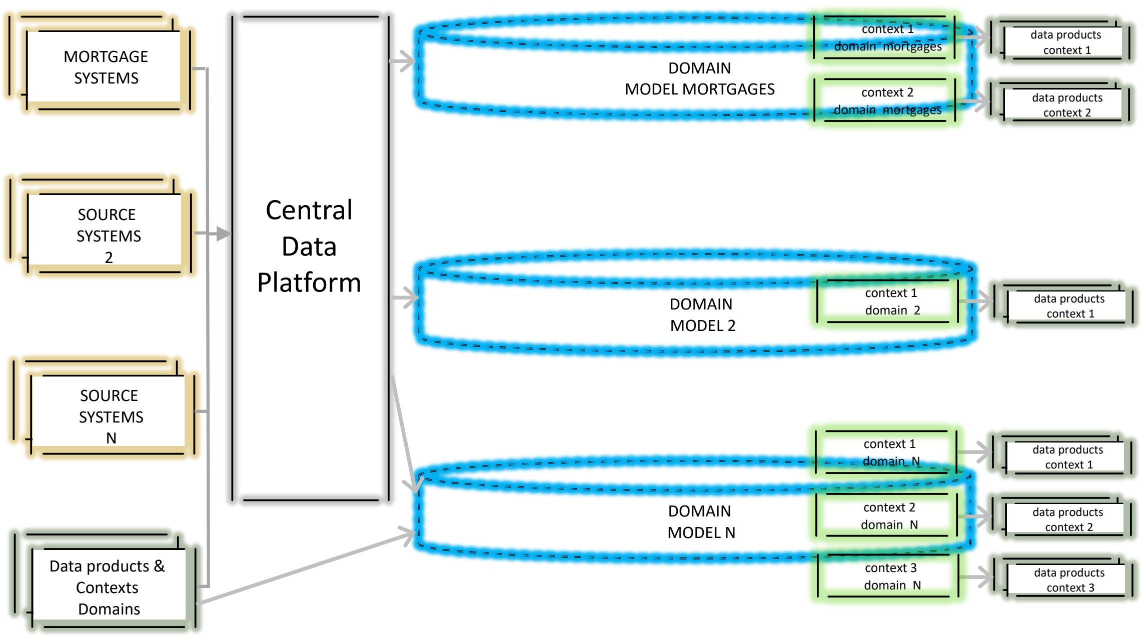


3.5 Sourcing from the central data platform with virtualization of the domain

Domains are sourced from the central data store mainly on virtual basis. Amount of command and control in data-product team's hands is similar to the previous pattern. Strong understanding of the centralized repository is needed. What goes for sourcing from the central data platform with materialization of the domain goes for this way of sourcing building a data mesh too. The central domain platform uses the data lake (nowadays mostly stored on native objects stores), the lake house and data warehouse patterns, whichever is fit for purpose to store data. The data in it will be aligned on a domain basis, for example in dedicated buckets/accounts/schemas. Domains get a slice of the technology stack.

Virtualization tooling and/or database views support the domain model and its contexts.

The concepts of source-, integration- and consumer domains are not strong here; however it is still possible for domains to consume data products directly from other domains. Integration domains could exist, but most of the time integration is done on the central data platform. Creating a separate integration domain must be done using good common sense.



4. How to organize the service landscape to implement data mesh

Data mesh is always implemented using domain driven design. Division of the business areas into domains, closing common functional and model chunks by bounded contexts is the way to enable data mesh. Microservices and modern integration/service enablement techniques usually follow, but it is DDD that lays the foundation for construction of successful data mesh implementation.

There are several techniques for subdividing business capabilities into the domains as well as construction of thereof (are addressed in the article devoted to modern architectures delivery), however it is easy to single out a stereotype of the data domain.

4.1 General organization of architecture layers for data mesh

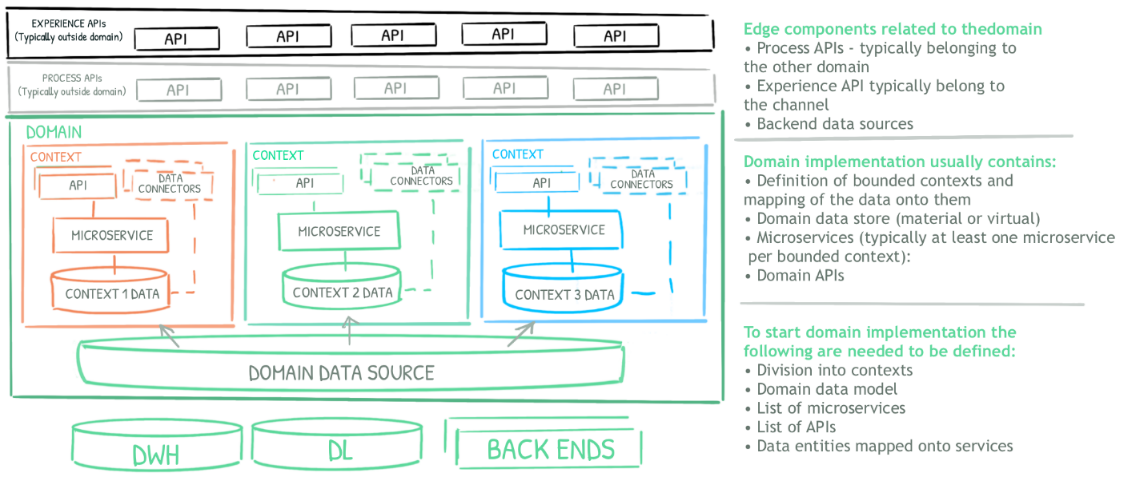
The data domain usually consists of several layers:

- Deep sources or operational systems that provide the transactional/operational data
- Domain data repositories – typically responsible for so called liberalization of the data
- (Micro)services to provide the main functionalities enabling and operating the data
- Data service layer serving data as products.

The domain can contain one or more bounded contexts. The contexts usually don't but might overlap with other domains and subdomains.

In the properly constructed data mesh (and DDD governed at all) solution is the rule that only master domain can change its data. Subordinate domains cannot, they can only read the master domains data. Whenever subordinate domain is a master of some data it is the only authority to change them.

Also, there is a substantial difference between the notions of data and business domains. The first one is a provider and change originator for the data and data products and can potentially serve business domains as data product provider. The latter one is usually a more behavioral and transactional concepts exposing services for operations on the business processes and state of business objects managed by organization. Keeping that in mind one can notice the patterns/styles in fact complement each other.



Edge components related to thedomain
- Process APIs - typically belonging to the other domain
- Experience API typically belong to the channel
- Backend data sources

Domain implementation usually contains:
- Definition of bounded contexts and mapping of the data onto them
- Domain data store (material or virtual)
- Microservices (typically at least one microservice per bounded context):
- Domain APIs

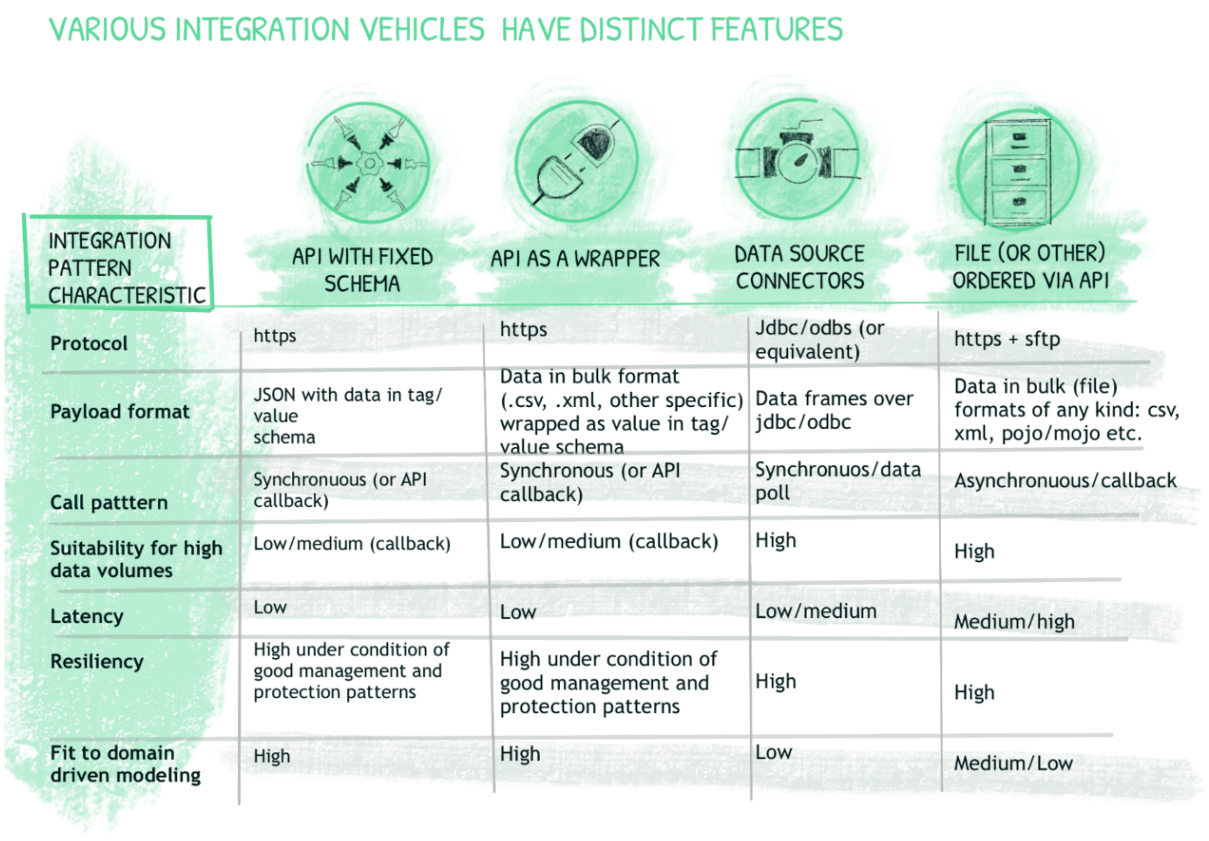
To start domain implementation the following are needed to be defined:
- Division into contexts
- Domain data model
- List of microservices
- List of APIs
- Data entities mapped onto services

It takes a very business and technology aware team to properly decompose and design from business services to implementation to properly distributed data mesh. Once the culture and routine are there, however, adding the domains or decomposing monoliths into the domains gets easier and easier.

4.2 Types of APIs used for data mesh

As stated in the earlier part of the document API driven connectivity is one of the most logical choices for exposing data products. We consider a couple popular ways to technically expose the data—see below.

The date volume exchanged differentiates the use of particular APIs over the others. Acquisition of moderate data portions makes direct APIs advantageous while querying mass data will always favor direct connections to the data or bulk file extracts. Those, however, benefit from being initiated and controlled by API calls.

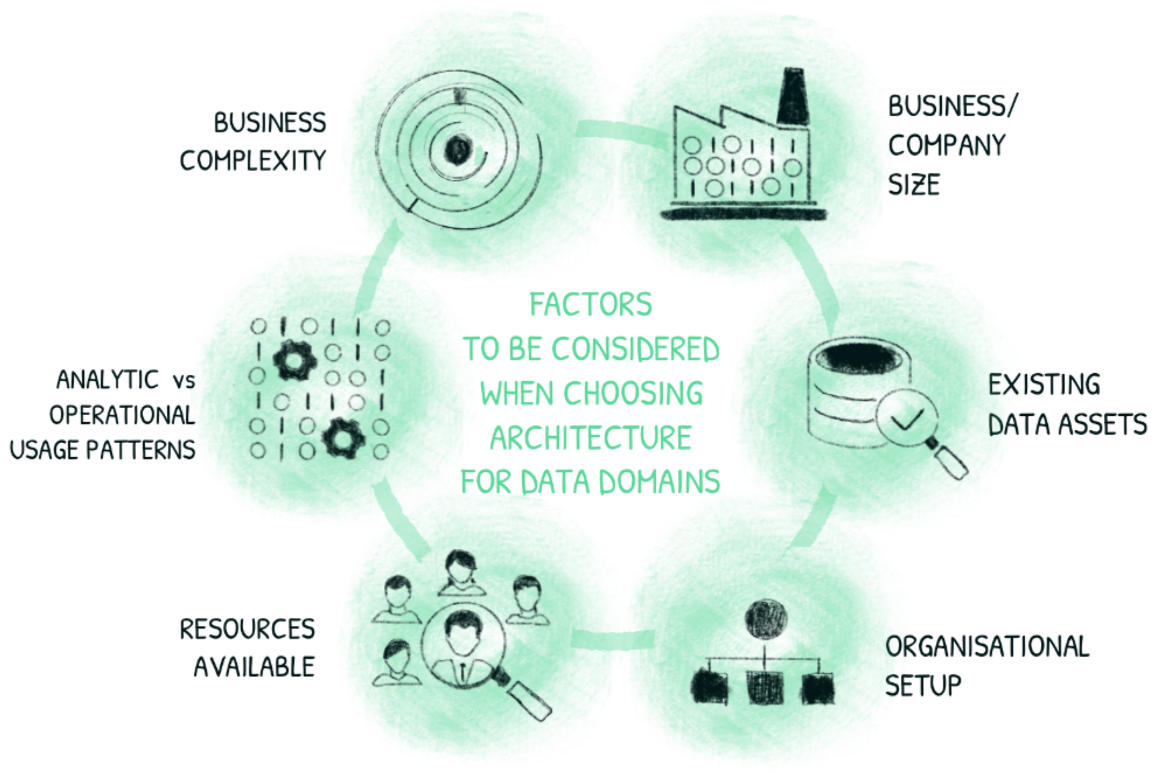


VARIOUS INTEGRATION VEHICLES HAVE DISTINCT FEATURES

INTEGRATION PATTERN CHARACTERISTIC	API WITH FIXED SCHEMA	API AS A WRAPPER	DATA SOURCE CONNECTORS	FILE (OR OTHER) ORDERED VIA API
Protocol	https	https	Jdbc/odbs (or equivalent)	https + sftp
Payload format	JSON with data in tag/ value schema	Data in bulk format (.csv, .xml, other specific) wrapped as value in tag/ value schema	Data frames over jdbc/odbc	Data in bulk (file) formats of any kind: csv, xml, pojo/mojo etc.
Call patttern	Synchronuous (or API callback)	Synchronous (or API callback)	Synchronuos/data poll	Asynchronuous/callback
Suitability for high data volumes	Low/medium (callback)	Low/medium (callback)	High	High
Latency	Low	Low	Low/medium	Medium/high
Resiliency	High under condition of good management and protection patterns	High under condition of good management and protection patterns	High	High
Fit to domain driven modeling	High	High	Low	Medium/Low

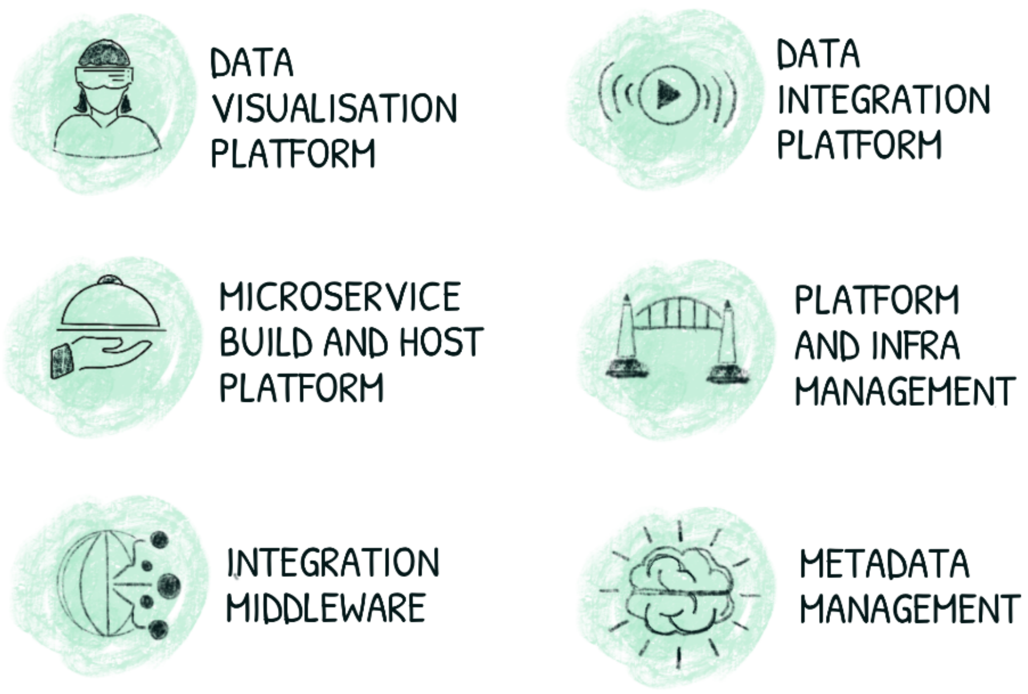
5. Ways of building data mesh depend on the scale and resources in hand

Whenever deciding to build a data mesh the most pragmatic approach needs to be taken. There is a list of factors that need to be considered ahead of deciding how to implement:


FACTORS TO BE CONSIDERED WHEN CHOOSING ARCHITECTURE FOR DATA DOMAINS
BUSINESS COMPLEXITY
BUSINESS/ COMPANY SIZE
EXISTING DATA ASSETS
ORGANISATIONAL SETUP
RESOURCES AVAILABLE
ANALYTIC vs OPERATIONAL USAGE PATTERNS

When the company IT department has sufficient scale, all the layers can be built or set up in almost every possible way, including building from the scratch. Most companies are, however, constrained by the budget, resources, and learning curve needed to master the skills to prepare all the layers.

In case supporting ready platforms need to be utilized to limit the overhead needed to build the solution, the following areas of considerations should be addressed:


DATA VISUALISATION PLATFORM
DATA INTEGRATION PLATFORM
MICROSERVICE BUILD AND HOST PLATFORM
PLATFORM AND INFRA MANAGEMENT
INTEGRATION MIDDLEWARE
METADATA MANAGEMENT

There are ready platforms for each of the areas mentioned. They in fact limit possibilities somehow but provide a quick starting point and platform that can be directly used to build and host the components of data mesh.

The next big step that is likely to emerge or even already on the horizon is data mesh as a service offering from significant players. Many can be mentioned, but the Teradata Vantage platform has native connections to sources like native object storage and using Teradata QueryGrid. This can be extended to Apache Hive and Apache Spark, Oracle and Google BigQuery. The Starburst Presto connector makes it possible to further extend the connection to a myriad of data engines through Starburst Presto. This makes the platform a ready-to-use solution offering a data mesh skeleton to be integrated, filled with the data and provisioned on the cloud for the use of data clients.

6. Dos and don'ts – when (not) to implement data mesh architecture

As with every pattern, there are some limitations and caveats that need to be taken into consideration whenever used. Below is a list of major points to consider:

1. Do not treat data mesh as a golden hammer to solve all the problems with data
2. Do not seek to replace of your data warehouse or data lake with data mesh if they already properly fulfill their function—think about data mesh as an evolutionary step
3. If a data mesh style is planned for implementation, changing or evolving technology and data models as well as the organization of federated governance, domain decomposition and strict domain ownership is important
4. Do not start with technology—technology is an enabler but objectives come from business definitions of the domains
5. Leverage modern service and deployment patterns such as cloud, data virtualization, CI/CD etc. to fully explore decomposed and additive models

7. Teradata as a provider of core technology to enable data mesh architecture

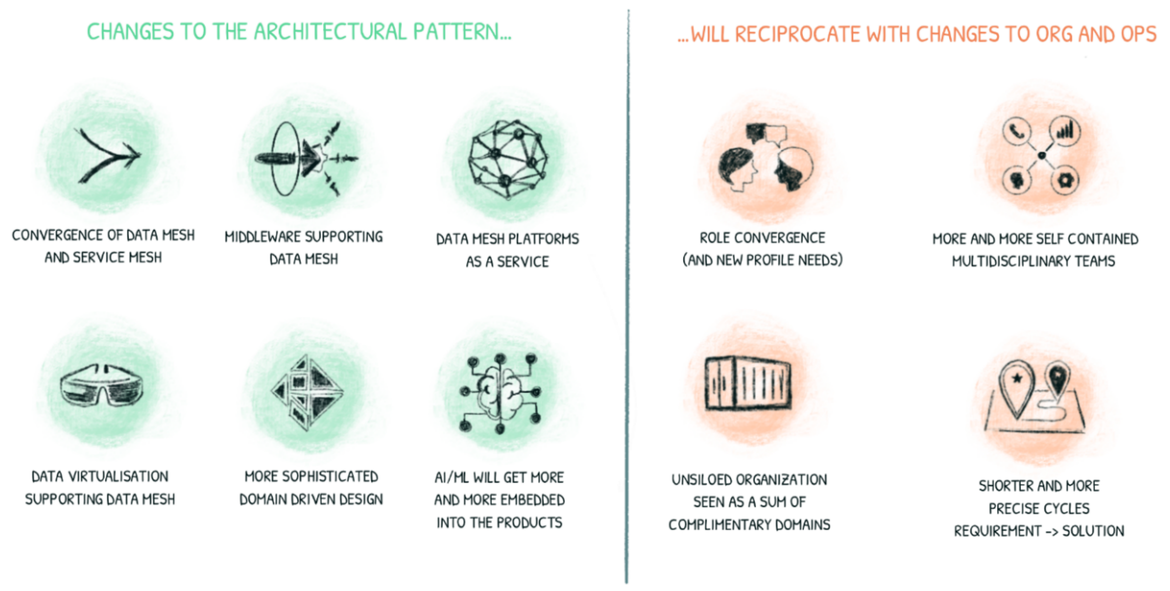
Teradata has been around for more than 40 years now providing unparallel capabilities in the processing of huge amounts of data. It is an original MPP design which stems from its shared nothing architecture that has resolved problems with storing, selecting and joining huge, large-scale data sets. Over the years lots of capabilities have been developed around Teradata:

- Efficient and scalable data integration tools (via Teradata Parallel Transporter, part of Teradata's Tools and Utilities that come standard with Teradata Vantage)
- Best in class workload management solution (called TASM) that allows protecting workloads of various types (tactical, strategical, analytical, etc) and making it possible to meet your SLA's
- Scalable on-prem, hybrid- and multi cloud offering (Vantage) that allows to scale data processing solutions easily to adapt to varying workload (storage and computation intensity). Teradata has a 'cloud first' strategy
- Heterogenous data integration platform allowing to combine various technologies into one data ecosystem managed from the Teradata Vantage platform
- Separated storage and compute which makes Teradata Vantage extremely suitable to adapt cloud capabilities, like scalability, elasticity, agility, flexibility, efficiency in resource usage, etc.
- Teradata connectors like amongst others the Teradata Kafka connector which makes it possible to use Kafka to stream data into Vantage
- Teradata QueryGrid, which makes it possible to connect Vantage to a myriad of data engines and understand the statistics in those data engines. This way Vantage can work together with those engines to determine the best path to your data and thus limiting resource usage.
- Reading and writing to Native Object Stores (NOS) like AWS S3, Azure Blob and ADLS, Google cloud storage or on-prem object stores that use the AWS S3 API. Making it possible to use this cheap storage for archiving or event store purposes
- A "Bring Your Own Model," making it possible to score your models in the database, bringing the processing to the data instead of bringing the data to the processing, which can be very expensive in the cloud
- Running R, Python and Java in the database, again bringing the processing to the data. No need to move your data to your R, Python or Java clients anymore

- A robust set of Teradata Vantage ecosystem management tools, including back and recovery, sandboxing, moving data, Vantage management and business continuity management
- Teradata Vantage supports the data mesh concept in all three schema deployment strategies (Isolation, Co-location and Connection)

8. Outlook for the future

The future is now, as numerous communities think and work on refinement and preparation of the new patterns. Below, the authors allow themselves to speculate on the future evolution of the data mesh style. Some of the anticipated novelties are applicable for the other patterns as well and as such will probably be used widely.


CHANGES TO THE ARCHITECTURAL PATTERN...
CONVERGENCE OF DATA MESH AND SERVICE MESH
MIDDLEWARE SUPPORTING DATA MESH
DATA MESH PLATFORMS AS A SERVICE
DATA VIRTUALISATION SUPPORTING DATA MESH
MORE SOPHISTICATED DOMAIN DRIVEN DESIGN
AI/ML WILL GET MORE AND MORE EMBEDDED INTO THE PRODUCTS
...WILL RECIPROCATE WITH CHANGES TO ORG AND OPS
ROLE CONVERGENCE (AND NEW PROFILE NEEDS)
MORE AND MORE SELF CONTAINED MULTIDISCIPLINARY TEAMS
UNSILOED ORGANIZATION SEEN AS A SUM OF COMPLIMENTARY DOMAINS
SHORTER AND MORE PRECISE CYCLES REQUIREMENT -> SOLUTION

The main predictions concern architecture/technology and organization/operations. The changes will reciprocate as business changes create new requirements while advancements in technology and architecture enable business and operations advancements.

8.1 Technology

8.1.1 Data mesh and service mesh will converge

Both patterns will converge as data mesh will more frequently support transactional and operational activities, while analytic data will more often be enabled via services defined as APIs or similar.

8.1.2 Middleware will support data mesh

As mentioned in the previous chapters, the enablement teams will be fuller and fuller supported with middleware dedicated or including data mesh pattern. Service, integrations as well as provisioning of data domains will become more and more codeless.

8.1.3 Data mesh platforms as a service

Hyperscalers and traditional analytic repositories providers will start building and advertising data mesh platforms offered as a service. A similar turn of events led to productized data lakes or lake houses.

8.1.4 Data virtualization as enabler for data mesh

Data mesh as a pattern is a natural candidate to be enabled by data virtualization tooling—for example Teradata QueryGrid. They typically allow for rapid microservice-style deployment of data domains or domains sets. It seems logical this trend will prevail.

8.1.5 Domain Driven Design sophistication

Domain driven design is a great enabler for data mesh. New techniques of decomposing business into domains as well as designing the domain will gain ground, and DDD will be automatically linked with Data Mesh pattern

8.1.6 AI/ML will get embedded into data mesh

Prediction relevant for both data mesh and lake house or similar patterns assumes AI and ML will get embedded into the data platform to properly match and translate between business and technical semantics. Ultimately, it would enable an AI supported fetch of the results for the queries requested in business language. Also, organization of the data, usage of semantic graphs, self-organizing structures, and database management will find its application.

8.2 Organization and operations

Organizational and operational changes will enable development of architectural standards and patterns. The process of agile methodologies driven convergence of the roles and skillsets will continue. Analysts and data scientists will acquire technology proficiency while IT oriented individuals will be gaining more awareness and excellence in usage of requirements and business process analysis. The teams are and will continue becoming multidisciplinary, typically organized in tribes or similar structures. Cooperation between topic/domain-oriented groups should remove siloes in organizations, which is the main prerequisite for federated governance over the data and the domains.

9. Conclusions

Data Mesh is a promising but already widely adopted pattern that allows overcoming some significant shortages of patterns used so far. It moves the development closer to the owners and users of the data while retaining their overall business alignment via federated, preferably computational, governance.

The market adopts the data mesh concept, while the vendors of IT solutions and service providers develop data mesh as a service or product.

Data Mesh is a style likely to coexist and integrate with patterns used so far such as data lake or lake house. It is also typically a driver of organizational and operational changes in large organizations, leading to more efficient handling and processing of data.

Teradata, as a company with 40 years of experience, was able to build a concept and products for implementing large scale data mesh. Teradata Vantage and Teradata QueryGrid allow to achieve every data mesh flavor.

BCG and Teradata partner on data mesh (and other patterns) projects shaping and delivering to support data intensive business organizations.

10. About Teradata

Teradata is the connected multi-cloud data platform company. Our enterprise analytics solve business challenges from start to scale. Only Teradata gives you the flexibility to handle the massive and mixed data workloads of the future, today.

The Teradata Vantage architecture is cloud native, delivered as-a-service, and built on an open ecosystem. These design features make Vantage the ideal platform to optimize price performance in a multi-cloud environment. Learn more at Teradata.com.

11. About the authors

Robbert Naastepad

Robbert started his career as a Cobol programmer on the IBM MVS operating system using the IMS-DB/DC database management system. After using dBase II/III/III+ and Foxpro got introduced to the Oracle 6.1.7 RDBMS in 1994.

From that moment, he got more and more involved in executive information systems, data warehousing and business intelligence. Robbert has evolved from developer and analyst in several projects, to technical architect at Oracle into an enterprise data, BI and analytics architect at Teradata. He has been a Teradata team member since January 2017, as he felt Teradata had the right business strategy and products to support that strategy to be a leading company in data management for business intelligence and analytics. Now Robbert supports customers and partners of Teradata developing and implementing data architectures using the Teradata Vantage data platform.

Jakub Fila

Jakub started as an aerospace engineer in the turbine engine industry and then the nuclear industry. He is a graduate of the Aerospace Engineering and Physics faculties where he also learned software engineering and started his interest in data processing and parallel programming.

He has been working as an IT architect on various levels of seniority for companies like Accenture, IBM or Teradata. Currently, Jakub is a Principal at BCG Platinion, helping clients to make strategic decisions and implementing right technologies to fulfill the strategy. He specializes in integration architecture, enterprise architecture, massive data processing and software engineering. He is an enthusiast of MPP platforms and efficient parallel programming techniques and leads the Application Architecture chapter in BCG EMESA region. Privately, Jakub is keen on aerospace and triathlon.

Jens Mueller

Jens is with his whole heart an engineer. He started coding J2EE solutions in financial industries and evolved into a domain/enterprise architect over the years via performance measurement and optimization cases which helped build a thorough understanding of traditional relational DBMS and their inner workings (Oracle and DB2).

Moving on to strategic IT consulting in 2006, he explored all aspects of IT management. The path to data platforms was laid out pilot cases on SAS around 2010 and then from 2012 on taking over the role of head of design authority capital markets in a large German bank. There he substantially helped setting up a data hub based on data lake and streaming technologies that was later moved to a cloud native setup. After re-joining BCG Platinion in 2019, Jens took over the data architecture chapter to extend our competences further and work on (cloud) data platform cases in financial industries.

Vincent von Hof

Vincent is a senior IT architect at BCG and engineer by his background and in his heart. Vincent is one key members of BCG Data Chapter.

Vincent started his career as a software developer focused on Java and C#, solving data challenges in the middleware and backend for large scale data processing and creating data ontologies for the German government, as well as working on solving data challenges on the opposite scale in constrained environments on Android, where he led a development team at a startup for multiple years. He holds Ph.D. in software engineering majoring in automated test case generation. With 10+ years of development experience he joined BCG as an architect.

His main interests include massive data processing, software engineering, test automation and cloud architecture. Vincent typically leads large data management projects delivering content, architecture concepts, implementing and aligning business with IT.