



Theo cách của bạn

DATA MESH ARCHITECTURE

MINI-PROJECT REPORT - TOPIC 8

DATA ENGINEERING SPECIALIZATION

RESIDENT: Tạ Ngọc Minh
BSc. student in Data Science & AI @ HUST

SUPERVISOR: Mr. Nguyễn Chí Thanh

PREFACE

Dear the board of directors of Viettel Group, the mentors, and all of my friends,

This document is a part of my research mini-project as a data engineer intern for the first period of Viettel Digital Talent 2023 Program. The completion of this research report does not mean the end of my duties as learners. In fact, this report is a first step towards consistently studying what I have reviewed and written here.

Data has grown up so quickly recently. The term 'big data' has become familiar with every tech-lovers is the clearest evidence of the explosion of data. This leads to many requirements of upgrading the current system and management, along with finding new architecture to store, process and analyze data. Once the central data lake has been overloaded, we need to design a decentralized architecture. This report is written under the research about the Data Mesh architecture, a new design that was first defined by Zhamak Dehghani in 2019.

To write this document, I would like to give many thanks to Viettel Group, and Viettel Digital Talent program for giving me chance to do my research. I want to send a big thanks to Mr. Nguyễn Chí Thành, and all the mentors of the first period of the program, who gave me many useful lectures and guidance. Also, I want to say thanks to professor Huỳnh Thị Thành Bình, professor Đỗ Tuấn Anh for their initial supports, and many other professors at the International Research Center for AI and the School of Information and Communication Technology, Hanoi University of Science and Technology about their public documents about data engineering. Last, I would like to say thanks to all of my friends, who always read my documents, comment about them and support me whenever I need.

Because this document is written in such a short time and some of the concepts are so new and hard to understand for a fresher like me, it is so hard to make sure it is completely correct, including the grammar errors and formatting errors in L^AT_EX. I would love to receive your contributions via ngocminhta.nmt@gmail.com.

Thank you for reading this report and looking forward to getting many comments from you.

Ngoc-Minh Ta
Viettel Digital Talent Residency 2023

CONTENTS

Preface	iii
List of Figures	vii
List of Tables	viii
1 Problems and Initial Approach	1
1.1 Real-life Problem	1
1.1.1 A problem from history	1
1.1.2 Demands from recent developments	2
1.2 Initial approach	2
1.2.1 What is Data Mesh?	2
1.2.2 What will changes after data mesh?	2
2 Data Mesh Architecture Design	5
2.1 Four Fundamental Principles of Data Mesh	5
2.1.1 Principle of Domain Ownership	5
2.1.2 Principle of Data as a Product	7
2.1.3 Principle of the Self-Serve Data Platform	8
2.1.4 Principle of Federated Computational Governance	8
2.2 Why we need Data Mesh?	10
2.3 Data Mesh Architecture Design	12
2.3.1 The Logical Architecture	12
2.3.2 The Multiplane Data Platform Architecture	15
3 Data Product Design & Implementation	21
3.1 Design a Data Product by Affordances	21

CONTENTS

3.2 Consuming, Transforming & Serving Data	22
3.2.1 Serve data	22
3.2.2 Consume Data	23
3.2.3 Transform Data	24
3.3 Discovering, Understanding & Composing Data	25
3.3.1 Discover, Understand, Trust, and Explore	25
3.3.2 Compose Data	26
3.4 Managing, Governing & Observing Data	27
3.4.1 Manage the Life Cycle	27
3.4.2 Govern Data	27
3.4.3 Observe, Debug, and Audit	28
4 Data Mesh in use	30
4.1 Data Mesh in combination with Data Lakehouse	30
4.2 Frameworks and technologies for Data Mesh	30
4.3 Case study and Demo	30
References	a

LIST OF FIGURES

1.1	Central data team in a data-driven business	1
1.2	Cultural shift after implementing data mesh	3
1.3	Data mesh dimensions of organizational changes	4
2.1	Decomposing the analytical data ownership and architecture, aligned with business domains, along with their data archetypes.	6
2.2	The baseline usability attributes of data products (DAUTNIVS)	7
2.3	Example of data mesh governance operating model	9
2.4	Macro drivers for the creation of data mesh	10
2.5	Data Mesh Logical Architecture [1]	12
2.6	The logical architectural components of embedded computational policies . .	14
2.7	Multiplane self-serve platform and platform users	15
2.8	High-level example of data product development journey using the platform	16
2.9	Example of data infrastructure plane to support data product delivery . . .	20
2.10	Example of data infrastructure plane to support data product delivery . . .	20
3.1	Data product's high-level components to design serve data	23
3.2	Data product's design to consume data	24
3.3	High-level components to design transformation for data products	25
3.4	A data product sidecar provides a unified model of discoverability for all data products	26
3.5	Data product's high-level components to design data composability	27
3.6	High-level interactions to manage the life cycle of a data product	28
3.7	High-level design of embedded policies as code	29

LIST OF TABLES

2.1	Summary of after the inflection point with data mesh	10
2.2	Data mesh logical architectural components and their maturity	12
2.3	Example interfaces provided by the platform planes	16
2.4	Data platform interfaces that support data product experience plane APIs . .	18
3.1	Data product affordances	21
3.2	High-level data product components to serve data	23
3.3	High-level components to design consume data	23
3.4	High-level components to transform data for a data product	25
3.5	High-level components to transform data for a data product	27

Chapter 1

PROBLEMS AND INITIAL APPROACH

1.1 REAL-LIFE PROBLEM

1.1.1 A problem from history

Many organizations have invested in building a central data lake. To make data-driven business, there is a central data administration team take the responsibility for this.

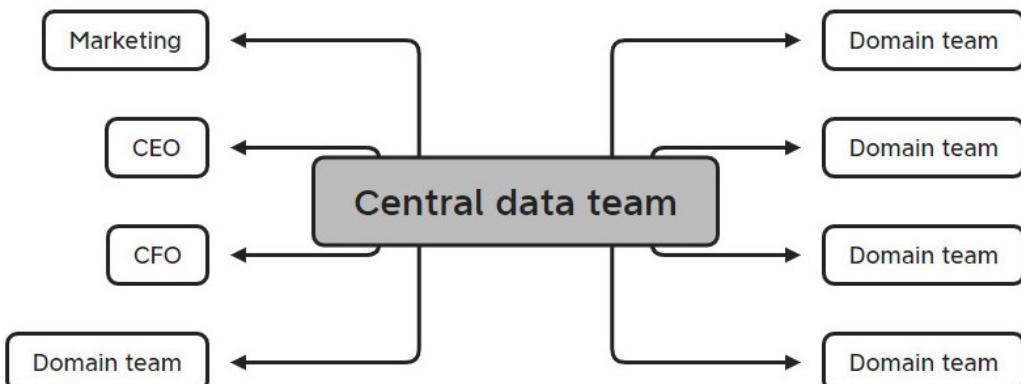


Figure 1.1: Central data team in a data-driven business

This model works quite well at the first time, however, after a while, they noticed that **the team often became bottleneck**. The team were overloaded with thousands of tasks and questions from multiple teams, along with the requests of time and accuracy. This lead to a massive problem since the competitiveness of business depends much on the speed of analysis. For example, when we create a landing page for a new product, how does this change the influence clicking and buying rate?

But, why the response of the central data team is so slow and struggle? Actually, once the operational database changes, the team has to spend much of their time fixing the broken data pipelines. The little remaining time is insufficient for them to discover and understanding all the necessary domain data **for each question**. Getting the required domain expertise is a daunting task. [2, 3]

In contrast, some firms employ domain-driven design strategies that include independent

1.2. INITIAL APPROACH

domain teams and a decentralized micro-service architecture to relieve burden on the central data team. These teams exclusively control and understand their domain, including the company's information demands. The issue hasn't entirely been resolved, though. The domain teams must contact the overloaded central data team despite being aware of the key information needs and the domain, in order to obtain the essential data-driven insights. [2]

1.1.2 Demands from recent developments

Let's consider the scale-up of the software development over time. When one task grows bigger and bigger, we have to decentralize it into many smaller tasks, otherwise, all the organization will overload and lose control. Take into consideration what we have done for the scale-up of software development:

- Decentralize business into domains;
- Decentralize engineering into autonomous teams;
- Decentralize monolith into micro-services;
- Decentralize operations into DevOps teams.

Hence, the next thing we need to do is scaling up data analytics by decentralizing data lake into data mesh.[3]

The position between domain teams and the central data team gets worse as the organization eventually grows. Transferring data management responsibilities from the central data team to the domain teams can help. Domain-oriented decentralization for analytical data is the central notion of the data mesh concept. Similar to APIs in a micro-service design, a data mesh architecture allows domain teams to conduct cross-domain data analysis on their own and connects data.

1.2 INITIAL APPROACH

1.2.1 What is Data Mesh?

The term data mesh was first stated by Zhamak Dehghani in 2019 and is based on four fundamental principles, which will be discussed in depth later:

- Principle of Domain Ownership;
- Principle of Data as a Product;
- Principle of the Self-Serve Data Platform;
- Principle of Federated Computational Governance.

1.2.2 What will changes after data mesh?

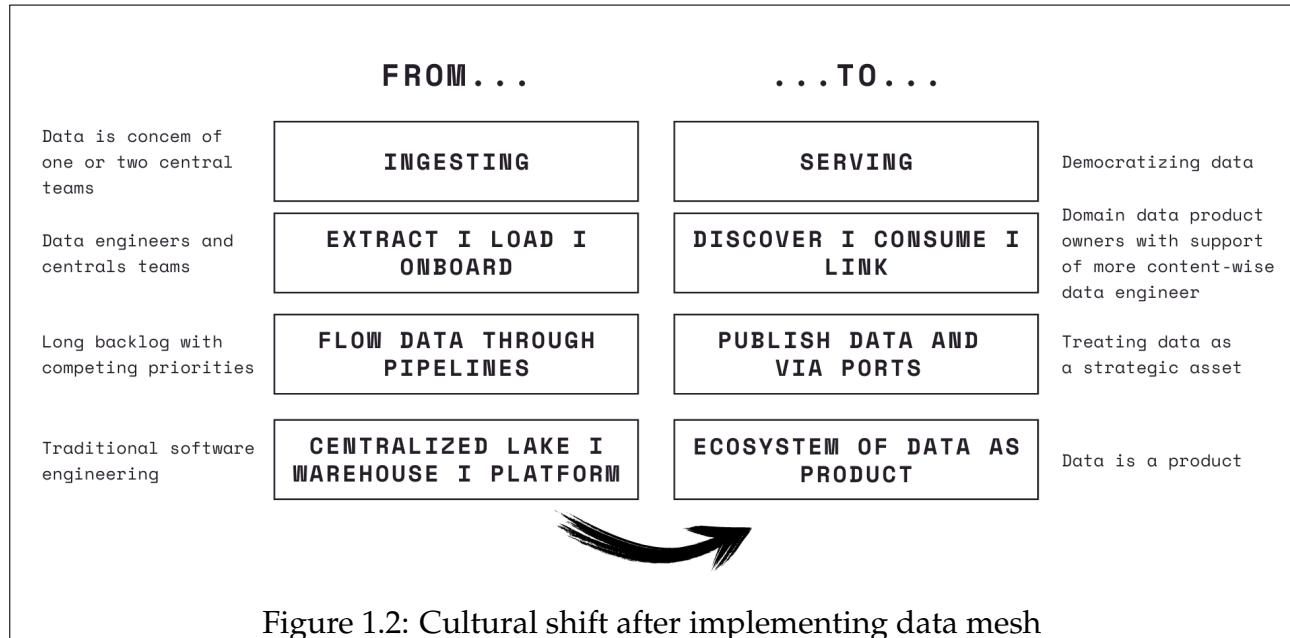
Data mesh is a fresh method for business intelligence, data analysis and management that is based on an innovative distributed architecture. Along with that, data lake and data warehouse do not disappear they just become nodes in the mesh. [1, 4]

Data mesh ensures organizations to continue to apply some data lake principles, such as making immutable data available for exploration or analytical use, and data lake tooling for

internal implementation of data products or as part of the shared data infrastructure. Data lake would not be the centerpiece anymore.

It is an implementation detail sub-serving the idea of domain data product as the first-class concern. The same applies to data warehouse in terms of business reporting and visualization. [4]

Implementing a data mesh is not a purely technical project that we can implement in isolation from the rest of the business. It's not some thing that we can just start with, and then it works from the first try, but we have to grow and develop with it. It also cause a cultural shift in the organization.



Also, data mesh calls for a fundamental shift in the assumptions, architecture, technical solutions, and social structure of our organizations, that means, how we manage, use and own analytical data. [5]

Data mesh can be used as part of an enterprise data strategy, articulating the target state of the enterprise architecture as well as an organizational operating model with an iterative execution plan.

In its most basic form, it can be characterized by four interacting principles, which will be discussed in depth in section 2.1.

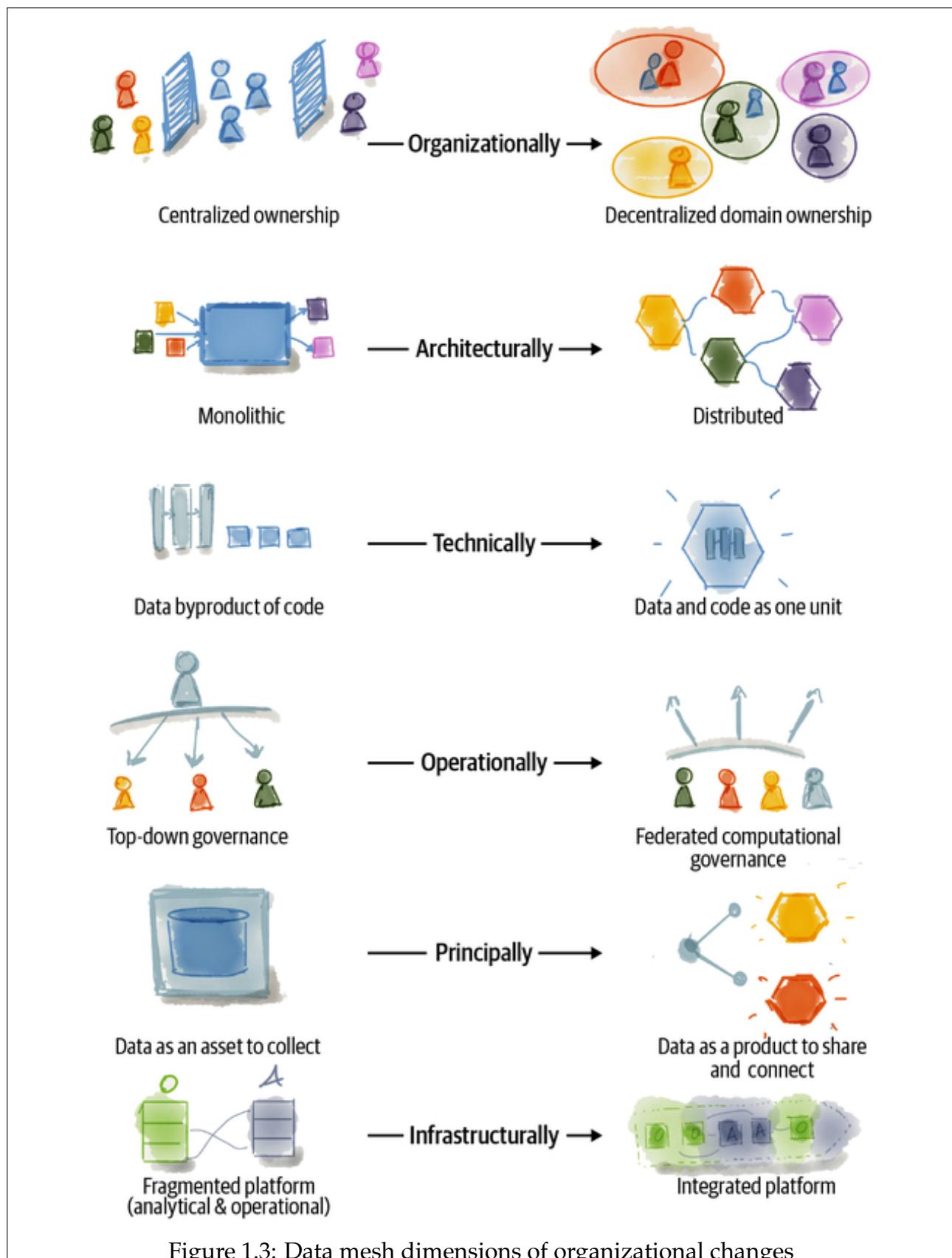


Figure 1.3: Data mesh dimensions of organizational changes

Chapter 2

DATA MESH ARCHITECTURE DESIGN

2.1 FOUR FUNDAMENTAL PRINCIPLES OF DATA MESH

Four simple principles can represent the logical architecture and operating model of data mesh. They are intended to move us closer to the goals of data mesh: increasing the value of data at scale, maintaining agility as an organization grows, and embracing change in a complex and turbulent business context.

2.1.1 Principle of Domain Ownership

Data mesh, at its core, is founded in decentralization and distribution of data responsibility to people who are closest to the data. This is to support a scale-out structure and continuous and rapid change cycles.

However, in contrast to traditional data structures with technological partition (e.g. data warehouse, data lake), data mesh follows *the seams of organizational units*. It follows the lines of division of responsibility aligned with the business using *domain-driven design (DDD) strategies*. Data mesh also gives the data sharing responsibility to each of the business domains, each domain is responsible for the data it is most familiar with.

DDD is an approach to decomposition of software design and team allocation, based on the seams of a business. It defines a *domain* as "a sphere of knowledge, influence or activity." DDD's Strategic Design embraces modeling based on multiple models each contextualized to a particular domain, called a bounded context¹. As Z. Dehghani's recommendation, data mesh adopts the boundary of bounded contexts to individual data products - data, its models, and its ownership.

Domain data ownership is the foundation of scale in a complex system like enterprises today. When we map the data mesh to an organization and its domains, we discover a few different archetypes of domain-oriented analytical data:

- Source-aligned domain data (native data product): Analytical data reflecting the business facts generated by the operational systems.

¹A bounded context is the delimited applicability of a particular model that gives team members a clear and shared understanding of what has to be consistent and what can develop independently. [6]

2.1. FOUR FUNDAMENTAL PRINCIPLES OF DATA MESH

- Aggregate domain data: Analytical data that is an aggregate of multiple upstream domains.
- Consumer-aligned (fit-for-purpose) domain data: Analytical data transformed to fit the needs of one or multiple specific use cases.

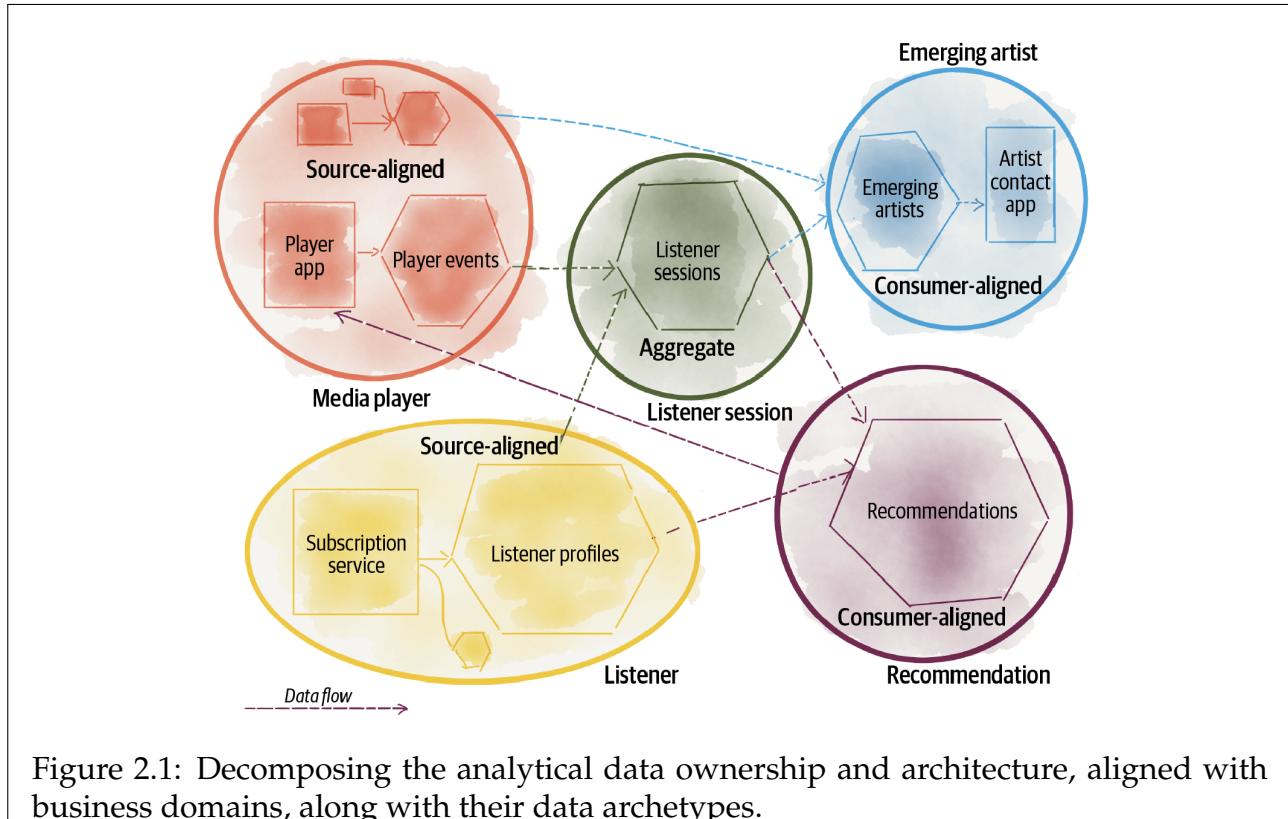


Figure 2.1: Decomposing the analytical data ownership and architecture, aligned with business domains, along with their data archetypes.

The shift toward domain-oriented data ownership leads to accepting and working with real-world messiness of data, particularly in high-speed and scaled environments:

- **Push Data Ownership Upstream:** Data can be consumable and useful right at the source analytical domain (source-aligned domain data). At a later point downstream, source-aligned domain data can be aggregated and transformed to create a new higher order insight (aggregate domain data or fit-for-purpose analytical data).
- **Define Multiple Connected Models:** We implement multiple models of polysemes². In data mesh, each domain can model its data according to its context, share this data and its models with others, and identify how one model can relate and map to others.
- Data mesh does not enforce the idea of searching for the single source of truth for each shared business concept. However, it places multiple practices in place that reduces the likelihood of multiple copies of out-of-date data. Long-term domain-oriented ownership with accountability to share discoverable, high-quality, and usable data in multiple modes for analysts and scientists.
- **Hide the Data Pipelines:** Data pipelines are first-class architectural concerns in traditional data architectures, composing more complicated data processing and transportation. A data pipeline in data mesh is just an internal implementation of the data domain that is handled within the domain.

²Polysemes are shared concepts across different domains. They point to the same entity, with domain-specific attributes. [7]

Domains are taking up extra data responsibilities with data mesh. To acquire agility and authenticity, responsibilities and efforts transfer from a centralized data team to domains. [7]

2.1.2 Principle of Data as a Product

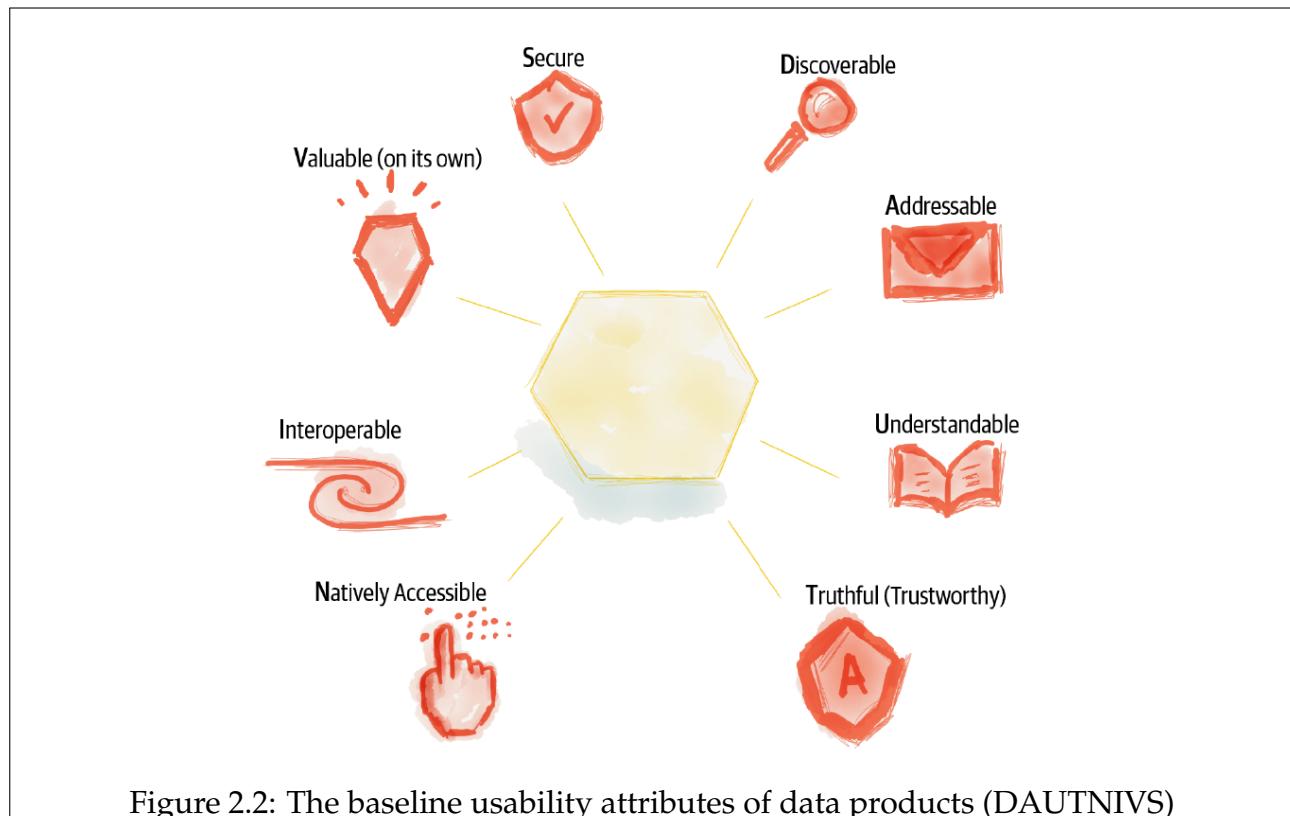


Figure 2.2: The baseline usability attributes of data products (DAUTNIVS)

The principle of data as a product is a response to the data siloing challenge that may arise from the distribution of data ownership to domains. It is also a shift in the data culture toward data accountability and data trust at the point of origin. The ultimate goal is to make data simply usable.

In approach to data product, there is a set of non-negotiable baseline characteristics in figure 2.2 that a data product incorporates to be considered useful. These characteristics apply to all data products, regardless of their domain or archetype. We call these baseline data product usability attributes. Every data product incorporates these characteristics to be part of the mesh. These are an addition to what has been known as **FAIR** data in the past—data that meets the principles of **D**indability, **A**ccessibility, **I**nteroperability, and **R**eusability. [8]

The introduction of analytical data as a product adds to the list of existing responsibilities of cross-functional domain teams [9] and expands their roles to:

- **Data product developer:** The role responsible for developing, serving, and maintaining the domain's data products as long as the data products live and are being used. They will be working alongside their fellow application developers in the domain.
- **Data product owner:** The role accountable for the success of a domain's data products in delivering value, satisfying and growing the data users, and maintaining the life cycle of the data products. He assure continuity of ownership of data and accountability

2.1. FOUR FUNDAMENTAL PRINCIPLES OF DATA MESH

of success metrics such as data quality, decreased lead time of data consumption, and in general data user satisfaction through net promoter score.

Define these roles for each domain and allocate one or multiple people to the roles depending on the complexity of the domain and the number of its data products. Moreover, to make a data as a product, we need to satisfy these conditions:

- Reframe the Nomenclature: Data mesh suggests reframing receiving upstream data from ingestion to consumption. The minor distinction is that the upstream data has already been cleansed, processed, and is ready for consumption.
- Think of Data as a Product, not a mere asset.
- Establish a Trust-But-Verify Data Culture: The data as a product principle entails a variety of actions that contribute to a culture in which data users can trust the veracity of the data and focus on proving its suitability for their use cases. Data-as-a-product practices strive to create a new culture, moving away from presumption of guilt.
- Join Data and Compute as One Logical Unit: Coexistence of data and code is not a novel concept. The expansion of operational systems has resulted in a model in which each service handles its own code and data, as well as schema definition and upgrades. The link between the code and its data distinguishes an operational system.

2.1.3 Principle of the Self-Serve Data Platform

The principle comes to the rescue in order to reduce the cognitive load imposed on the existing domain teams by the other two principles: own your analytical data and distribute it as a product.

It shares several of the same features as existing platforms, such as access to polyglot storage, data processing engines, query engines, streaming, and so on. It differs from previous platforms, however, in its users: autonomous domain teams comprised mostly of generalist technologists. It maintains a higher-level data product architecture that encapsulates data, metadata, code, and policy as a single entity.

Its goal is to empower domain teams by hiding low-level complexity behind simpler abstractions and minimizing friction from their journeys toward their goal of exchanging data products as a unit of value. It prefers decentralized, interoperable methods to expand out data exchange beyond a particular deployment environment or organizational unit.

2.1.4 Principle of Federated Computational Governance

The response to the questions about the data mesh governance model's change from centralized to decentralized governance, known as federated computational governance. Data mesh, like lake and warehouse, serves a similar set of governance goals. However, it differs in its operating strategy and how these goals are attained.

The data mesh governance model consists of three complementary pillars:

- **System thinking:** Consider the mesh to be an ecosystem of interconnected data product and platform systems, with independent and yet linked teams. Then, try to locate the leverage points and feedback loops that will allow you to manage the behavior of

the mesh as a whole toward the goal of creating value by exchanging data products at scale.

- **Apply a federated operating model:** Create a federated team of individual domains and platform representatives from a social and organizational standpoint. Create incentives that are aligned with the performance of both domains' data products and the overall success of the ecosystem. Allow domains autonomy and accountability for the vast majority of policies within their influence and control, while leaving cross-functional and a small number of policies to be developed worldwide.
- **Embed the governance policies** into each data product in an automated and computational fashion.

To bring these three pillars together, Figure 2.3 shows an example of this model.

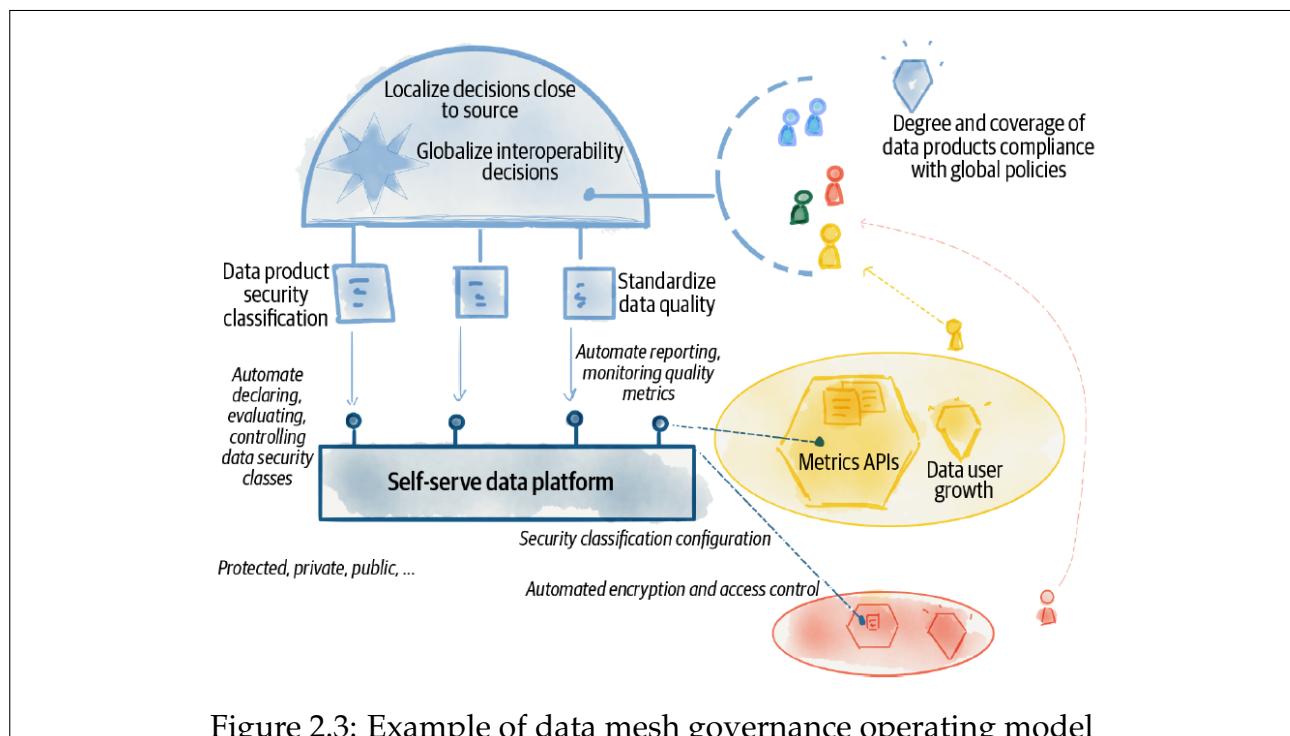


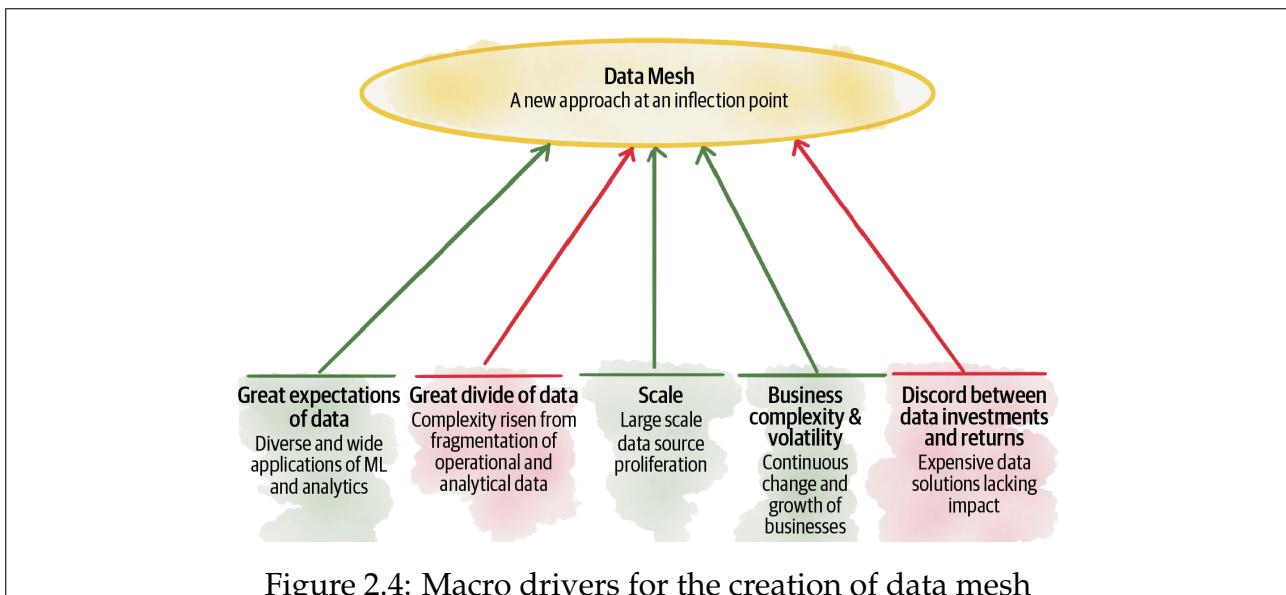
Figure 2.3: Example of data mesh governance operating model

Data mesh governance seeks to improve the existing approach to data governance at the intersection of a decentralized value system meeting automation and computation.

Data mesh governance is the method by which we define and apply what is right and how to do the right thing in an ecosystem of data product teams. It defines an approach that attempts to continuously find the dynamic equilibrium between localization of decisions so that the domains can go fast versus globalization and centralization of decisions for everyone to go far.

However, the mesh will not reach an optimal state unless the majority of domains become intelligently augmented - with embedded ML-based systems in each of their products and systems. The continuous need for trustworthy and useful data across multiple domains to train ML-based solutions will be the ultimate motivator for the adoption of data mesh governance and doing the right thing.

2.2. WHY WE NEED DATA MESH?



2.2 WHY WE NEED DATA MESH?

Data mesh is what comes after an inflection point, shifting our approach, attitude, and technology toward data. Data mesh assumes a new default starting state: proliferation of data origins within and beyond organizations' boundaries, on one or across multiple cloud platforms.

After the inflection point, data mesh make us reimagine data, specifically how to design solutions to manage it, how to govern it, and how to structure our teams: [10]

Table 2.1: Summary of after the inflection point with data mesh

Goal	What to do	How to do it
Manage changes to data gracefully in a complex, volatile, and uncertain business environment	Align business, tech, and data	Create cross-functional business, tech, and data teams each responsible for long-term ownership of their data <i>Principle of domain data ownership</i>
	Close the gap between the operational and analytical data planes	Remove organization-wide pipelines and the two-plane data architecture Integrate applications and data products more closely through dumb pipes <i>Principle of data as a product</i>
	Localize data changes to business domains	Localize maintenance and ownership of data products in their specific domains Create clear contracts between domain-oriented data products to reduce impact of change <i>Principle of data as a product</i>
	Reduce the accidental complexity of pipelines and copying of data	Breakdown pipelines, move the necessary transformation logic into the corresponding data products, and abstract them as an internal implementation <i>Principle of data as a product</i> <i>Data product quantum architectural component</i>

continued on next page -

Table 2.1 – *continued from previous page*

Goal	What to do	How to do it
Sustain agility in the face of growth	Remove centralized architectural bottlenecks	Remove centralized data warehouses and data lakes Enable peer-to-peer data sharing of data products through their data interfaces <i>Principle of domain ownership</i> <i>Principle of data as a product</i>
	Reduce the coordination of data pipelines	Move from a top-level functional decomposition of pipeline architecture to a domain-oriented decomposition of architecture Introduce explicit data contracts between domain-oriented data products. <i>Principle of domain ownership</i> <i>Principle of data as a product</i>
	Reduce coordination of data governance	Delegate governance responsibilities to autonomous domains and their data product owners Automate governance policies as code embedded and verified by each data product quantum <i>Principle of federated computational governance</i>
	Enable team autonomy	Give domain teams autonomy in moving fast independently. Balance team autonomy with computational standards to create interoperability and a globally consistent experience of the mesh. Provide domain-agnostic infrastructure capabilities in a self-serve manner to give domain teams autonomy. <i>Principle of federated computational governance</i> <i>Principle of the self-serve data platform</i>
Increase value from data over cost	Abstract complexity with a data platform	Create a data-developer-centric and a data-user-centric infrastructure to remove friction and hidden costs in data development and use journeys Define open and standard interfaces for data products to reduce vendor integration complexity <i>Principle of data as a product</i> <i>Principle of the self-serve data platform</i>
	Embed product thinking everywhere	Focus and measure success based on data user and developer happiness Treat both data and the data platform as a product <i>Principle of the self-serve data platform</i> <i>Principle of data as a product</i>
	Go beyond the boundaries of an organization	Share data across physical and logical boundaries of platforms and organizations with standard and internet-based data sharing contracts across data products <i>Principle of data as a product</i> <i>Principle of the self-serve data platform</i>

While the evolution of data architecture has been necessary and an improvement, all of the existing analytical data architectures share a common set of characteristics that inhibit

2.3. DATA MESH ARCHITECTURE DESIGN

them from scaling organizationally. They are all monolithic with centralized ownership and technically partitioned. It is no longer valid when data gets sources from hundreds of microservices and millions of devices from within and outside of enterprises.

2.3 DATA MESH ARCHITECTURE DESIGN

2.3.1 The Logical Architecture

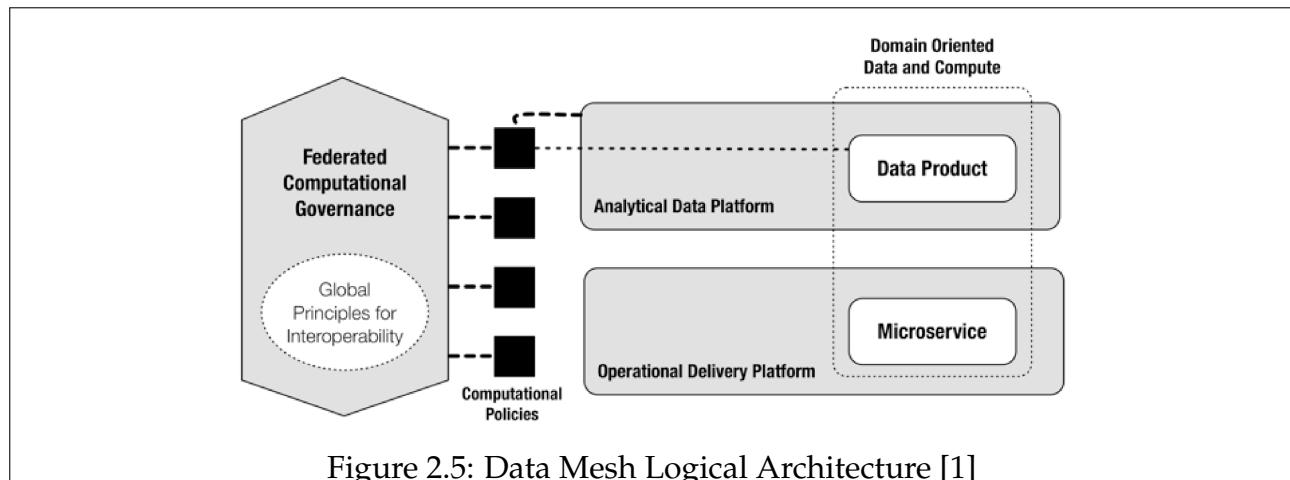


Table 2.2: Data mesh logical architectural components and their maturity

Aspect	Architectural component	Description
Domain-Oriented Data Sharing Interfaces	Domain	Systems, data products, and a cross-functional team aligned to serve a business domain function and outcomes and share its analytical and operational capabilities with the wider business and customers. <i>This is a well-established concept.</i>
	Domain analytical data interfaces	Standardized interfaces that discover, access, and share domain-oriented data products. <i>Up to now, the implementation of these APIs is custom or platform specific.</i> The proprietary platforms need to offer open interfaces to make data sharing more convenient and interoperable with other hosting platforms.
	Domain operational interfaces	APIs and applications through which a business domain shares its transactional capabilities and state with the wider organization. <i>This concept has mature implementations.</i> It is supported by de facto standards such as REST, GraphQL, gRPC, etc.

continued on next page –

Table 2.2 – *continued from previous page*

Aspect	Architectural component	Description
Data product quantum	Data product structural components	<p>Data product implemented as an architecture quantum that encapsulates all the structural components it needs to do its job—code, data, infrastructure specifications, and policies.</p> <p>It is referred to in architectural discussions. It is used interchangeably with data products.</p> <p><i>At the time of writing this is an experimental concept with custom implementations.</i></p>
	Data Product Data Sharing Interactions	<p>There are many technologies for implementing a data product's input and output data port of data mesh, but it shares the common properties:</p> <ul style="list-style-type: none"> • Input data port: A data product's mechanisms to continuously receive data from one or multiple upstream sources. <p><i>At the time of writing this has custom implementations with existing event streaming and pipeline management technologies.</i></p> <ul style="list-style-type: none"> • Output data port: A data product's standardized APIs to continuously share data. <p><i>At the time of writing this has vendor-specific custom implementations.</i></p> <p><i>A mature implementation of the concept requires open data sharing standards with support for multiple modes of access to temporal data.</i></p>
	Discovery and observability APIs	<p>A data product's standard APIs to provide discoverability information - to find, address, learn, and explore a data product - and observability information such as lineage, metrics, logs, etc.</p> <p><i>At the time of writing custom implementations of these APIs have been built.</i></p> <p><i>A mature implementation requires open standards for discoverability and observability information modeling and sharing. Some standards³ are currently under development.</i></p>
The Multiplane Data Platform	Platform plane	<p>A group of self-serve platform capabilities with high functional cohesion surfaced through APIs.</p> <p><i>This is a general concept and well established.</i></p>
	Data infrastructure (utility) plane	<p>Platform plane providing low-level infrastructure resource management - compute, storage, identity, etc.</p> <p><i>The infrastructure plane is mature and supported by many vendors with automated provisioning.</i></p>
<i>continued on next page –</i>		

2.3. DATA MESH ARCHITECTURE DESIGN

Table 2.2 – continued from previous page

Aspect	Architectural component	Description
	Data product experience plane	Platform plan providing operations on a data product. <i>Custom implementation of services for data product experience plane, no reference implementation available.</i>
	Mesh experience	Platform plane providing operations on the mesh of connected data products. <i>Custom implementations of discovery and search services have been implemented, but no reference implementation exists.</i>
Embedded Computational Policies ⁴	Data product container	A mechanism to bundle all the structural components of a data product, deployed and run as a single unit with its sidecar. <i>At the time of writing this is an experimental concept with custom implementations.</i>
	Data product sidecar	The accompanying process to the data product implements cross-functional and standardized behaviors. This still is an experimental concept with custom implementations.
	Control port	A data product's standard APIs to configure policies or perform highly privileged governance operations. <i>At the time of writing this concept is experimental.</i>

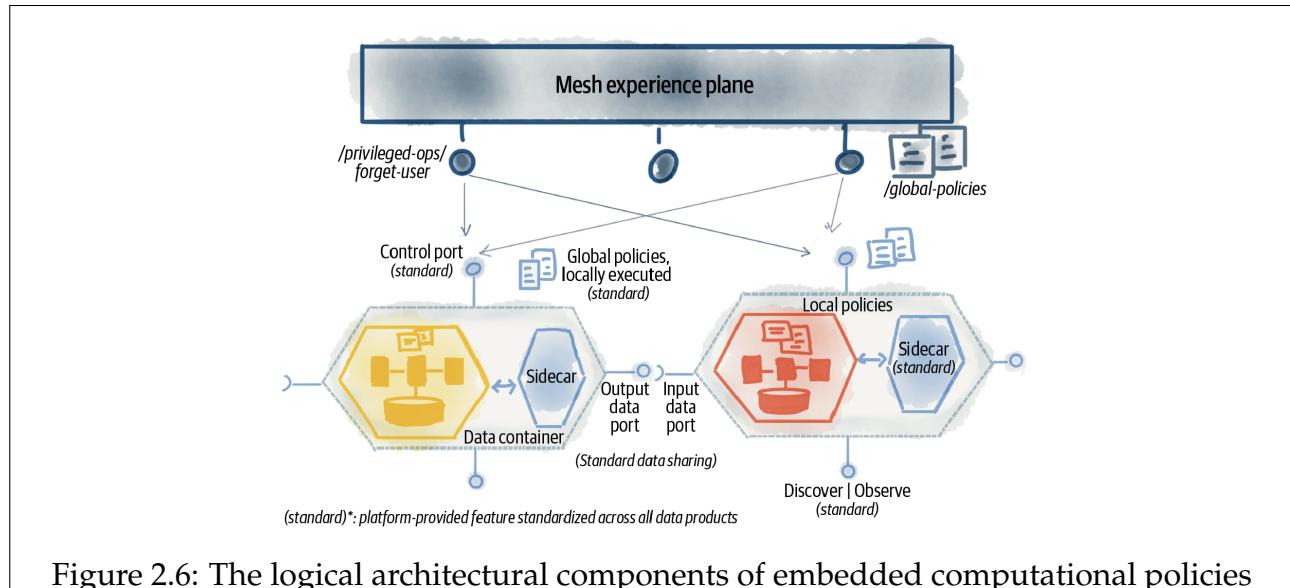


Figure 2.6: The logical architectural components of embedded computational policies

It is intentionally that the technology evolves to the point that we can get the logical architecture and its physical implementation as close to each other as possible. [11]

³OpenLineage is an example of an observability open standard that data products can adopt.

⁴See figure 2.6

2.3.2 The Multiplane Data Platform Architecture

As we have discussed previously, the multiplane data platform consists of three planes, with the interactive diagram as figure 2.7.

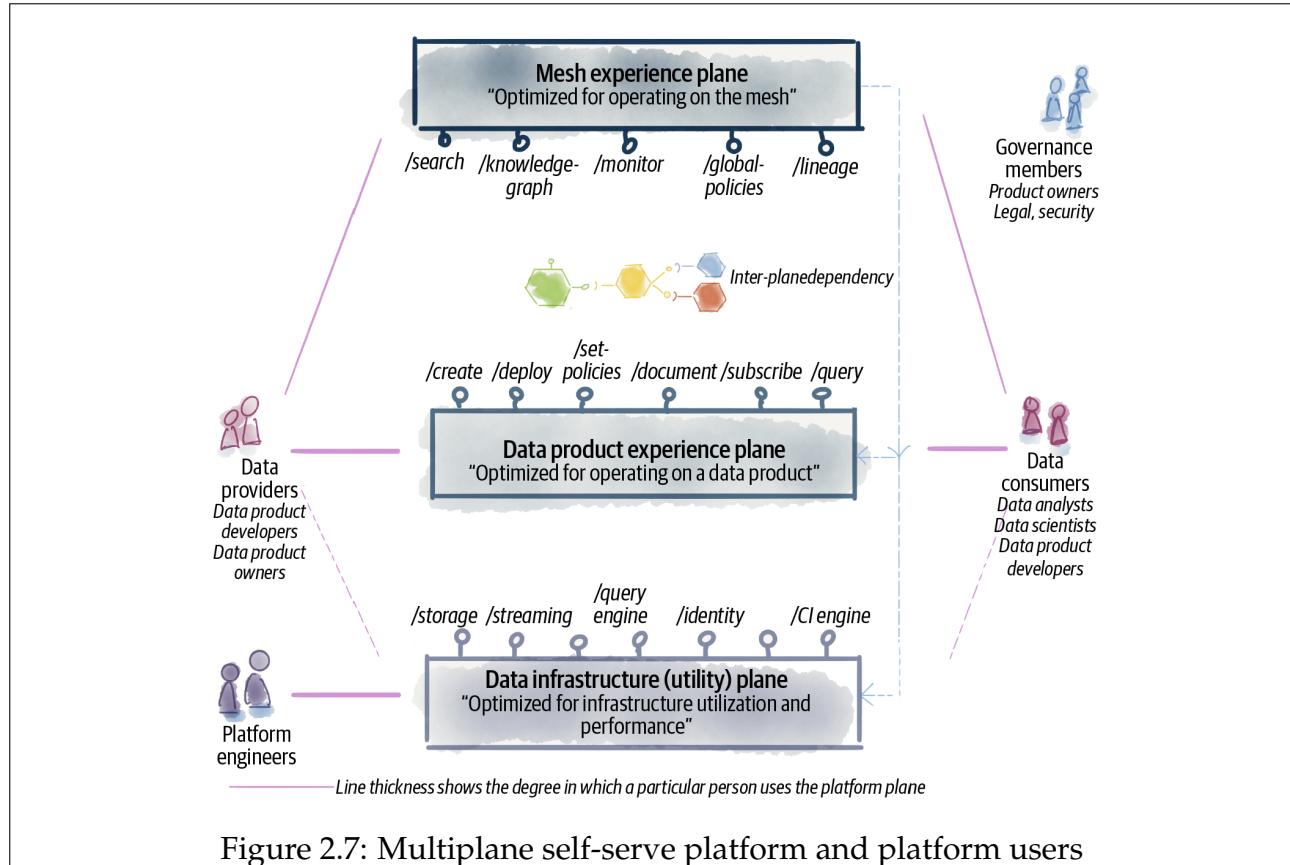


Figure 2.7: Multiplane self-serve platform and platform users

The data product experience plane provides operations on a data product and manages the complexity of provisioning its underlying infrastructure, while the data infrastructure utility plane optimizes the resources' performance and utilization to get the best out of the underlying infrastructure providers. The most important idea is to understand the main journeys of the platform users and evaluate how to make it easy for them to complete their journeys.

Data Product Developer Journey

Creating and operating a data product is one of the most important journeys of a data product developer. This is a comprehensive and long-term obligation that adheres to the continuous delivery premise of "everyone is responsible." A data product development journey interacts with other journeys. Let's look at the figure 2.8 for an example of high-level stages of data product development and how the platform interfaces are designed to support them.

We should go deeper to consider each stage of the platform planes and interfaces

2.3. DATA MESH ARCHITECTURE DESIGN

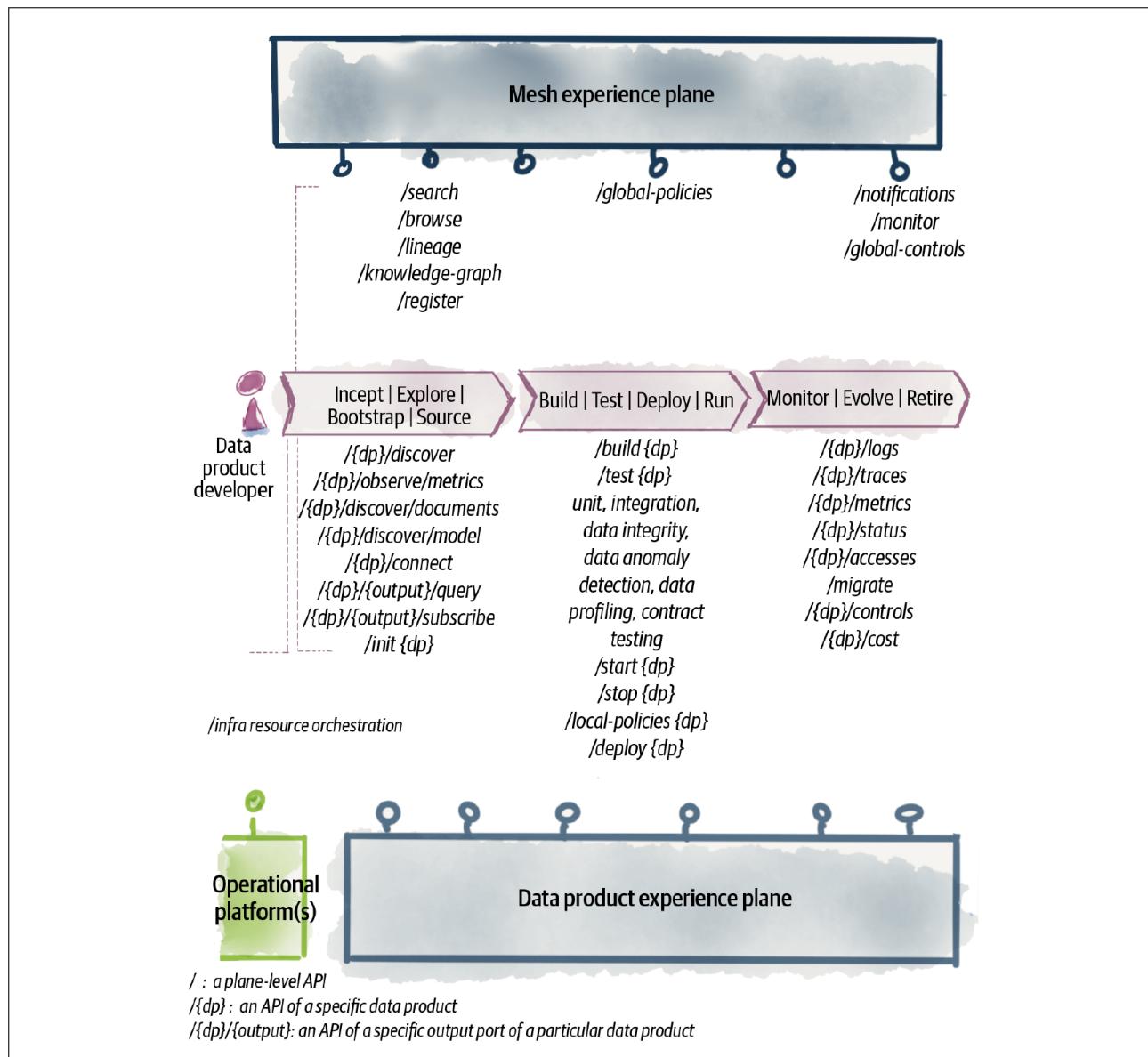


Figure 2.8: High-level example of data product development journey using the platform

Table 2.3: Example interfaces provided by the platform planes

Phase	Platform plane	Platform interface	Platform interface description
Data product inception			
Incept Explore	Mesh experience	/search	Search existing data products based on parameters such as operational systems, domains, and types of data.
		/knowledge-graph	Browse the mesh of related data products' semantic models. Traverse their semantic relationship to identify the desired sources of data.
		/lineage	Traverse the lineage of input-output data to identify desired sources based on origin and transformations.

continued on next page –

Table 2.3 – *continued from previous page*

Phase	Platform plane	Platform interface	Platform interface description
Bootstrap Source	Data product experience	/dp}/discover	Once a source is identified, access all the data product discoverability information such as documentation, data model, available output ports, etc.
		/dp}/observe	Access data product guarantees and metrics in real-time, such as release frequency, last release date, and data quality metrics.
		/init	API bootstraps a barebones data product with infrastructure to connect to sources, access data, run transformations, and serve output in a single access mode.
		/connect	The data product gets access to the source by connecting to it, validating access control policies and triggering a request for permission.
		/dp}/ {output}/query /dp}/ {output}/subscribe	Read data from a particular output port of the source data product, either in a pull-based querying model or subscribing to changes.
	Mesh experience	/register	Data products are registered with the mesh and given a unique global identifier and address, making them visible to the governance process.
Data product development			
Build	Data product experience	/build	Compile, validate, and compose components of a data product into a deployable artifact. See Table 2.3 for the data infrastructure plane interfaces used during this phase.
Test		/test	Testing capabilities are offered in different deployment environments to test various aspects of a data product, such as data transformation, integrity, versioning, data profile, and bias.
Deploy		/deploy	Deploy a data product to multiple environments, including local, development, pre-production, and production.
Run		/start /stop	Run or stop running the data product instance in a particular environment.
Build / Test / Deploy / Run		/local-policies	The platform facilitates configuration and authoring of these policies locally, during the development of a data product, their validation during test, and their execution during access to data.
<i>continued on next page –</i>			

2.3. DATA MESH ARCHITECTURE DESIGN

Table 2.3 – *continued from previous page*

Phase	Platform plane	Platform interface	Platform interface description
Build/ Test/ Deploy/ Run	Mesh experience	/global-policies	The platform enables authoring of the global policies and application of these policies by all data products.
Data product maintenance			
Maintain, Evolve, and Retire	Data product experience	/dp}/status	Checking the status of a data product.
		/dp}/logs	Data products emit runtime observability information such as logs, traces, and metrics to be used by mesh layer monitoring services.
		/dp}/traces	
		/dp}/metrics	
	Mesh experience	/dp}/accesses	The logs of all accesses to the data product.
		/dp}/cost	Tracking the operational cost of a data product. This can be calculated based on the resource allocation and usage.
		/migrate	Updating a data product revision is a simple process of build and deploy.
	Data product experience plane	/monitor	Multiple monitoring abilities at the mesh level, logs, status, compliance, etc.
		/notifications	Notification and alerting in response to detected anomalies of the mesh.
		/global-controls	High-privilege administrative controls can be invoked on data products.
		/dp}/controls	Data products are registered with a unique global identifier and address, making them visible to governance.

The data infrastructure plane is a key enabler in lowering the cost and effort required to code or configure components of a data product. Developers interact with the data product experience plane to build, deploy, and test a single data quantum. The data product experience plane delegates the implementation of the data product components to the data infrastructure plane.

Table 2.4: Data platform interfaces that support data product experience plane APIs

Platform plane	Platform interface	Platform interface description
Data product experience	/dp}/logs	Data products emit runtime observability information such as logs, traces, and metrics to be used by mesh layer monitoring services.
	/dp}/traces	
	/dp}/metrics	
	/dp}/status	Checking the status of a data product.
	/dp}/accesses	The logs of all accesses to the data product.
	/dp}/cost	Tracking the operational cost of a data product. This can be calculated based on the resource allocation and usage.

continued on next page –

Table 2.4 – *continued from previous page*

Platform plane	Platform interface	Platform interface description
	/migrate	Ability to migrate a data product to a new environment. Updating the data product revision is simply a function of build and deploy.
Mesh experience	/monitor	Multiple monitoring abilities at the mesh level, logs, status, compliance, etc.
	/notifications	Notification and alerting in response to detected anomalies of the mesh.
	/global-controls	Ability to invoke high-privileged administrative controls such as right to be forgotten on a collection of data products on the mesh.
Data product experience plane	/{dp}/controls	Data products are registered with the mesh and given a unique global identifier and address, making them visible to the governance process.

Data Product Consumer Journey

A data consumer is a user with various skill sets and responsibilities, such as consuming existing data products to train an ML model and then deploying it as a data product. ML models can belong to either plane. For example, an ML model can be deployed as a microservice to make inferences online as the end user makes requests. Alternatively, an ML model can be deployed as the transformation logic of a data product.

The data scientist's journey to continuously deliver an ML model as a data product follows the practice of continuous delivery for machine learning (CD4ML⁵) to continuously hypothesize, train, deploy, monitor, and improve the model in rapid feedback loops. The figure 2.10 illustrate this journey.

In conclusion, there is no single entity such as a platform but rather a set of well-integrated services, such as APIs, services, SDKs, and libraries that provide a seamless experience for users. Additionally, the separation of planes should be respected to optimize both for the human experience and machine efficiency. For example, the data product experience plane should optimize the user experience at the logical level, while the data infrastructure plane should optimize for the machine. Lower plane optimization decisions should not impair the developer experience, and vice versa.

⁵This process is also known as MLOps; however, I use the CD4ML notation since it overlays and integrates closely with continuous delivery of software and data products.

2.3. DATA MESH ARCHITECTURE DESIGN

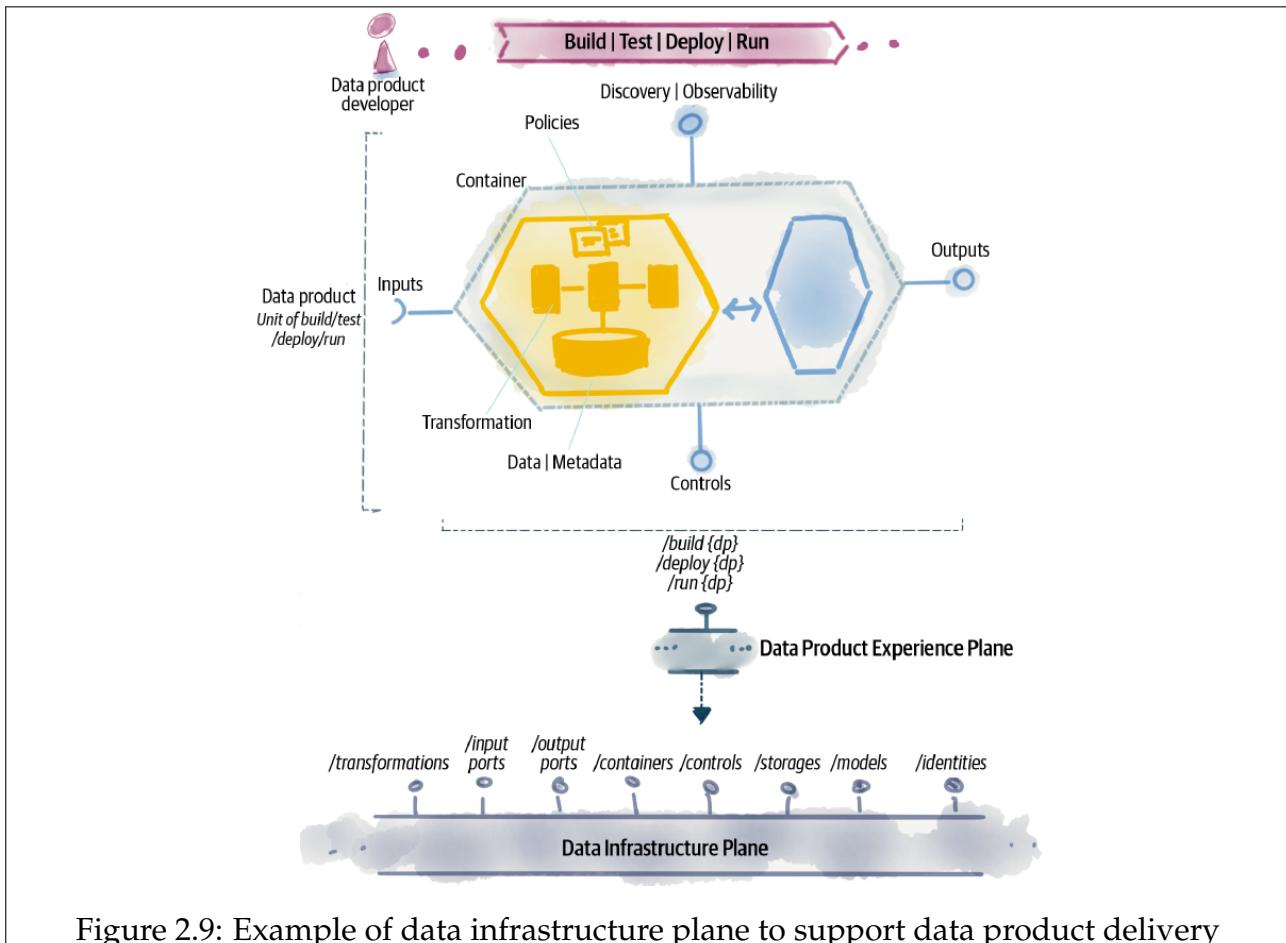


Figure 2.9: Example of data infrastructure plane to support data product delivery

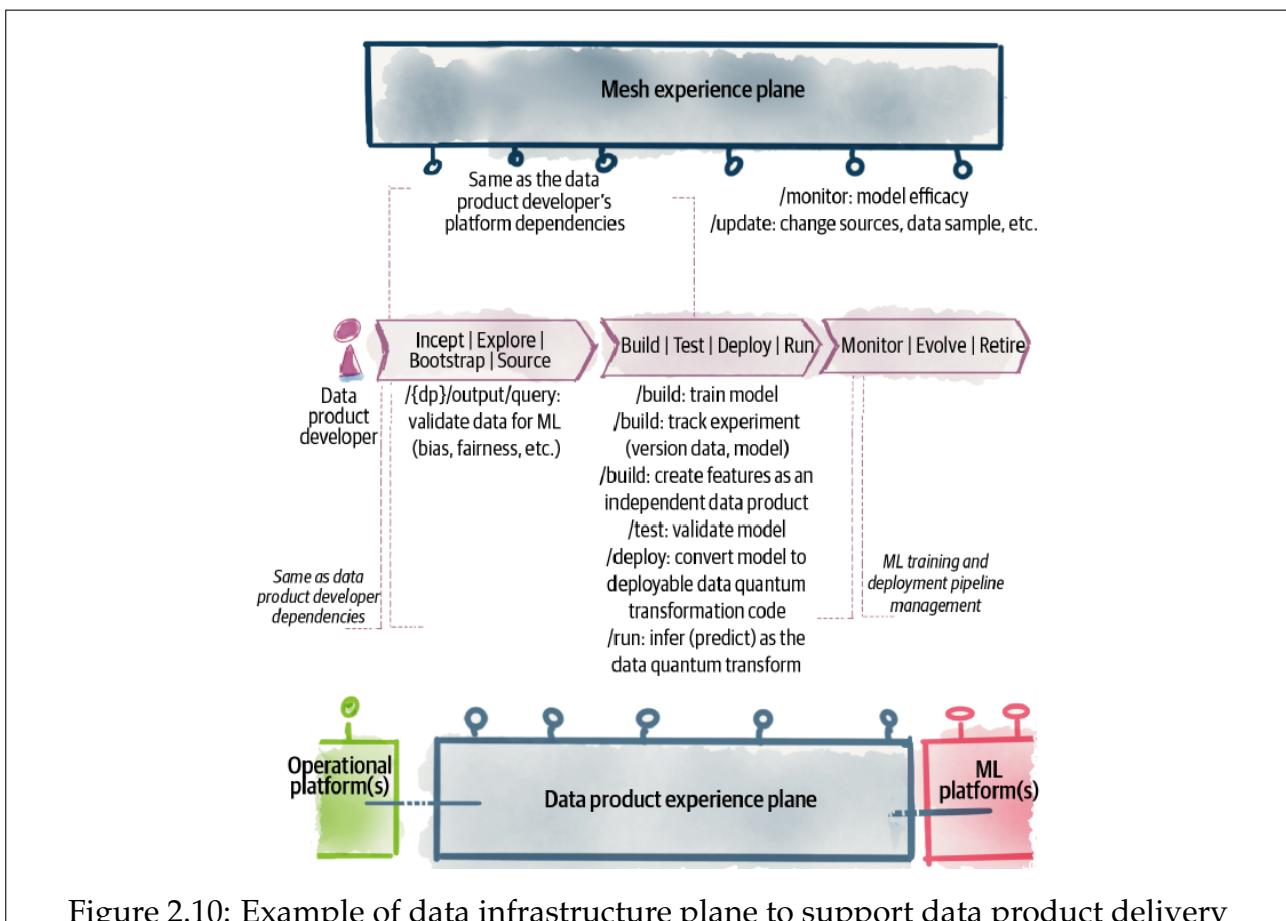


Figure 2.10: Example of data infrastructure plane to support data product delivery

Chapter 3

DATA PRODUCT DESIGN & IMPLEMENTATION

3.1 DESIGN A DATA PRODUCT BY AFFORDANCES

Data mesh is a distributed scale-out architecture that designs data products as self-contained, autonomous, and coequal nodes, providing users and developers with the capabilities they need to do their job: discover, understand, use, build, govern, and debug.

Fundamentals of Software Architecture defines the constituents of architecture as: Structure, Characteristics, Decisions, Principles. In addition to these, we introduce and focus on another element: affordances - one of the five fundamental principles of interactions [12]. Hence, this approach to the design of data product architecture design by affordances.

Table 3.1 lists affordances for a data product to autonomously transform and serve meaningful, understandable, trustworthy data that can be connected to other data in the ecosystem. It has all the structural components to do its job, and its dependencies to other data products or platform services are through APIs with explicit contracts, creating loose coupling.

Table 3.1: Data product affordances

Affordances	Description
Serve Data	The data product shares immutable and bitemporal data, allowing users to access it for training ML models, generating reports, data analysis, and building data-intensive applications, but not for transactional and operational applications.
Consume Data	The data product consumes data from upstream sources, but only from sources it identifies and configures through the platform, not from sources it does not identify and configure.
Transform Data	The data product transforms input data into new data, which can be program code, a machine learning model, or a complex query. It can generate new data or improve the quality of input data.
Discover Understand Explore Trust	The data product serves APIs and information that affords data product users to discover, explore, understand, and trust it.

continued on next page –

3.2. CONSUMING, TRANSFORMING & SERVING DATA

Table 3.1 – *continued from previous page*

Affordances	Description
Compose Data	The data product allows users to compose, correlate, and join data with other data products, while the data quantum provides programmatic data composability by performing set operations.
Manage Life Cycle	Data products provide a set of configuration and code to enable developers to build, provision, and maintain them.
Observe Debug Audit	The data product provides programmatic APIs to enable users to monitor its behavior, debug issues, and audit it.
Govern	The data product provides data users and the mesh experience plane with APIs and policies to self-govern their data, enabling build-time configuration and runtime execution of policies. It also maintains data security and protects privacy and confidentiality through encryption.

Data products must be designed to meet the data mesh objective of "getting value from data at scale", that means, we have to satisfy:

- Design for change: Design for change is essential for data products to respond gracefully to change, such as fronting aspects with APIs or adding time.
- Design for scale: A data product must scale out while maintaining agility and speed of access.
- Design for value: Data product interfaces should focus on sharing easily understandable, trustworthy data with minimal friction.

The final thing is design simple and local rules and behaviors in data products to create mesh-level knowledge and intelligence.

3.2 CONSUMING, TRANSFORMING & SERVING DATA

A data product's primary job is to consume data from upstream sources using its input data ports, transform it, and serve the result as permanently accessible data via its output data ports.

3.2.1 Serve data

Data products serve domain-oriented data to diverse consumers through output data ports and APIs. Serve data design has some properties:

- Multimodal data: Data products serve data of a domain with a specific and unique domain semantic. They can be served in different formats. When a user accesses a data product, they first learn about its semantic and then access one of its output APIs.
- Immutable data: Immutable data is a key axiom in functional programming. This is relevant to analytical use cases, as it allows users to rerun analytics in a repeatable way.
- Bitemporal data: Bitemporal data modeling records two timestamps: actual time and processing time, making it possible to serve data as immutable entities and perform temporal analysis and time travel.

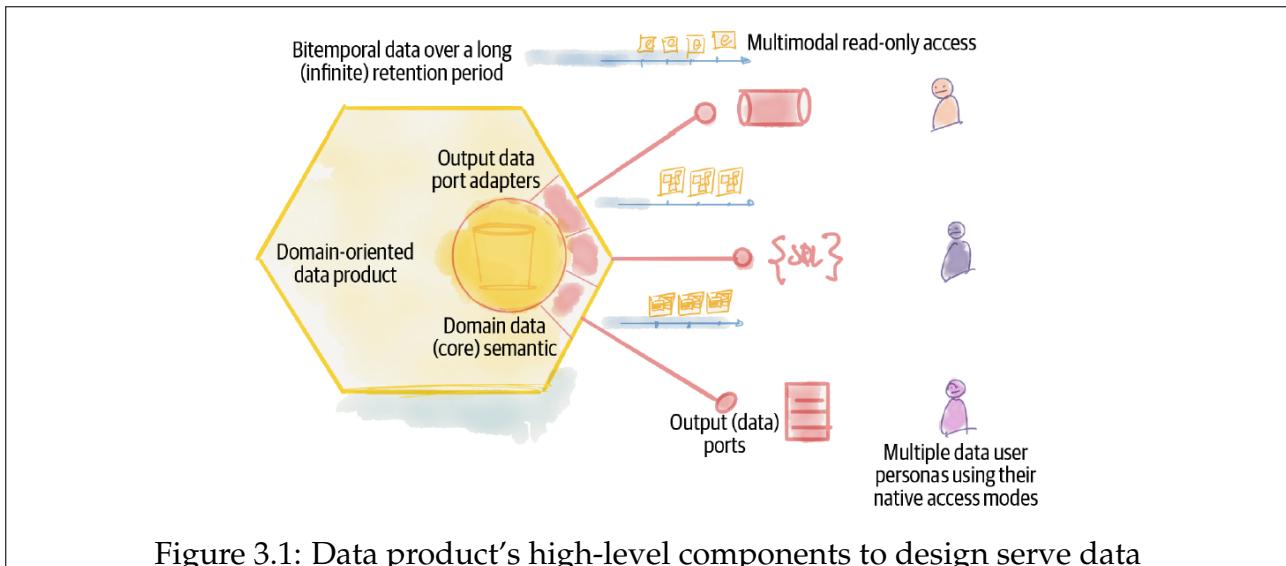


Table 3.2: High-level data product components to serve data

Component	Description
Output data port	APIs are used to serve data according to a specific mode of access for a particular spatial format.
Output (data) port adapter	The code responsible for presenting data for a particular output port is either a step in the data product's transformation code or a runtime gateway.
Core data semantic	Expression of the data semantic - agnostic to the modes of access or its spatial syntax.

3.2.2 Consume Data

In most of cases, data in organizations originates from internal or external systems, and data products consume it from one or multiple sources. Input data ports are a logical architectural construct that allow a data product to connect to a source, execute queries, and receive data as a continuous stream or one-off payload. Consume data design has some notable characteristics that impact the design of a data product's input data:

- Archetypes of data sources: collaborate operational systems, data products or itself.
- Locality of Data Consumption.

Table 3.3: High-level components to design consume data

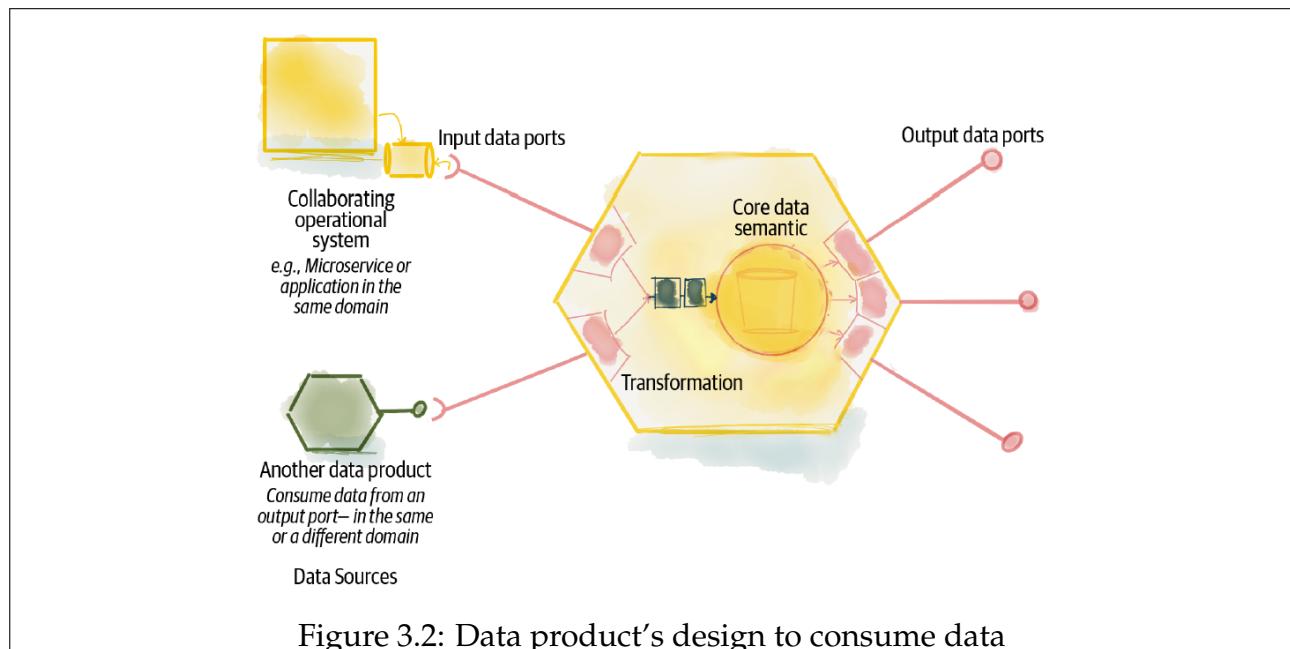
Component	Description
Input data port	The mechanism by which a data product receives its source data and makes it available for internal transformation.
Input data port specification	A declarative specification of input data ports that configures from where and how data is consumed.
Asynchronous input data port	Asynchronous input ports reactively execute transformation code when necessary source data is available.

continued on next page –

3.2. CONSUMING, TRANSFORMING & SERVING DATA

Table 3.3 – *continued from previous page*

Component	Description
Remote query	An input port specification can include a query executed on the source to receive desired data, reducing the amount of data fetched redundantly.
Input port synchronizer and temporary storage	Temporary storage is necessary to keep track of observed and unprocessed data until all observations are available for processing.



3.2.3 Transform Data

Data products are created to share a new analytical model of existing data. Transformation is an internal implementation of a data product and is controlled by it. It is up to the data product developer to choose how to implement the transformation, and it is helpful to look at a few different ways of implementation:

- Nonprogrammatic: Nonprogrammatic transformations use relational algebra or flow-based functions to capture the intent of how data is transformed, but are limited to the features of the statement.
- Programmatic: Programmatic data processing uses code logic, conditions, and statements to transform data, allowing it to be modularized and tested, making it more extensible.
- Dataflow-Based: Dataflow programming is a natural paradigm for implementing transformation code, as long as the pipeline stages don't extend beyond the boundary of a data product.
- ML Model: ML models can be deployed in many contexts to make predictions and store recommendations for playlist extension.
- Time-Variant: Transformations respect the axes of time, processing and actual, by keeping track of processing time and generating output with actual time.

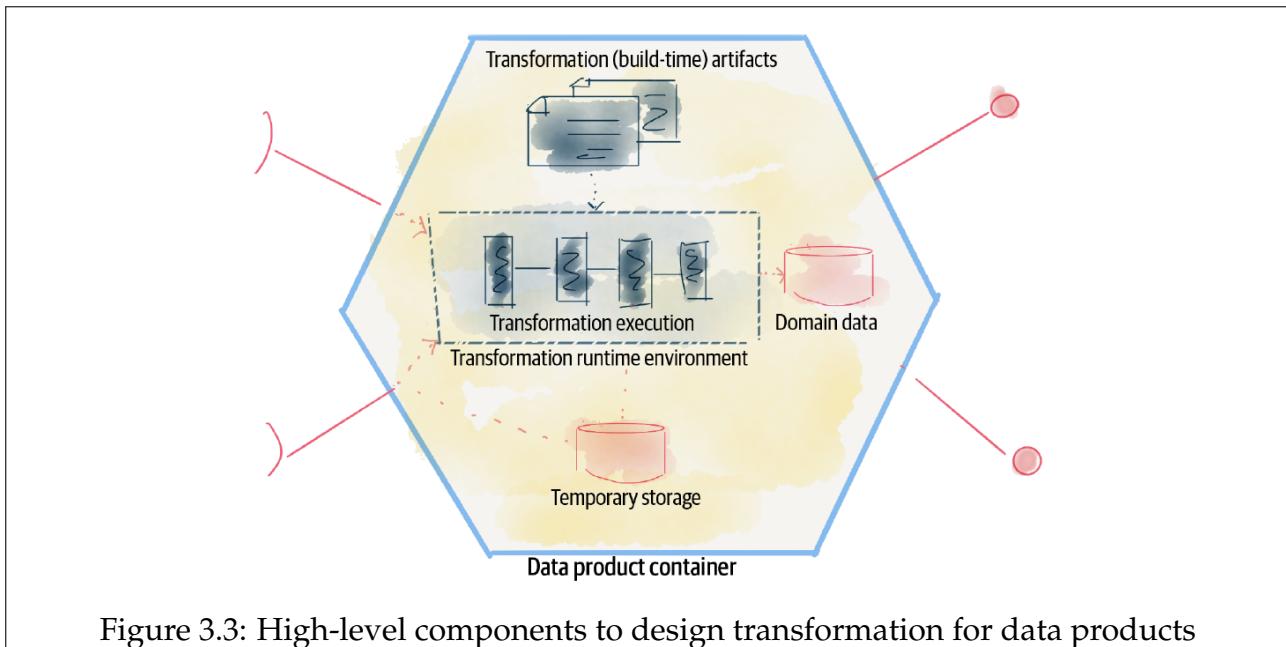


Figure 3.3: High-level components to design transformation for data products

Table 3.4: High-level components to transform data for a data product

Component	Description
Transformation artifact(s)	Code, configuration, statement, or model executes transformation on input data.
Transformation runtime environment	Transformations require a computational environment to execute, provided by the underlying platform.
Temporary storage	Platform provides temporary storage for transformation code steps.

3.3 DISCOVERING, UNDERSTANDING & COMPOSING DATA

3.3.1 Discover, Understand, Trust, and Explore

Data products are responsible for bridging the gap between the known and unknown, so users trust them. Interfaces and behaviors are designed to share information about data semantics, formats, usage documentations, statistical properties, expected quality, timeliness, and other metrics. ML-based discovery functions can be built to search, browse, examine, and get meaning from the mesh of connected data products.

To satisfy the above design, we need to follow these steps:

1. Begin discovery with self-registration,
2. Discover the global URI,
3. Understand semantic and syntax models,
4. Establish trust with data guarantees,
5. Learn with documentation.

The data product sidecar is a platform-provided agent that runs within the computational context of a data quantum and is responsible for cross-cutting concerns that need to be stan-

3.3. DISCOVERING, UNDERSTANDING & COMPOSING DATA

dardized across all data products. Figure demonstrates the data product sidecar's interactions in support of discoverability.

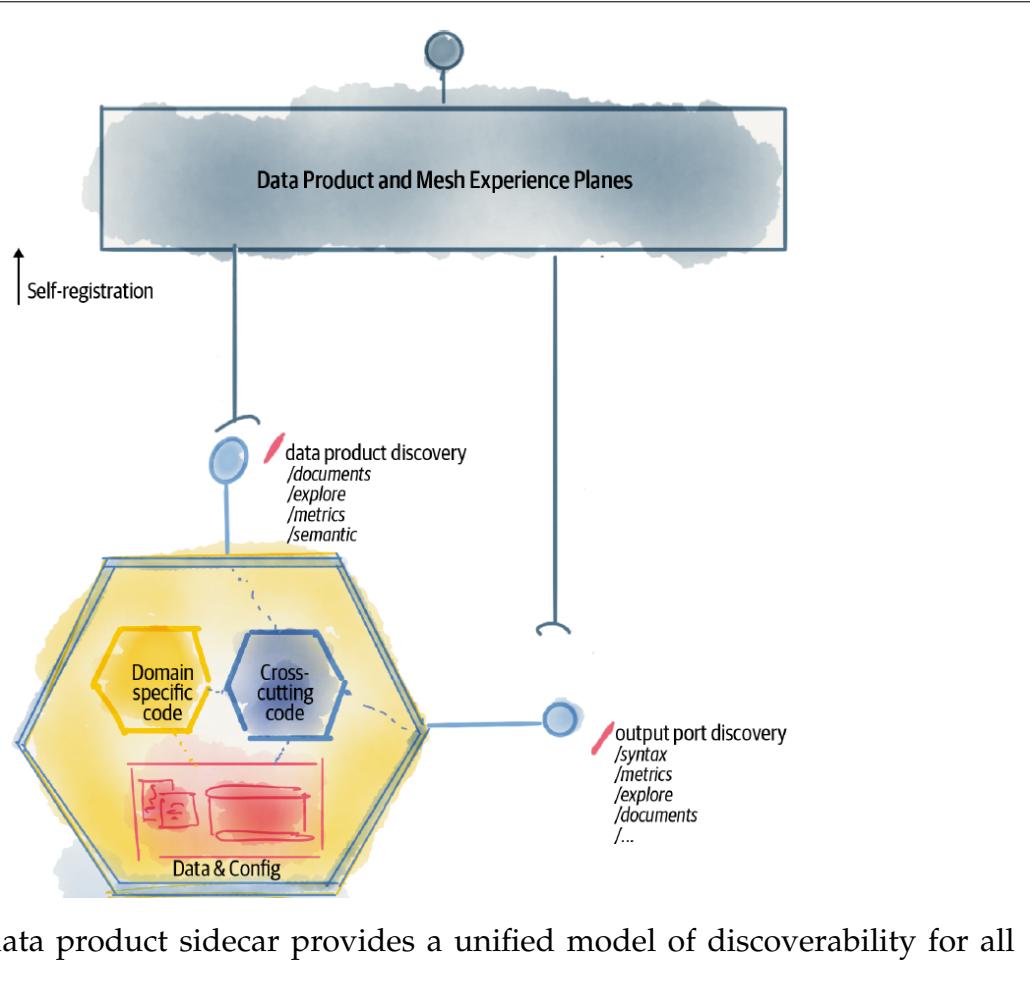


Figure 3.4: A data product sidecar provides a unified model of discoverability for all data products

3.3.2 Compose Data

The design considerations for a system of distributed data models that enables data composability and quick evolution without building tightly connected or centralized models are the most essential details in this article. Ownership of a time-variant, sharable, and referenceable semantic data model, semantic model linking, a global identification system to map data across data products, and the responsibility of data product owners to continuously look for opportunities to discover and create meaningful relationships with other data products are among the design considerations. These design principles are required for individual data products to generate higher-order intelligence and understanding.

The design prioritizes loose coupling between data products and minimizing centralized synchronization points. Data composability across the mesh is based on a distributed type system (schemas) where each data product owns and controls its own life cycle.

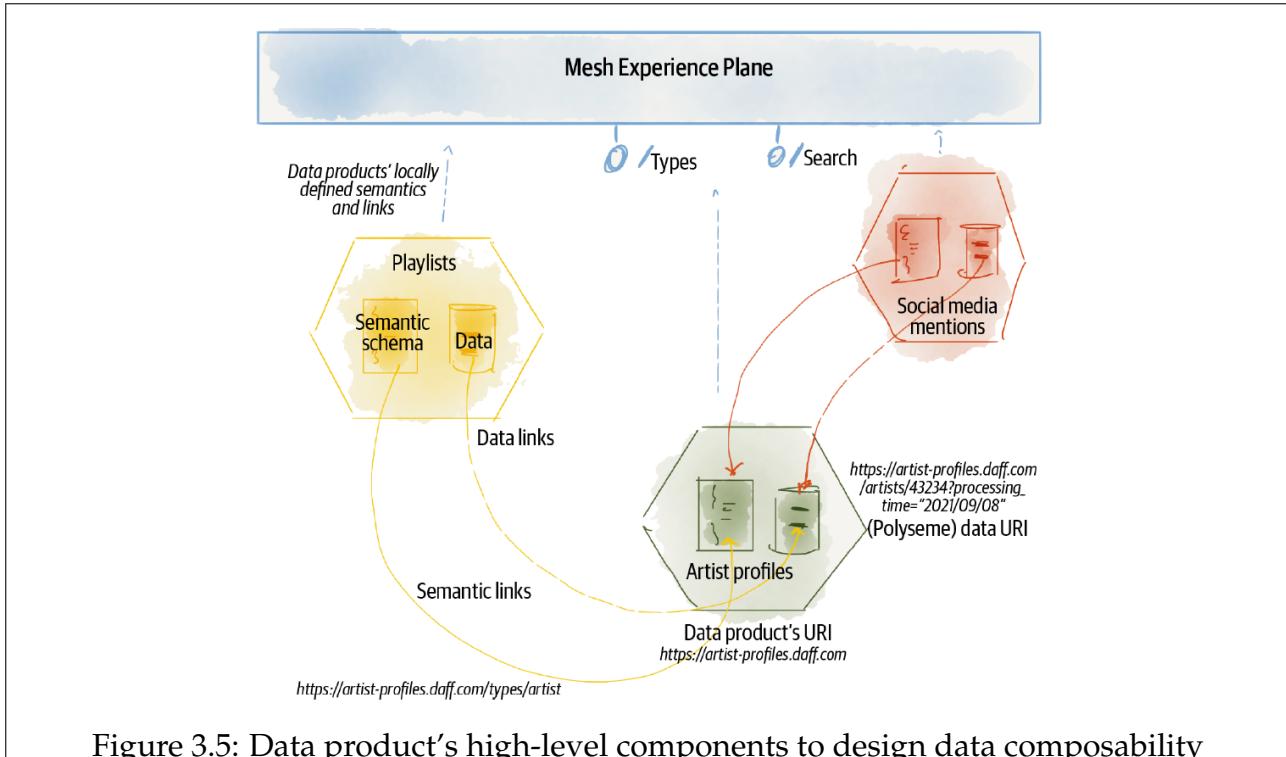


Figure 3.5: Data product's high-level components to design data composability

3.4 MANAGING, GOVERNING & OBSERVING DATA

3.4.1 Manage the Life Cycle

A data product's manifest is the specification of its target state, which can change and evolve during its lifetime. Developers create two groups of artifacts: source artifacts and data quantum manifest, which are used to generate the build and runtime artifacts, provision necessary resources, and run the data product container. The declarative modeling of a data product hides complexity, allowing developers to communicate various facets and properties of a data product in a standard fashion.

Table 3.5: High-level components to transform data for a data product

Component	Description
Data product's URI	Data product's globally unique identifier.
Output ports	Platform declares output ports, access modes, and guarantees.
Output port SLOs	Ports must declare the service level agreements they guarantee.
Input ports	Declaration of input ports, data sources, and retrieval methods.
Local policies	Configuration of local policies such as locality, confidentiality, etc.
Source artifacts	Data product includes transformation code and input ports queries.

3.4.2 Govern Data

Data products are architectural components that enable configuration and execution of policies. These components include a control port, data product sidecar, and data product container. These components work in collaboration with the mesh experience plane, which

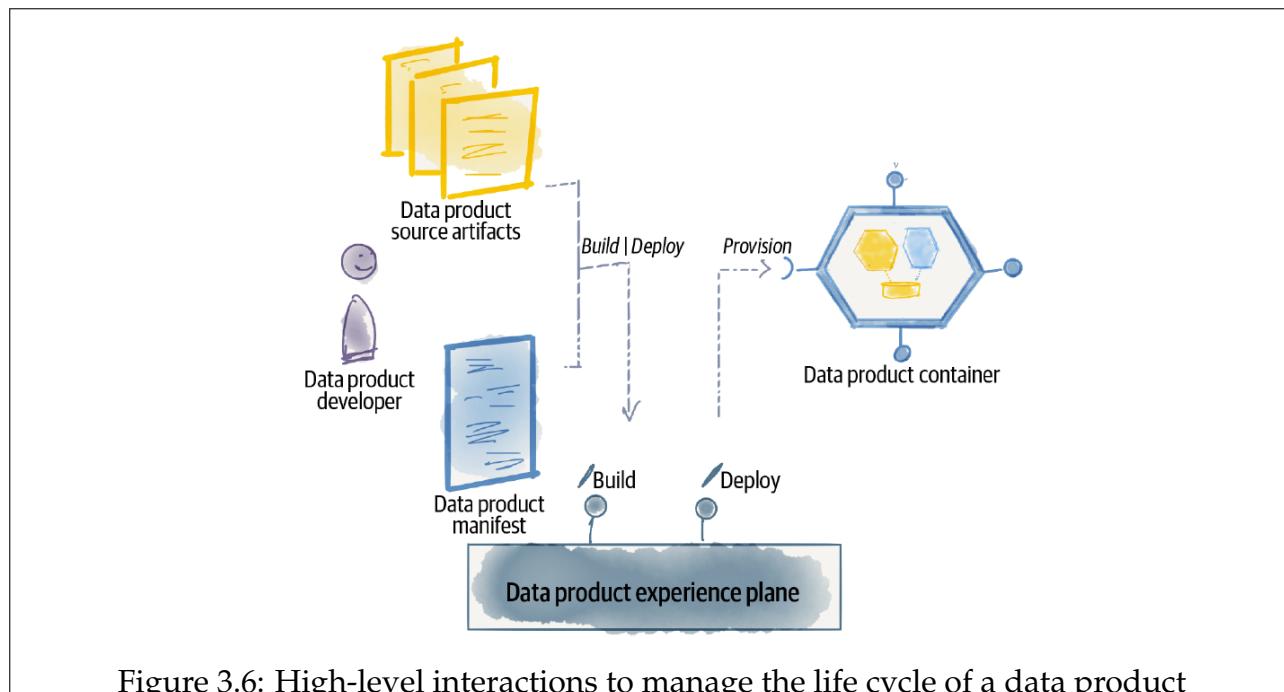


Figure 3.6: High-level interactions to manage the life cycle of a data product

provides capabilities for governance processes such as configuring policies and the right to be forgotten globally across the mesh. Governance of policies requires observability of their state of adoption, which can be domain-specific or agnostic. Successful implementation of data mesh governance requires design characteristics such as conformance to HIPAA and guaranteed data accuracy levels.

3.4.3 Observe, Debug, and Audit

Data product observability is the ability to infer the internal state of data products by observing their external outputs. The distributed architecture of data mesh creates complexity in observability, as there are many moving parts that can fail and failures may go unnoticed.

The use cases for data product observability can be summarized as follows:

- Monitor the operational health of the mesh,
- Debug and perform postmortem analysis,
- Perform audits,
- Understand data lineage.

Data mesh observability begins with each data product sharing its external output and reporting its status, allowing the experience plane to observe and monitor the mesh-level status. Some notable design characteristics of data products enable observability:

- Observable outputs: Distributed architectures use logs, traces, and metrics to enable observability.
- Traceability across operational and data planes: Observability of data must extend beyond data products to operational systems to provide a complete picture of data lineage and root cause analysis.
- Structured and standardized observability data: Data product metrics, logs, and traces must be structured and standardized to create higher-order intelligence and insights.

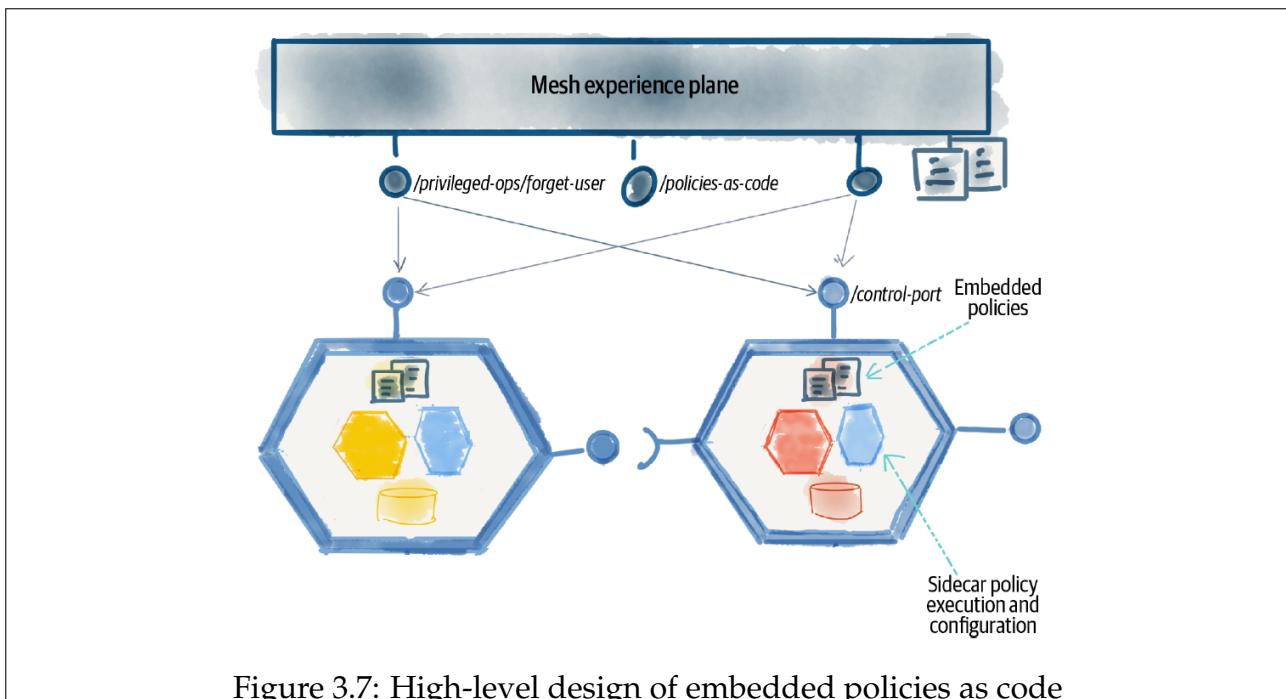


Figure 3.7: High-level design of embedded policies as code

- Domain-oriented observability data: Domain-oriented design can scale and extend observability by creating domains of observability and managing them as data products.

Chapter **4**

DATA MESH IN USE

4.1 DATA MESH IN COMBINATION WITH DATA LAKEHOUSE

4.2 FRAMEWORKS AND TECHNOLOGIES FOR DATA MESH

4.3 CASE STUDY AND DEMO

REFERENCES

- [1] I. A. Machado, C. Costa, and M. Y. Santos, "Data mesh: Concepts and principles of a paradigm shift in data architectures," *Procedia Computer Science*, vol. 196, pp. 263–271, 2022.
- [2] Z. Dehghani, "Prologue: Imagine data mesh," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. xxv–xxxviii.
- [3] C. Jochen, V. Larysa, and S. Harrer. (2023) Data mesh from an engineering perspective. [Online]. Available: <https://www.datamesh-architecture.com/>
- [4] D. Nicolas. (2023) Will the buzz around data mesh really help solve my data challenges? [Online]. Available: <https://kpmg.com/be/en/home/insights/2023/03/lh-the-impact-of-data-mesh-on-organizational-data.html>
- [5] Z. Dehghani, "Data mesh in a nutshell," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 3–14.
- [6] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.
- [7] Z. Dehghani, "Principle of domain ownership," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 15–28.
- [8] M. D. Wilkinson *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.
- [9] Z. Dehghani, "Principle of data as a product," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 29–46.
- [10] ——, "Why data mesh?" in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 93–137.
- [11] ——, "How to design the data mesh architecture," in *Data Mesh: Delivering Data-Driven Value at Scale*. O'Reilly Media, 2022, pp. 139–190.
- [12] D. A. Norman, *The Design of Everyday Things*. Basic Books, 2002.