Data Mesh Principles and Logical Architecture

Our aspiration to augment and improve every aspect of business and life with data, demands a paradigm shift in how we manage data at scale. While the technology advances of the past decade have addressed the scale of volume of data and data processing compute, they have failed to address scale in other dimensions: changes in the data landscape, proliferation of sources of data, diversity of data use cases and users, and speed of response to change. Data mesh addresses these dimensions, founded in four principles: domain-oriented decentralized data ownership and architecture, data as a product, self-serve data infrastructure as a platform, and federated computational governance. Each principle drives a new logical view of the technical architecture and organizational structure.

03 December 2020

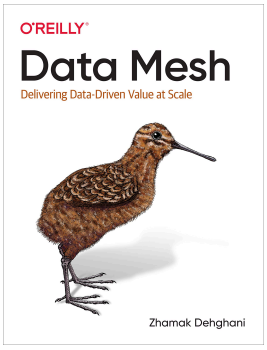
Zhamak Dehghani

Zhamak is the director of emerging technologies at Thoughtworks North America with focus on distributed systems architecture and a deep passion for decentralized solutions. She is a member of Thoughtworks Technology Advisory Board and contributes to the creation of Thoughtworks Technology Radar.

DATA ANALYTICS

DATA MESH

CONTENTS



For more on Data Mesh, Zhamak went on to write a full book that covers more details on strategy, implementation, and organizational design.

The original writeup, How to Move Beyond a Monolithic Data Lake to a Distributed Data Mesh - which I encourage you to read before joining me back here - empathized with today's pain points of architectural and organizational challenges in order to become data-driven, use data to compete, or use data at scale to drive value. It offered an alternative perspective which since has captured many organizations' attention, and given hope for a different future. While the original writeup describes the approach, it leaves many details of the design and implementation to one's imagination. I have no intention of being too prescriptive in this article, and kill the imagination and creativity

around data mesh implementation. However I think it's only responsible to clarify the architectural aspects of data mesh as a stepping stone to move the paradigm forward.

This article is written with the intention of a follow up. It summarizes the data mesh approach by enumerating its underpinning principles, and the high level logical architecture that the principles drive. Establishing the high level logical model is a necessary foundation before I dive into detailed architecture of data mesh core components in future articles. Hence, if you are in search of a prescription around exact tools and recipes for data mesh, this article may disappoint you. If you are seeking a simple and technology-agnostic model that establishes a common language, come along.

The great divide of data

What do we really mean by data? The answer depends on whom you ask. Today's landscape is divided into **operational data** and **analytical data**. Operational data sits in databases behind business capabilities served with microservices, has a transactional nature, keeps the current state and serves the needs of the applications running the business. Analytical data is a temporal and aggregated view of the facts of the business over time, often modeled to provide retrospective or future-perspective insights; it trains the ML models or feeds the analytical reports.

The current state of technology, architecture and organization design is reflective of the divergence of these two data planes - two levels of existence, integrated yet separate. This divergence has led to a fragile architecture. Continuously failing ETL (Extract, Transform, Load) jobs and ever growing complexity of a labyrinth of data pipelines, is a familiar sight to many who attempt to connect these two planes, flowing data from operational data plane to the analytical plane, and back to the operational plane.

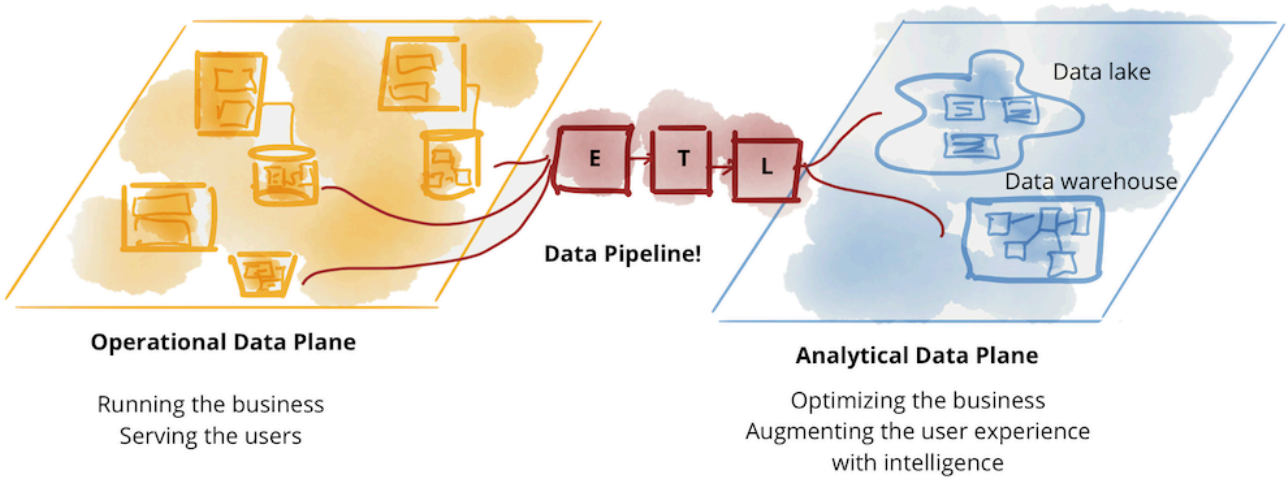


Figure 1: The great divide of data

Analytical data plane itself has diverged into two main architectures and technology stacks: *data lake* and *data warehouse*; with data lake supporting data science access patterns, and data warehouse supporting analytical and business intelligence reporting access patterns. For this conversation, I put aside the dance between the two technology stacks: data warehouse attempting to onboard data science workflows and data lake attempting to serve data analysts and business intelligence. The original writeup on data mesh explores the challenges of the existing analytical data plane architecture.

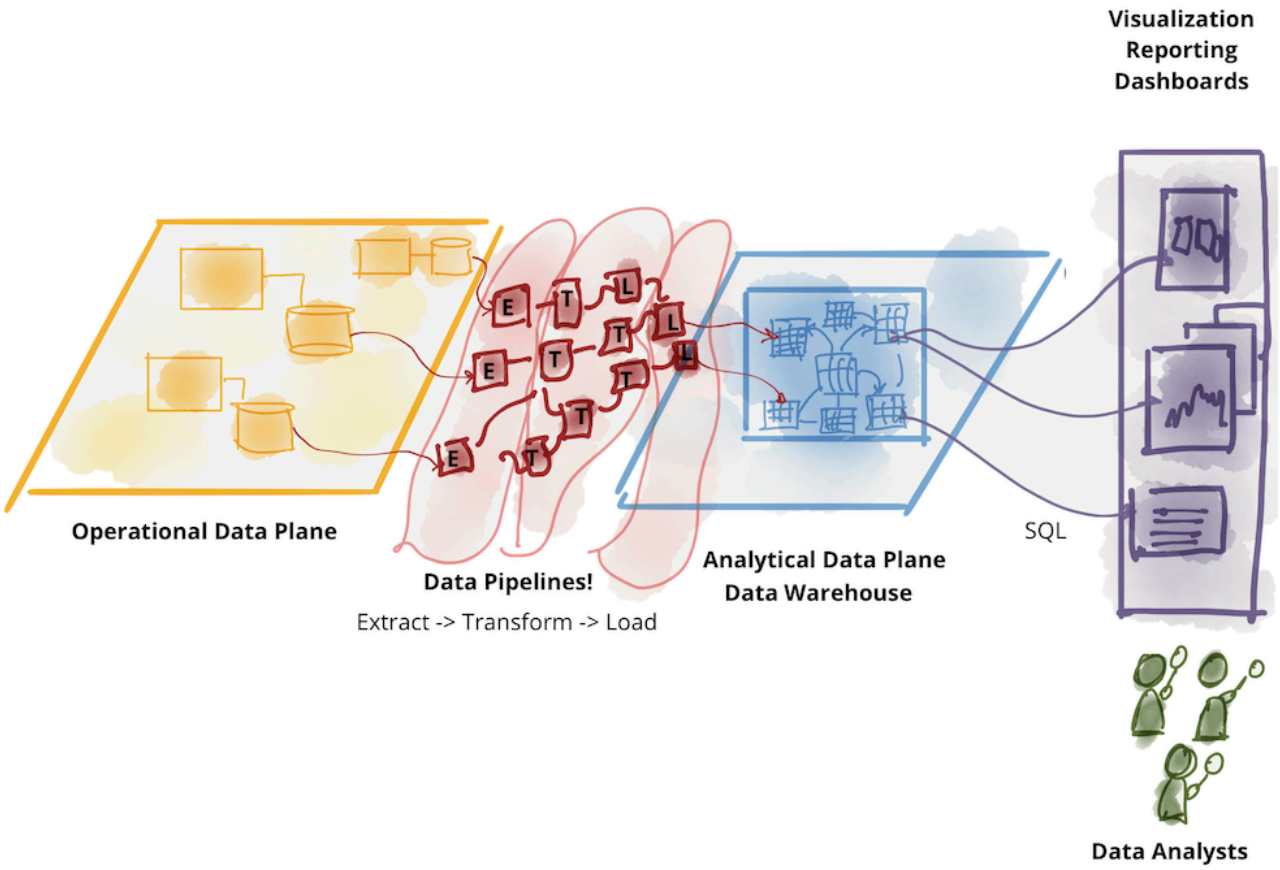


Figure 2: Further divide of analytical data - warehouse

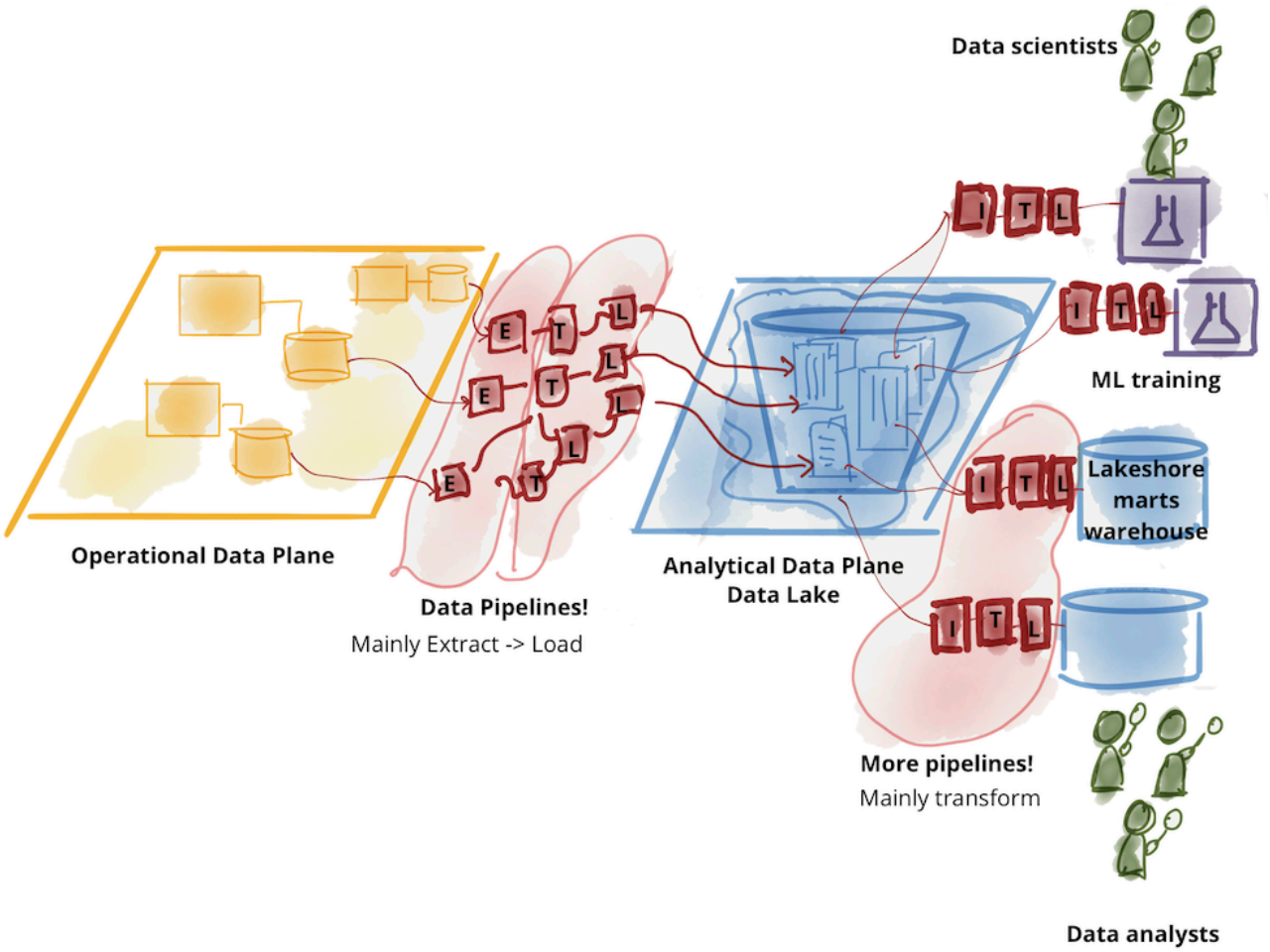


Figure 3: Further divide of analytical data - lake

Data mesh recognizes and respects the differences between these two planes: the nature and topology of the data, the differing use cases, individual personas of data consumers, and ultimately their diverse access patterns. However it attempts to connect these two planes under a different structure - *an inverted model and topology based on domains and not technology stack* - with a focus on the analytical data plane. Differences in today's available technology to manage the two archetypes of data, should not lead to separation of organization, teams and people who work on them. In my opinion, the operational and transactional data technology and topology is relatively mature, and driven largely by the microservices architecture; data is hidden on the inside of each microservice, controlled and accessed through the microserivce's APIs. Yes there is room for innovation to truly achieve multi-cloud-native operational database solutions, but from the architectural perspective it meets the needs of the business. However it's the management and access to the analytical data that remains a point of friction at scale. This is where data mesh focuses.

I do believe that at some point in the future our technologies will evolve to bring these two planes even closer together, but for now, I suggest we keep their concerns separate.

Core principles and logical architecture of data mesh

Data mesh objective is to create a foundation for getting value from analytical data and historical facts at **scale** - scale being applied to *constant change of data landscape*, *proliferation of both sources of data and consumers*, *diversity of transformation and processing that use cases require*, *speed of response to change*. To achieve this objective, I suggest that there are **four underpinning principles** that any data mesh implementation embodies to achieve the promise of scale, while delivering quality and integrity guarantees needed to make data usable : 1) domain-oriented decentralized data ownership and architecture, 2) data as a product, 3) self-serve data infrastructure as a platform, and 4) federated computational governance.

While I expect the practices, technologies and implementations of these principles vary and mature over time, these principles remain unchanged.

I have intended for the four principles to be *collectively necessary and sufficient*; to enable scale with resiliency while addressing concerns around siloing of incompatible data or increased cost of operation. Let's dive into each principle and then design the conceptual architecture that supports it.

Domain Ownership

Data mesh, at core, is founded in *decentralization* and *distribution of responsibility* to people who are closest to the data in order to support continuous change and scalability. The question is, how do we decompose and decentralize the components of the data ecosystem and their ownership. The components here are made of *analytical data*, its *metadata*, and the *computation* necessary to serve it.

Data mesh follows the seams of organizational units as the axis of decomposition. Our organizations today are decomposed based on their business domains. Such decomposition localizes the impact of continuous change and evolution - for the most part - to the domain's bounded context. Hence, making the business domain's bounded context a good candidate for distribution of data ownership.

In this article, I will continue to use the same use case as the original writeup, 'a digital media company'. One can imagine that the media company divides its operation, hence the systems and teams that support the operation, based on domains such as

'podcasts', teams and systems that manage podcast publication and their hosts; 'artists', teams and systems that manage onboarding and paying artists, and so on. Data mesh argues that the ownership and serving of the analytical data should respect these domains. For example, the teams who manage 'podcasts', while providing APIs for releasing podcasts, should also be responsible for providing historical data that represents 'released podcasts' over time with other facts such as 'listenership' over time. For a deeper dive into this principle see Domain-oriented data decomposition and ownership.

Logical architecture: domain-oriented data and compute

To promote such decomposition, we need to model an architecture that arranges the analytical data by domains. In this architecture, the domain's interface to the rest of the organization not only includes the operational capabilities but also access to the analytical data that the domain serves. For example, 'podcasts' domain provides operational APIs to 'create a new podcast episode' but also an analytical data endpoint for retrieving 'all podcast episodes data over the last <n> months'. This implies that the architecture must remove any friction or coupling to let domains serve their analytical data and release the code that computes the data, independently of other domains. To scale, the architecture must support autonomy of the domain teams with regard to the release and deployment of their operational or analytical data systems.

The following example demonstrates the principle of domain oriented data ownership. The diagrams are only logical representations and exemplary. They aren't intended to be complete.

Each domain can expose one or many operational APIs, as well as one or many analytical data endpoints

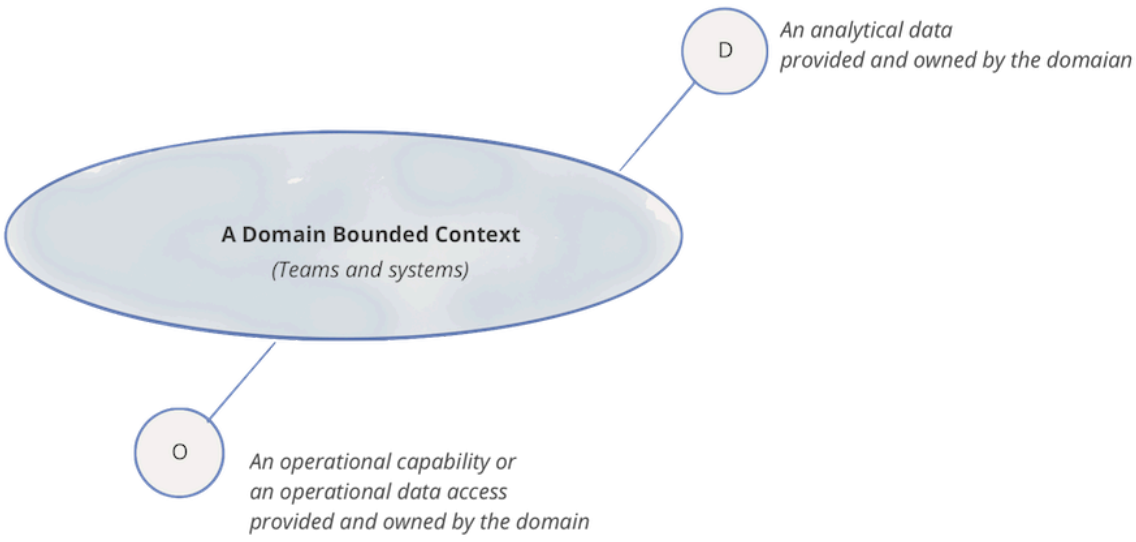


Figure 4: Notation: domain, its analytical data and operational capabilities

Naturally, each domain can have dependencies to other domains' operational and analytical data endpoints. In the following example, 'podcasts' domain consumes analytical data of 'users updates' from the 'users' domain, so that it can provide a picture of the demographic of podcast listeners through its 'Podcast listeners demographic' dataset.

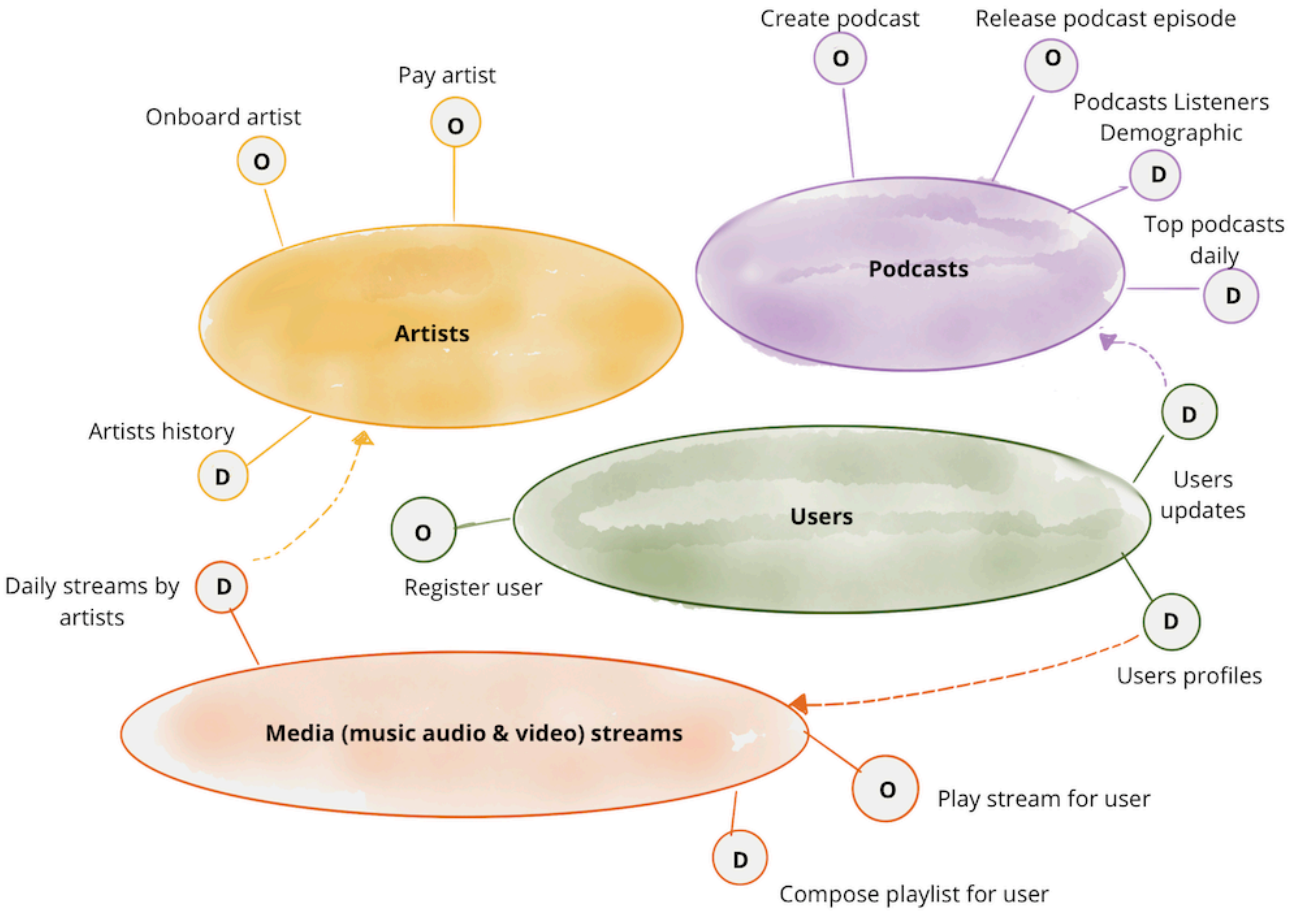


Figure 5: Example: domain oriented ownership of analytical data in addition to operational capabilities

Note: In the example, I have used an imperative language for accessing the operational data or capabilities, such as 'Pay artists'. This is simply to emphasize the difference between the intention of accessing operational data vs. analytical data. I do recognize that in practice operational APIs are implemented through a more declarative interface such as accessing a RESTful resource or a GraphQL query.

Data as a product

One of the challenges of existing analytical data architectures is the high friction and cost of *discovering, understanding, trusting, and ultimately using quality data*. If not addressed, this problem only exacerbates with data mesh, as the number of places and teams who provide data - domains - increases. This would be the consequence of our first principle of decentralization. Data as a product principle is designed to address the data quality and age-old data silos problem; or as Gartner calls it dark data - "the information assets organizations collect, process and store during regular business activities, but generally fail to use for other purposes". Analytical data provided by the domains must be treated as a product, and the consumers of that data should be treated as customers - happy and delighted customers.

The original article enumerates a list of capabilities, including *discoverability*, *security*, *explorability*, *understandability*, *trustworthiness*, etc., that a data mesh implementation should support for a domain data to be considered a product. It also details the roles such as **domain data product owner** that organizations must introduce, responsible for the objective measures that ensure data is delivered as a product. These measures include *data quality*, *decreased lead time* of data consumption, and in general *data user satisfaction* through net promoter score. Domain data product owner must have a deep understanding of who the data users are, how do they use the data, and what are the native methods that they are comfortable with consuming the data. Such intimate knowledge of data users results in design of data product interfaces that meet their needs. In reality, for the majority of data products on the mesh, there are a few conventional personas with their unique tooling and expectations, data analysts and data scientists. All data products can develop standardized interfaces to support them. The conversation between users of the data and product owners is a necessary piece for establishing the interfaces of data products.

Each domain will include **data product developer roles**, responsible for building, maintaining and serving the domain's data products. Data product developers will be working alongside other developers in the domain. Each domain team may serve one or multiple data products. It's also possible to form new teams to serve data products that don't naturally fit into an existing operational domain.

Note: this is an inverted model of responsibility compared to past paradigms. The accountability of data quality shifts upstream as close to the source of the data as possible.

Logical architecture:data product the architectural quantum

Architecturally, to support data as a product that domains can autonomously serve or consume, data mesh introduces the concept of **data product** as its **architectural quantum**. Architectural quantum, as defined by Evolutionary Architecture, is the

smallest unit of architecture that can be independently deployed with high functional cohesion, and includes all the structural elements required for its function.

Data product is the node on the mesh that encapsulates three structural components required for its function, providing access to the domain's analytical data as a product.

- **Code**: it includes (a) code for data pipelines responsible for consuming, transforming and serving upstream data - data received from domain's operational system or an upstream data product; (b) code for APIs that provide access to data, semantic and syntax schema, observability metrics and other metadata; (c) code for enforcing traits such as access control policies, compliance, provenance, etc.
- **Data and Metadata**: well that's what we are all here for, the underlying analytical and historical data in a polyglot form. Depending on the nature of the domain data and its consumption models, data can be served as events, batch files, relational tables, graphs, etc., while maintaining the same semantic. For data to be usable there is an associated set of metadata including data computational documentation, semantic and syntax declaration, quality metrics, etc; metadata that is intrinsic to the data e.g. its semantic definition, and metadata that communicates the traits used by computational governance to implement the expected behavior e.g. access control policies.
- **Infrastructure**: The infrastructure component enables building, deploying and running the data product's code, as well as storage and access to big data and metadata.

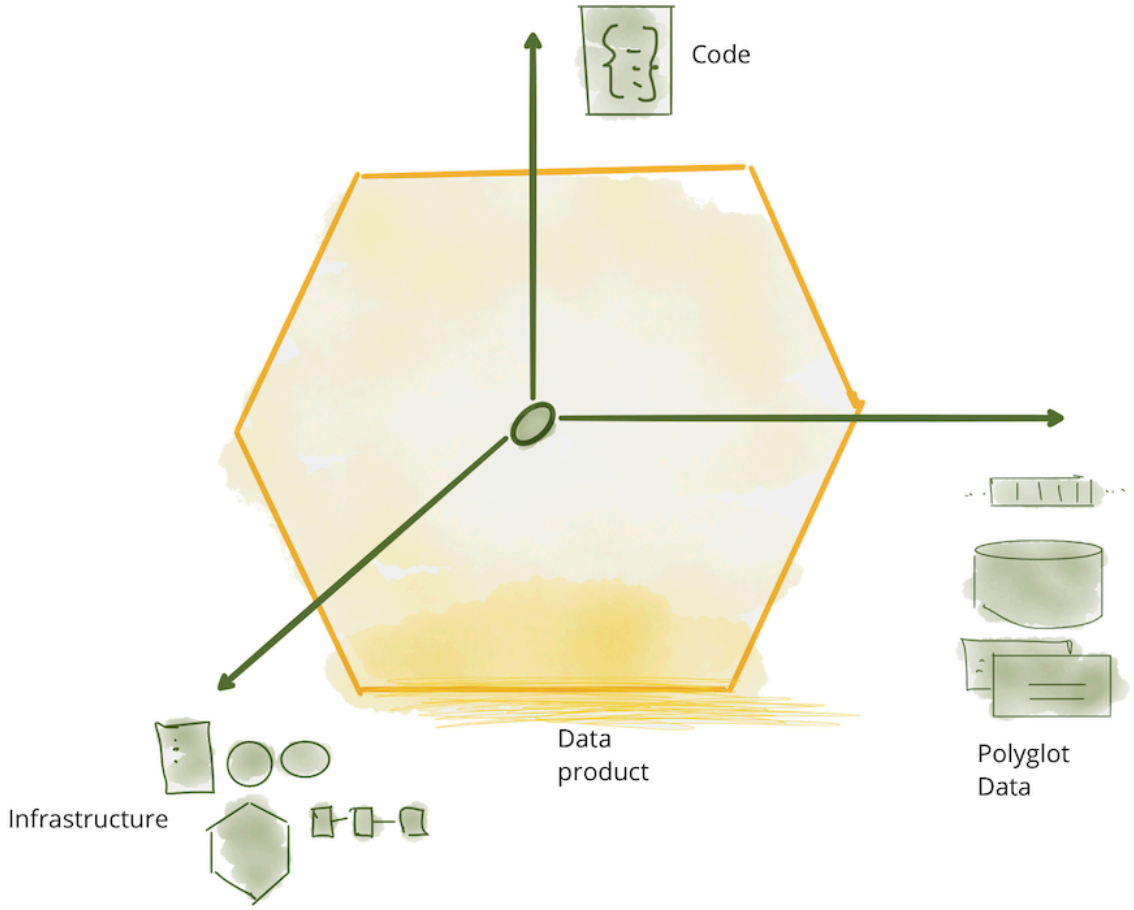


Figure 6: Data product components as one architectural quantum

The following example builds on the previous section, demonstrating the data product as the architectural quantum. The diagram only includes sample content and is not intended to be complete or include all design and implementation details. While this is still a logical representation it is getting closer to the physical implementation.

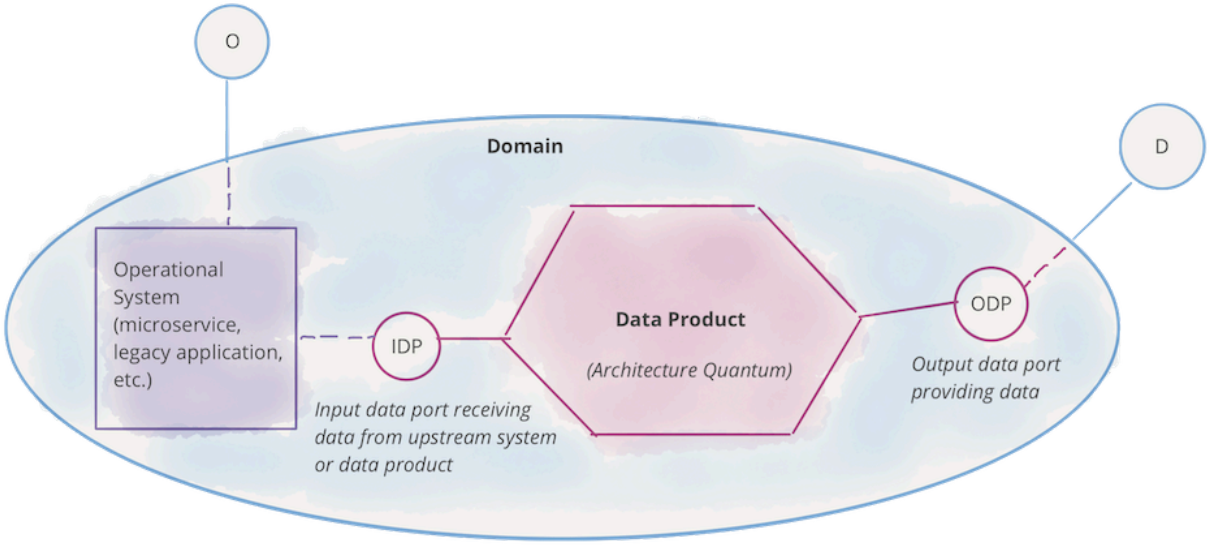


Figure 7: Notation: domain, its (analytical) data product and operational system

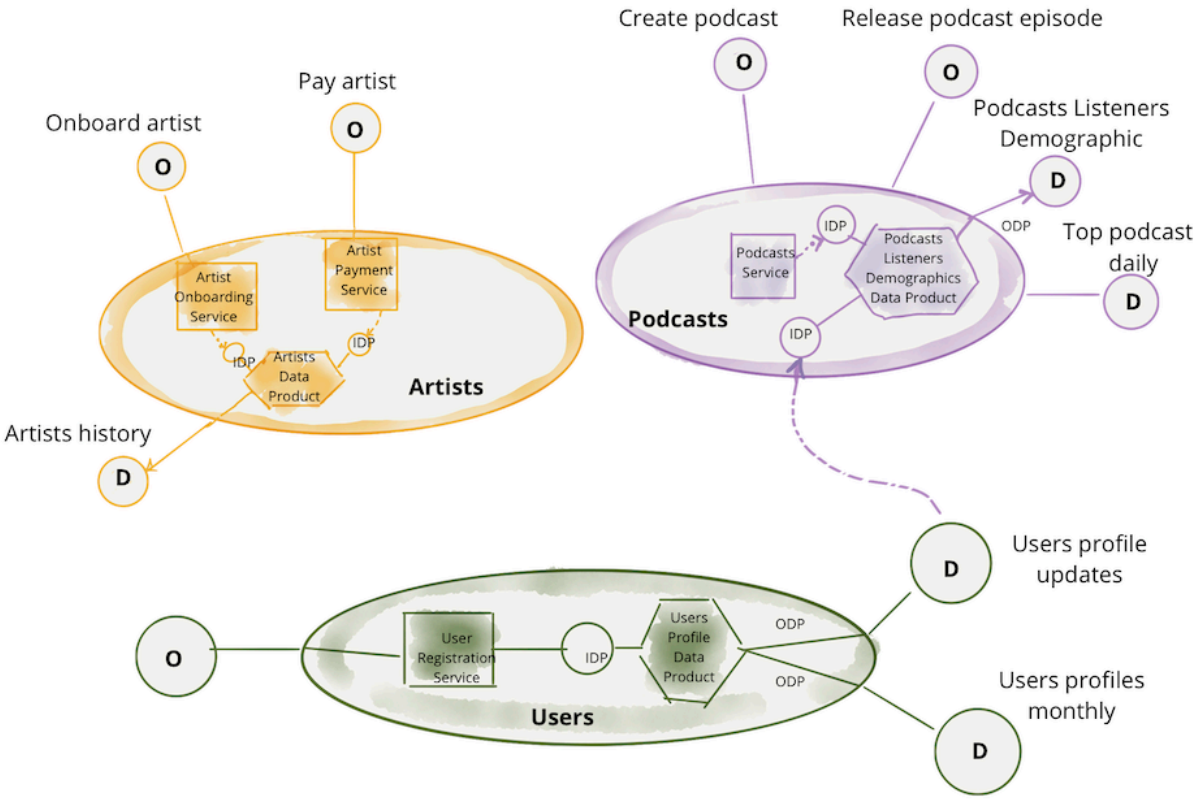


Figure 8: Data products serving the domain-oriented analytical data

Note: Data mesh model differs from the past paradigms where pipelines (code) are managed as independent components from the data they produce; and often infrastructure, like an instance of a warehouse or a lake storage account, is shared among many datasets. Data product is a composition of all components - code, data and infrastructure - at the granularity of a domain's bounded context.

Self-serve data platform

As you can imagine, to build, deploy, execute, monitor, and access a humble hexagon - a data product - there is a fair bit of infrastructure that needs to be provisioned and run; the skills needed to provision this infrastructure are specialized and would be difficult to replicate in each domain. Most importantly, the only way that teams can autonomously own their data products is to have access to a high-level abstraction of

infrastructure that removes complexity and friction of provisioning and managing the lifecycle of data products. This calls for a new principle, *Self-serve data infrastructure as a platform to enable domain autonomy*.

The data platform can be considered an extension of the delivery platform that already exists to run and monitor the services. However the underlying technology stack to operate data products, today, looks very different from the delivery platform for services. This is simply due to divergence of big data technology stacks from operational platforms. For example, domain teams might be deploying their services as Docker containers and the delivery platform uses Kubernetes for their orchestration; However the neighboring data product might be running its pipeline code as Spark jobs on a Databricks cluster. That requires provisioning and connecting two very different sets of infrastructure, that prior to data mesh did not require this level of interoperability and interconnectivity. My personal hope is that we start seeing a convergence of operational and data infrastructure where it makes sense. For example, perhaps running Spark on the same orchestration system, e.g. Kubernetes.

In reality, to make analytical data product development accessible to generalist developers, to the existing profile of developers that domains have, the self-serve platform needs to provide a new category of tools and interfaces in addition to simplifying provisioning. A self-serve data platform must create tooling that supports a domain data product developer's workflow of creating, maintaining and running data products with less specialized knowledge that existing technologies assume; self-serve infrastructure must include capabilities to lower the current cost and specialization needed to build data products. The original writeup includes a list of capabilities that a self-serve data platform provides, including access to *scalable polyglot data storage*, *data products schema*, *data pipeline declaration and orchestration*, *data products lineage*, *compute and data locality*, etc.

Logical architecture: a multi-plane data platform

The self-serve platform capabilities fall into multiple categories or planes as called in the model. Note: A plane is representative of a level of existence - integrated yet separate. Similar to physical and consciousness planes, or control and data planes in networking. A plane is neither a layer and nor implies a strong hierarchical access model.

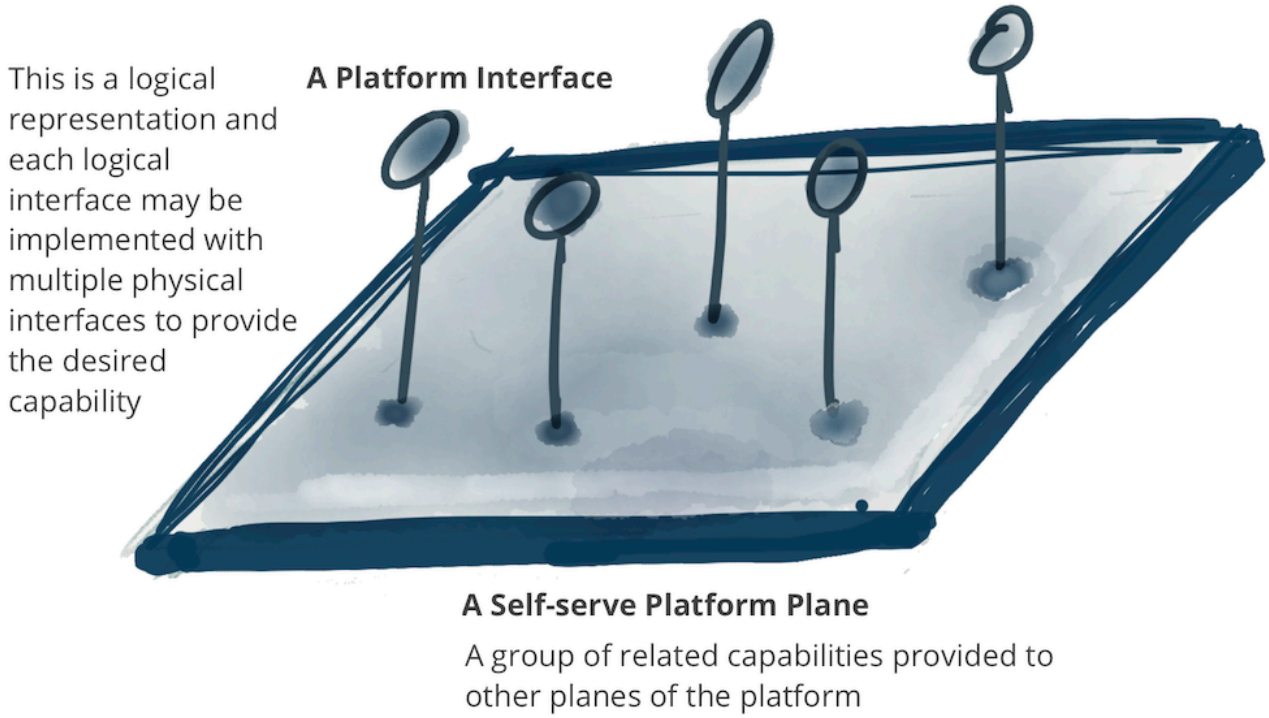


Figure 9: Notation: A platform plane that provides a number of related capabilities through self-serve interfaces

A self-serve platform can have multiple planes that each serve a different profile of users. In the following example, lists three different data platform planes:

- **Data infrastructure provisioning plane**: supports the provisioning of the underlying infrastructure, required to run the components of a data product and the mesh of products. This includes provisioning of a distributed file storage, storage accounts, access control management system, the orchestration to run data products internal code, provisioning of a distributed query engine on a graph of data products, etc. I would expect that either other data platform planes or only advanced data product developers use this interface directly. This is a fairly low level data infrastructure lifecycle management plane.
- **Data product developer experience plane**: this is the main interface that a typical data product developer uses. This interface abstracts many of the complexities of what entails to support the workflow of a data product developer. It provides a higher level of abstraction than the 'provisioning plane'. It uses simple declarative interfaces to manage the lifecycle of a data product. It automatically implements the cross-cutting concerns that are defined as a set of standards and global conventions, applied to all data products and their interfaces.
- **Data mesh supervision plane**: there are a set of capabilities that are best provided at the mesh level - a graph of connected data products - globally. While the implementation of each of these interfaces might rely on individual data products capabilities, it's more convenient to provide these capabilities at the level of the mesh. For example, ability to discover data products for a particular use case, is

best provided by search or browsing the mesh of data products; or correlating multiple data products to create a higher order insight, is best provided through execution of a data semantic query that can operate across multiple data products on the mesh.

The following model is only exemplary and is not intending to be complete. While a hierarchy of planes is desirable, there is no strict layering implied below.

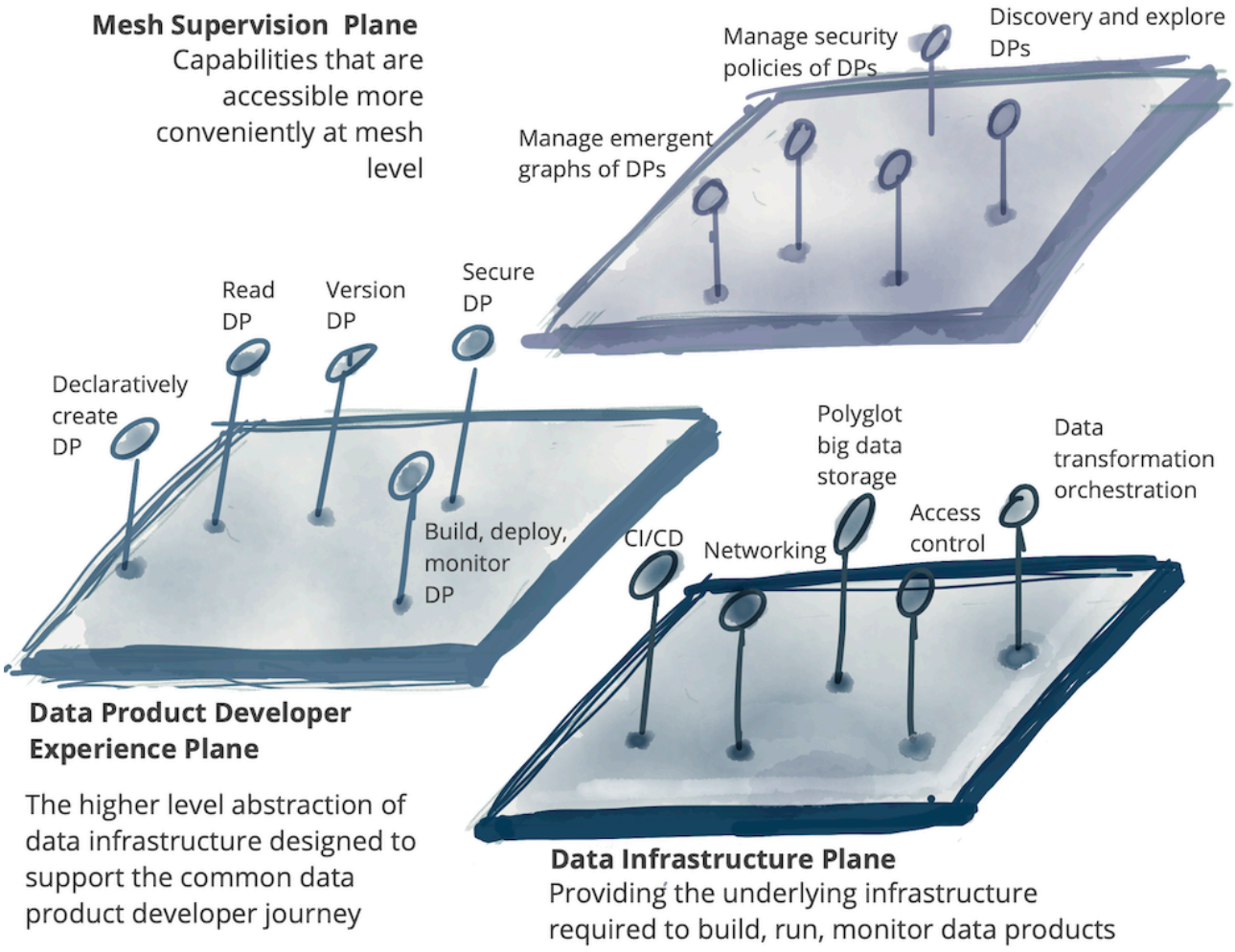


*Figure 10: Multiple planes of self-serve data platform *DP stands for a data product*

Federated computational governance

As you can see, data mesh follows a distributed system architecture; a collection of independent data products, with independent lifecycle, built and deployed by likely independent teams. However for the majority of use cases, to get value in forms of higher order datasets, insights or machine intelligence there is a need for these

independent data products to interoperate; to be able to correlate them, create unions, find intersections, or perform other graphs or set operations on them at scale. For any of these operations to be possible, a data mesh implementation requires a governance model that embraces *decentralization and domain self-sovereignty*, *interoperability through global standardization*, *a dynamic topology* and most importantly *automated execution of decisions by the platform*. I call this a federated computational governance. A decision making model led by the federation of domain data product owners and data platform product owners, with autonomy and domain-local decision making power, while creating and adhering to a set of global rules - rules applied to all data products and their interfaces - to ensure a healthy and interoperable ecosystem. The group has a difficult job: maintaining an *equilibrium between centralization and decentralization*; what decisions need to be localized to each domain and what decisions should be made globally for all domains. Ultimately global decisions have one purpose, creating *interoperability* and a *compounding network effect* through discovery and composition of data products.

The priorities of the governance in data mesh are different from traditional governance of analytical data management systems. While they both ultimately set out to get value from data, traditional data governance attempts to achieve that through centralization of decision making, and establishing global canonical representation of data with minimal support for change. Data mesh's federated computational governance, in contrast, embraces change and multiple interpretive contexts.

> *Placing a system in a straitjacket of constancy can cause fragility to evolve.*
>
> *-- C.S. Holling, ecologist*

Logical architecture: computational policies embedded in the mesh

A supportive *organizational structure*, *incentive model* and *architecture* is necessary for the federated governance model to function: to arrive at global decisions and standards for interoperability, while respecting autonomy of local domains, and implement global policies effectively.

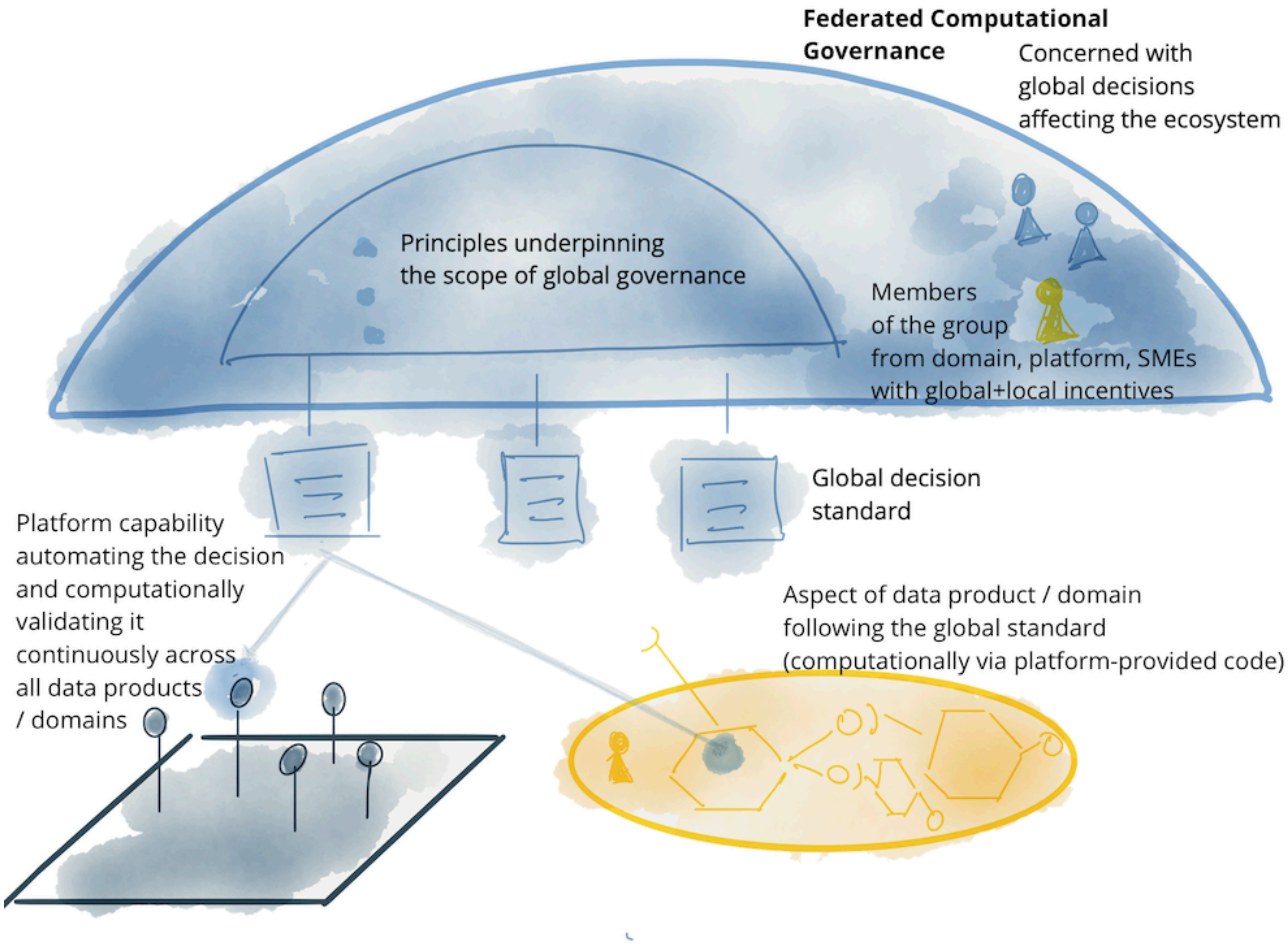


Figure 11: Notation: federated computational governance model

As mentioned earlier, striking a balance between what shall be standardized globally, implemented and enforced by the platform for all domains and their data products, and what shall be left to the domains to decide, is an art. For instance the domain data model is a concern that should be localized to a domain who is most intimately familiar with it. For example, how the semantic and syntax of 'podcast audienceship' data model is defined must be left to the 'podcast domain' team. However in contrast, the decision around how to identify a 'podcast listener' is a global concern. A podcast listener is a member of the population of 'users' - its upstream bounded context - who can cross the boundary of domains and be found in other domains such as 'users play streams'. The unified identification allows correlating information about 'users' who are both 'podcast listeners' and 'stream listeners'.

The following is an example of elements involved in the data mesh governance model. It's not a comprehensive example and only demonstrative of concerns relevant at the global level.

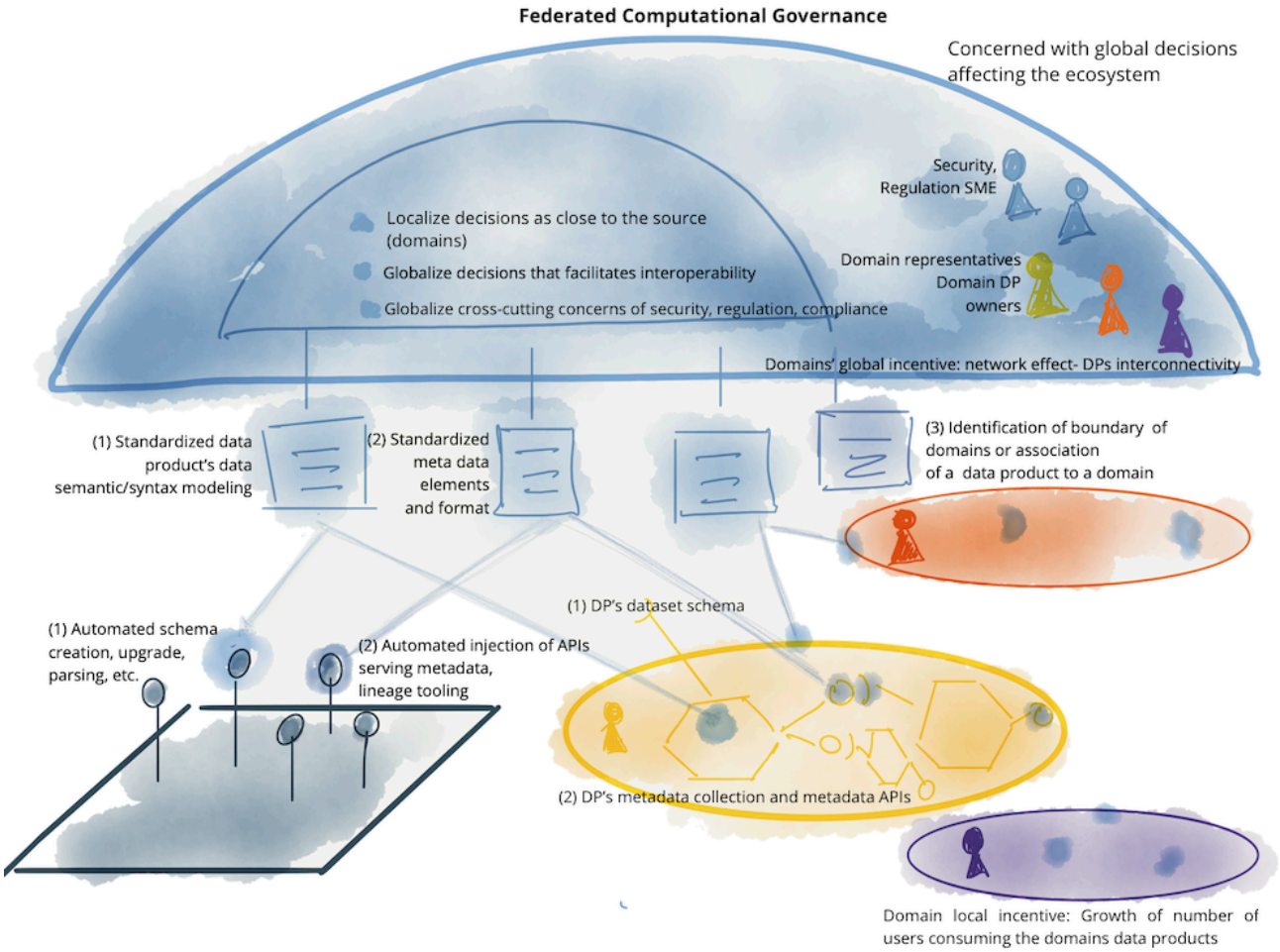


Figure 12: : Example of elements of a federated computational governance: teams, incentives, automated implementation, and globally standardized aspects of data mesh

Many practices of pre-data-mesh governance, as a centralized function, are no longer applicable to the data mesh paradigm. For example, the past emphasis on certification of golden datasets - the datasets that have gone through a centralized process of quality control and certification and marked as trustworthy - as a central function of governance is no longer relevant. This had stemmed from the fact that in the previous data management paradigms, data - in whatever quality and format - gets extracted from operational domain's databases and gets centrally stored in a warehouse or a lake that now requires a centralized team to apply cleansing, harmonization and encryption processes to it; often under the custodianship of a centralized governance group. Data mesh completely decentralizes this concern. A domain dataset only becomes a data product after it locally, within the domain, goes through the process of quality assurance according to the expected data product quality metrics and the global standardization rules. The domain data product owners are best placed to decide how to measure their domain's data quality knowing the details of domain operations producing the data in the first place. Despite such localized decision making and autonomy, they need to comply with the modeling of quality and specification of SLOs based on a global standard, defined by the global federated governance team, and automated by the platform.

The following table shows the contrast between centralized (data lake, data warehouse) model of data governance, and data mesh.

Pre data mesh governance aspect	Data mesh governance aspect
Centralized team	Federated team
Responsible for data quality	Responsible for defining how to model what constitutes quality
Responsible for data security	Responsible for defining aspects of data security i.e. data sensitivity levels for the platform to build in and monitor automatically
Responsible for complying with regulation	Responsible for defining the regulation requirements for the platform to build in and monitor automatically
Centralized custodianship of data	Federated custodianship of data by domains
Responsible for global canonical data modeling	Responsible for modeling polysemes - data elements that cross the boundaries of multiple domains
Team is independent from domains	Team is made of domains representatives
Aiming for a well defined static structure of data	Aiming for enabling effective mesh operation embracing a continuously changing and a dynamic topology of the mesh
Centralized technology used by monolithic lake/warehouse	Self-serve platform technologies used by each domain
Measure success based on number or volume of governed data (tables)	Measure success based on the network effect - the connections representing the consumption of data on the mesh
Manual process with human intervention	Automated processes implemented by the platform
Prevent error	Detect error and recover through platform's automated processing

Principles Summary and the high level logical architecture

Let's bring it all together, we discussed four principles underpinning data mesh:

Domain-oriented decentralized data ownership and architecture	**So that** the ecosystem creating and consuming data can scale out as the number of sources of data, number of use cases, and diversity of access models to the data increases; simply increase the autonomous nodes on the mesh.
Data as a product	**So that** data users can easily discover, understand and securely use high quality data with a delightful experience; data that is distributed across many domains.
Self-serve data infrastructure as a platform	**So that** the domain teams can create and consume data products autonomously using the platform abstractions, hiding the complexity of building, executing and maintaining secure and interoperable data products.
Federated computational governance	**So that** data users can get value from aggregation and correlation of independent data products - the mesh is behaving as an ecosystem following global interoperability standards; standards that are baked computationally into the platform.

These principles drive a logical architectural model that while bringing analytical data and operational data closer together under the same domain, it respects their underpinning technical differences. Such differences include where the analytical data might be hosted, different compute technologies for processing operational vs. analytical services, different ways of querying and accessing the data, etc.

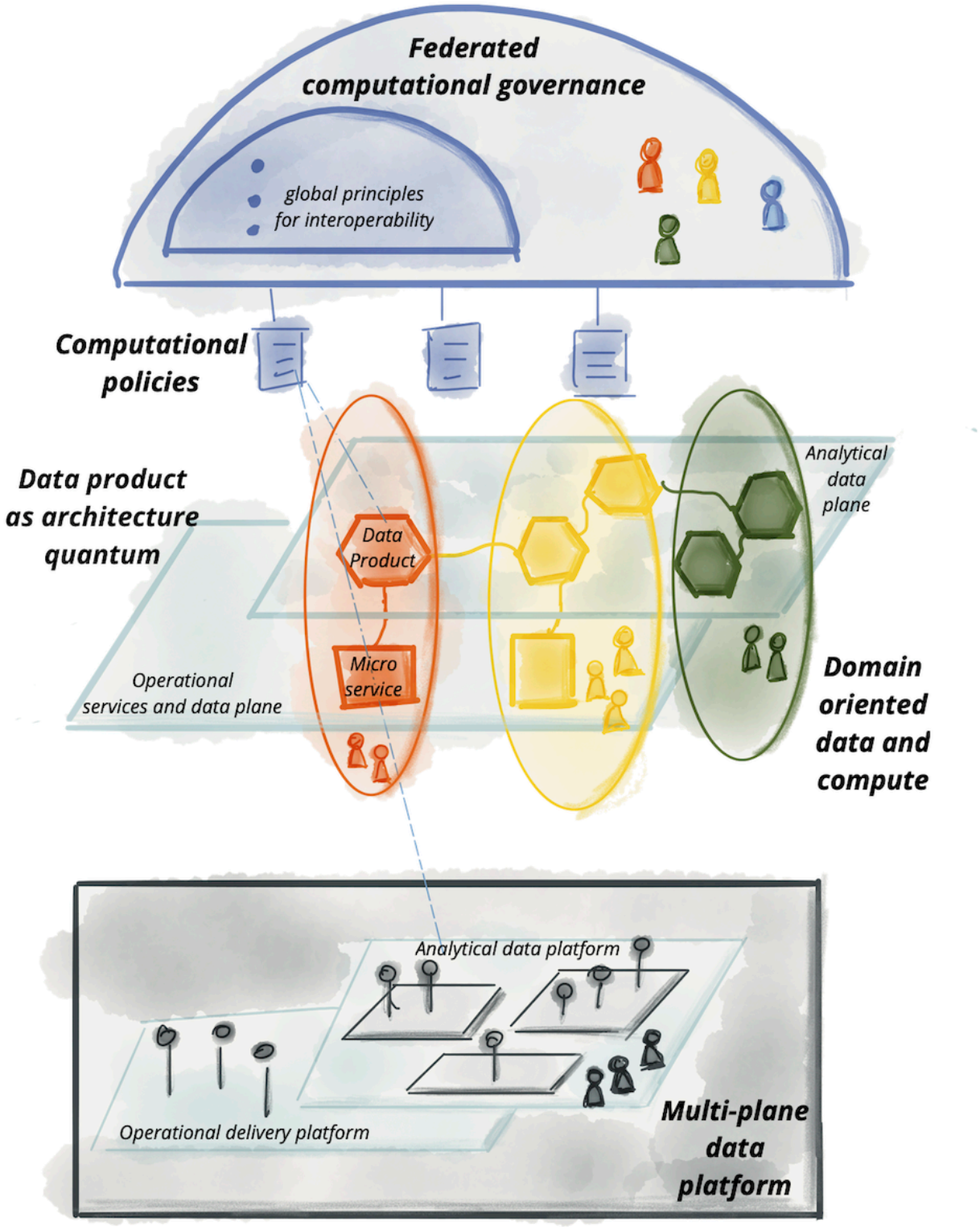


Figure 13: Logical architecture of data mesh approach

I hope by this point, we have now established a common language and a logical mental model that we can collectively take forward to detail the blueprint of the components of the mesh, such as the data product, the platform, and the required standardizations.

Acknowledgments

I am grateful to Martin Fowler for helping me refine the narrative and structure of this article, and for hosting it.

Special thanks to many ThoughtWorkers who have been helping create and distill the ideas in this article through client implementations and workshops.

Also thanks to the following early reviewers who provided invaluable feedback: Chris Ford, David Colls and Pramod Sadalage.

▶ Significant Revisions