POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Data Friction in Data Sharing: a Physics Inspired Model

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Giacomo Lombardo**

Student ID: 996379
Advisor: Prof. Pierluigi Plebani
Co-advisors: Matteo Falconi
Academic Year: 2022-23

Abstract

In the era of Big Data, the increasing volumes of generated and collected data represent a key resource for organizations to create economic value. Data sharing has become a pivotal factor to drive progress, especially in areas where data availability is low and regulatory frameworks heavily constrain data movement.

Researchers have begun to identify Data Friction all those factors that impede the natural movement of data, and to study how to overcome it in a data exchange. However, as organizations try to scale along with data volumes, traditional data architectures fail to capture the value of data assets.

In this context, the data mesh paradigm advocates for the decentralization of data ownership and management to maximize value generation through data sharing. Yet, as the data mesh paradigm promises to reduce frictions in data management, it lacks an effective method to quantify and overcome data friction in data exchanges.

This thesis proposes a model to structure data exchanges, introducing a method to quantify and overcome their inherent data friction. This model is then applied in the data mesh platform architecture to provide a friction-aware approach to data sharing in such a context. As data sharing ecosystems become increasingly relevant, this contribution marks a first step in modeling and quantifying data friction in data exchanges.

Keywords: Data Sharing, Data Exchanges, Data Friction, Data Mesh

Abstract in lingua italiana

Nell'era dei Big Data, i crescenti volumi di dati generati e raccolti rappresentano per le organizzazioni una risorsa fondamentale per creare valore economico. La condivisione dei dati è diventata un fattore cruciale per il progresso tecnologico, soprattutto in quelle aree in cui la disponibilità dei dati è bassa e la loro circolazione è fortemente limitata da quadri normativi.

I ricercatori hanno iniziato a identificare come Data Friction tutti quei fattori che ostacolano la naturale circolazione dei dati, e astudiare come superarli in uno scambio degli stessi. Tuttavia, anche se le organizzazioni provano a utilizzare i crescenti volumi di dati per spingere la propria crescita, le architetture di dati tradizionali non riescono a catturare e rendere disponibile il loro valore.

In questo contesto, il paradigma del data mesh propone la decentralizzazione della proprietà e della gestione dei dati per massimizzare la generazione di valore attraverso la loro condivisione. Tuttavia, sebbene il paradigma del data mesh prometta di ridurre gli attriti nella gestione dei dati, manca di un metodo efficace per quantificare e superare questi attriti negli scambi.

Questa tesi propone un modello per strutturare gli scambi di dati, fornendo un approccio strutturato per quantificare e superare i loro attriti intrinseci. Questo modello viene poi applicato all'architettura della piattaforma data mesh per fornire un approccio consapevole degli attriti nella condivisione dei dati in un contesto di questo tipo. Con gli ecosistemi di condivisione dei dati che diventano sempre più importanti, questo contributo segna un primo passo nella creazione di un modello per quantificare l'attrito generato in uno scambio di dati.

Parole chiave: Condivisione dei dati, Scambi di dati, Data Friction, Data Mesh

Contents

1 | Introduction

As the amount of generated data increases daily, today's economy revolves more and more around the value that can be generated from it. With the advent of *Big Data*, large volumes of data can be generated, processed, and analyzed to obtain precious insights on potentially every business aspect of an organization.

In today's *Data Economy*, organizations revise and re-adapt their original business models and processes to effectively capture data-generated value and make data-driven decisions to maximize their efficiency. It is no surprise that today's biggest and most successful organizations are *digital-native*, such as the FAANG (Facebook, Apple, Amazon, Netflix, Google) companies, with data-centric business models enabling quick scaling and continuous innovation. These organizations are able to effectively collect data and extract the most value from it, supporting their strategic planning with data-driven insights and decisions. Multi-sided platforms such as Facebook and Amazon are constantly incorporating data-driven solutions to maximize their profits. Netflix's series "House of Cards" was conceived by analyzing users' data about viewing preferences, making it one of the most successful series ever even before its launch [11]. On the other hand, industry incumbents are revising their traditional business models towards more data-centered ones to survive in the current economy. If a company does not effectively leverage its data assets, it is doomed to fail in today's competitive landscape.

Not only organizations can use data to improve their own businesses, but, just like every other asset, data can be shared and exchanged with other players to maximize the exploitation of its inherent value. This is the case of Strava, a popular fitness app, and its commercial data service Strava Metro. Strava Metro aggregates, de-identifies and contextualizes data generated by Strava users and sells it to municipalities and researchers to study mobility patterns and evaluate the impact of urban planning initiatives [30]. On the same page, Uber Technologies, Inc. launched a similar service to help urban planners, leveraging the large amounts of data collected daily through the company's operations [22].

Data sharing can be beneficial not only for data-driven businesses but also in the scientific

community, where, oftentimes, the availability of data represents a problem for researchers trying to validate their studies. The presence of data sharing platforms could significantly speed up scientific progress in research areas characterized by low data availability and highly restricted data access, such as healthcare studies.

As the amounts of generated data increase exponentially, as well as the data sharing opportunities that come with them, we are often brought to think that data *flows* effortlessly from device to device, traversing databases and servers in a continuous cycle of processing and value generation. However, this is not true. As Borgman observes, "Data do not flow like oil" [8]. Rather, the flow of data is influenced by a complex web of technical, social, and institutional factors, and it often encounters significant barriers and challenges along the way. A series of efforts is needed to let the data flow effectively, supporting the value generation during the data lifecycle.

In this sense, we can draw a parallel with physics. Objects do not move by themselves, and a force needs to be applied to enable the movement of an object, overcoming friction. Just like that, data movement is opposed by the presence of *data friction*, and external forces are needed to enable it. Data friction depends on various factors such as the characteristics of communication channels, data quality metrics, and regulatory constraints. The higher the data friction, the harder it is to successfully move and exchange data among different actors.

In the Big Data era, data is being generated at extremely high volumes and speed from heterogeneous sources and in heterogeneous formats. Along traditional and structured formats, nowadays, data is getting generated and ingested in semi-structured and unstructured formats, favoring speed and volume over standardization and re-usability. The consequence of this trend is that data friction is becoming increasingly prominent in online communications as well as in the scientific world. To overcome the barriers and challenges derived by data friction, professional figures such as data engineers, data scientists, and data analysts are becoming crucial for businesses to thrive in today's Data Economy.

In order to process data and extract value from it, adequate IT infrastructures are needed, too. Traditional business intelligence (BI) components such as *data warehouses* fail to capture the heterogeneous nature of big data, as well as requisites such as speed and real-time processing and analysis. In the last decade, new solutions, such as *data lakes*, emerged to overcome these challenges. Data lakes can handle both structured and unstructured data, storing data as-is and in high volumes. However, this solution has its downsides as well. Without effective data governance, Data lakes can easily turn into *data swamps*, poorly maintained data repositories where data is hardly accessible and processable. In

a data swamp, the movement of data and the consequent value generation is impeded by poor data management practices leading to high levels of data friction. Furthermore, data lakes' centralized approach is getting questioned as data volumes increase exponentially and data engineers become overwhelmed by data sources and data consumers. The inability to scale along with the organization and its business needs leads to increasing amounts of data friction in state-of-the-art data architectures, making it harder for organizations to implement effective data-driven projects and decisions in their businesses.

To address these challenges, a new paradigm for data management has been proposed: the *data mesh* [14]. The data mesh principles advocate for the decentralization of data management and data ownership, reorganizing the data repositories in *data products* closer to data sources or data consumers. By decentralizing data ownership, data engineering teams can work on more specialized data repositories, ensuring higher levels of data quality and data governance. This results in a data architecture where data is more accessible and managed effectively, ultimately reducing friction and allowing organizations to make successful data-driven decisions and thrive in the Big Data era.

While the data mesh paradigm offers a novel approach to data management, reducing data friction and promising maximized value generation, it lacks a structured approach to set up data exchanges between data products, defining data exchange key components and steps. Moreover, at the state of the art, there is no model that defines and quantifies data friction in a data exchange. Such a model, in a decentralized system like the data mesh, would be beneficial in estimating the efforts needed to overcome data frictions and effectively exchange data maximizing value generation.

The contribution of this thesis is twofold. First, we structure the data exchange process and propose a method to calculate the data friction in a data exchange and the effort needed to overcome it. Second, we apply the defined model to the data mesh paradigm to quantify and reduce the data friction in data exchanges between data products and data consumers. We then implement a simple proof of concept to verify the validity of our model and to show a possible application of it in the data mesh. Finally, we draw conclusions and identify research directions for future work on the topic.

This thesis contributes to the TEADAL project[1], an European project developed under the Horizon Europe work programme, and it is structured as follows:

- Chapter 1 introduces the topics addressed in this thesis;
- Chapter 2 presents the state of the art on the topics of data friction and data mesh;

[1] https://www.teadal.eu

- Chapter 3 provides an overview of the current state of data sharing in the data mesh paradigm;
- Chapter 4 presents our data exchange model, our definition of data friction, and how to calculate it in a data exchange;
- Chapter 5 applies the defined model to data exchanges in the data mesh;
- Chapter 6 presents an implementation to verify the model's validity;
- Chapter 7 outlines our conclusions, summarizing the achieved goals and identifying future works on the topic.

2 | Related Work

This chapter aims to analyze the state of the art of the two main topics covered in this thesis: *data Friction* and *data mesh*. By doing so, we establish a common ground upon which we will base our observations and hypotheses.

2.1. Data Friction

First introduced by Paul Edwards in [18], the term *data friction* refers to "the costs in time, energy, and attention required simply to collect, check, store, move, receive, and access data". Similarly to what happens in physics, data friction arises at the interface of two "surfaces", i.e. two points where data is moving (e.g. an IoT sensor to a computer). Data friction restricts and impedes the natural movement of data and requires costs and effort to overcome it. Data friction arises when data is not collected and managed properly, mainly due to low level of data governance and data curation. Poorly managed, low-quality data leads to the inability to use and reuse it effectively, ultimately resulting in friction when it comes to data sharing.

Edwards et al. [19] observe how data friction leads to *science friction*: the challenges and obstacles that arise when two scientific disciplines work on related problems. Edwards et al. [19] propose the interfacing between climatology and weather forecasting as an example to understand these forces. Weather forecasting mostly relies on short-term, accurate data, and makes no use of weeks, let alone years, old climate observations. However, these long-term observations are the basis of climatology studies. Although the interfacing between the two sciences seems trivial, the ever-evolving landscape of weather forecasting, with new practices and measurements over the years, generates data and science friction. As climatologists aggregate and compare data from different years, decades, and centuries, they have to cope with the fact that this data has been generated and managed differently over the years. Different data from different periods (and from different regions, countries, and political forces) generate friction, requiring an effort to make those data interoperable and to ultimately produce effective climatology studies.

According to scientific literature, data sharing in scientific research is subject to high levels of friction. Although data sharing and open data practices are common in fields such as genomics and astronomy [7], in other disciplines such as healthcare [6] and plant science [31] seldom collected data is made available and reusable for other experiments. Data sharing and collaboration in scientific disciplines are enabled by a complex web of technological infrastructure, social and cultural relations, and institutional support [28]. Sources of data and science friction, challenging the flow of data hence its sharing and re-usability, can be observed in each of the three factors mentioned above and in their interplay [4].

From a technological standpoint, data friction is mainly caused by the lack of proper data sharing infrastructure and proper data management practices [4]. Murray-Rust [38] observed how, while technological progress made data collection easier, data sharing and re-usability in the scientific community are hampered by the lack of licenses and standards. Even within the same discipline, unified models and platforms supporting data sharing are often lacking [31]. Edwards et al. [19] observe also how poor metadata practices restrict the movement of data. In a survey about data sharing practices in the scientific community, Tenopir et al. [48] observe how the majority of the respondents do not use metadata standards to describe their data and almost half of the respondents do not use metadata at all. Furthermore, efforts in cost and time to properly curate the collected datasets are often prohibitive for researchers and not covered by the research-funding institutions [48][2].

Data friction in the scientific community is not only caused by technological challenges but also by the socio-cultural context. Especially in multi-disciplinary setups, different research goals and data management practices can lead to friction in collecting and using data, as observed by Borgman et al. [9] when studying the interdependence between science and technology teams in the Center for Embedded Networked Sensing (CENS). According to the authors, oftentimes, the difference between data and context is not so clear and varies depending on the point of view, resulting in divergences in data management practices. Although researchers are wary of the benefits of data sharing, oftentimes they do not share their data or share it with restricted access [48]. Borgman [7] observes four rationales behind data sharing in the scientific community: i) to reproduce or to verify research, ii) to make the results available to the public, iii) to enable others to ask new questions, and iv) to advance the state of research and innovation. While some rationales, such as iv), are in some fields the main force driving data sharing practices, rationales i) and ii) can discourage data sharing as researchers' results are exposed to public scrutiny. Furthermore, as Tenopir et al. [48] survey results show, researchers of-

ten prioritize their publications, fearing that sharing data too soon may result in other research groups "hijacking" their work.

Proper regulatory frameworks are finally key to reducing friction when it comes to data sharing in the scientific community. By addressing legal, ethical, and technical considerations, regulations help minimize friction, fostering a collaborative and transparent research ecosystem. However, the presence of policies standardizing the collection and the sharing of scientific data does not always lead to lower frictions, as authors may not be compliant and the various licenses under which data is shared may not allow data re-use [38].

Bates [4] introduces data friction in online communications, analyzing how the three factors mentioned above differ in this field from the scientific one. As Bates observes [4], in the world of online communications the efforts are shifting from removing frictions to introducing them favoring the privacy and protection of online users. Contrary to scientific data, online data is highly standardized and interoperable, hence subject to low friction when it comes to sharing and reusing it. As internet users become more aware of the possibility of their data being exposed, they are more concerned to share it publicly. This tendency is exacerbated by recent scandals such as the U.S. National Security Agency leaks and Cambridge Analytica. Furthermore, data friction in online communication is increasing with the recent introduction of regulatory frameworks and policies such as the General Data Protection Regulation (GDPR) by the European Union, which aims to regulate the sharing and movement of personal data.

2.2. Data Mesh

The proliferation of digital, interconnected devices resulted in increasingly larger volumes of data generated day by day. In the Big Data era, data are generated by an increasing number of sources, at very high rates, in structured, semi-structured, and unstructured formats. Through accurate collection, processing, and analysis these large volumes of data can produce insights that can drive organizations' decisions and potentially obtain competitive advantage. However, proper management of such volumes of data is an increasing concern among organizations trying to transition into today's data economy. Without a proper organizational data infrastructure and culture, Big Data projects are almost always doomed to fail [39], while a recent survey by MIT Technology Review Insights in collaboration with Databricks highlighted that only 13% of companies excel at delivering their data strategy [26].

Originally, organizations' BI units powered analytics and data-driven decisions through *data warehouses*. Data warehouses are components that ingest data from operational

sources such as traditional DBMSs to obtain analytical insights as a result of Extract-Transform-Load (ETL) operations. Data warehouses usually are integrated with a centralized approach and have a structured schema optimized for analytical operations: as the operational data gets ingested into the data warehouse, it must be transformed to adhere to the repository's schema. However, high storage costs, growing demand for volume and speed of ingestion, as well as the emergence of semi-structured and unstructured data formats in online communications have posed significant challenges to Data Warehouses in the Big Data era.

In response to the need for storing vast amounts of data coming from heterogeneous sources, at the beginning of the 2010s *data lakes* emerged as an alternative to data warehouses. A data lake is a central repository allowing the storage of structured, semi-structured and unstructured data ensuring high ingestion speed and scalability. Unlike data warehouses, data lakes are designed to ingest raw data as-is, without the need for immediate structuring or schema definition. Data lakes are designed to be accessed with various methods such as SQL queries or APIs and are often integrated with Big Data technologies and frameworks such as Apache Hadoop and Apache Spark. While data lakes can store data from heterogeneous sources in a flexible manner, effective data governance and data management practices are crucial to ensure data quality and to support data exploration and analytics. Proper metadata management, data cataloging, and data quality controls are essential to ensure the usefulness and reliability of data stored in a data lake. Without proper data management, Data lakes can turn into *data swamps*: repositories where data becomes disorganized and difficult to manage.

However, centralized data repositories such as data warehouses and data lakes fail to scale as data sources and consumer needs increase. According to Ataei and Litchfield [3], such a proliferation of data sources and data consumers is one of the main threats to the maintainability and scalability of Big Data architectures. As Dehghani observes [14], while data source teams and data consumer teams in organizations are domain-oriented, data engineering teams maintaining data platforms are domain agnostic, often without business and domain knowledge. As data sources increase and data is used in a larger number of use cases, hyper-specialized data engineers cannot deal with the increasing pressures coming both from data sources and data consumers. For this reason, Dehghani advocates for a paradigm shift towards a decentralized approach: the *Data Mesh*.

2.2.1. Data Mesh Principles

According to Dehghani [14–16], data mesh is a socio-technical concept based on four principles that allow organizations to manage data at scale:

- **Domain-oriented decentralized data ownership and architecture**: in the data mesh, data ownership shifts from the centralized ETL pipelines to decentralized units closer to the data following the principles of Domain-Driven Design.
- **Data as a product**: data teams must apply product thinking to the datasets provided, ensuring discoverability and quality-control. Data products must adhere to the DATSIS principles: a data product must be Discoverable, Addressable, Trustworthy, Self-describing, Interoperable and Secure.
- **Self-serve data platform**: data product creation and maintenance should be supported by a self-serve data platform, providing the needed tools and infrastructures and accessible without specialized knowledge.
- **Federated computational governance**: data mesh implementations require a governance model ensuring data sovereignty, standardization, interoperability, and automated decision execution.

Based on the four principles presented above, Dehghani models the logical architecture of data mesh as a multi-plane model [15][16], where the *self-serve data platform* provides high-level abstractions supporting the creation and usage of *data products*, and the *federated computational governance* ensures interoperability and standardization to make the ecosystem work.

Given that the concept of data mesh was first introduced in 2019 and its industry-based roots, there is a small body of scientific literature on the topic. Machado et al. [36] present the motivations and the principles of the data mesh paradigm shift. The same authors also provide domain and conceptual models for the data mesh and a possible on-premise implementation [35]. Wider et al. [53] extend the conceptual model presented in [35] by including data products' input and output ports. A data mesh reference architecture and a possible implementation based on AWS components are also presented by Butte and Butte in [10].

Most of the data mesh literature can be found outside academic research. Goedegebuure et al. [23] perform a gray literature review to identify practical implementations of data mesh in the industry. The authors also identify similarities between data mesh and Service-Oriented Architecture (SOA) concepts, suggesting that the latter could be used to address

the ongoing practical challenges. Bode et al. [5], through interviews with industry experts, identify motivational factors, challenges, best practices, and benefits deriving from the implementation of data meshes. As the findings of [23] and [5] suggest, the industry-based nature of Data Mesh results in a larger practical body of work, with many organizations documenting their journey towards the implementation of a data mesh. Two of the most popular examples are the early-stage implementation approaches of Zalando [46] and Netflix [13]. These approaches are also documented in [36]. Joshi et al. [27] document the data mesh implementation in Saxo Bank, with a particular focus on decentralized data governance. Vestues et al. [49] study the transition towards decentralization and data mesh of a Norwegian public sector organization, outlining benefits and challenges. Other examples of organizations transitioning towards data mesh are Adidas [1], BMW [47], Roche [44], and Glovo [42].

As observed by Bode et al. [5] in their survey, practical implementations of data mesh often find challenges in adopting its principles in their entirety. Given the broad nature of data mesh, organizations' journeys often start with partial implementations, mostly revolving around the organization of data repositories in data products. According to the interviewees, large organizations can benefit from this approach by producing quick wins to promote change. For instance, BMW's Cloud Data Hub [47] still consists of a centralized approach although data products are being introduced. Saxo Bank [27] also started by implementing data mesh's principles only partially. Another finding is that oftentimes the terminology used to describe data mesh principles and implementations is heterogeneous both in industry and in academia. This is particularly evident in Loukiala et al. [33] decentralized data platform proposal, which shares most of data mesh principles although using different terminology.

One of the main concerns for industry and academia is how to achieve effective *federated data governance*. Dehghani provides a logical architecture of data mesh [15][16], but does not focus on implementation details. Among the challenges identified by Bode et al. [5], the shift from centralized to federated data governance is the most relevant for industry professionals. The main concerns regard security, regulatory, and privacy-related topics and problems such as data observability and compliance with data protection regulations. Podlesny et al. [41] observe how, without proper data governance, the decentralization of data may result in issues concerning the privacy of data, potentially exposing Personally Identifiable Information (PII). Similar concerns are expressed in [49], arguing that data privacy regulations such as the General Data Protection Regulation (GDPR) in the European Union may problematize the implementation of data mesh. Wider et al. [53] discuss various approaches to drive governance in data mesh, observing the lack of tools

to achieve federated data governance. According to the authors, the key to overcoming the challenge is finding a balance between decentralization and standardization. In this sense, metadata catalogs are crucial for data observability and discoverability in the data mesh. Dolhopolov et al. [17] present different implementation approaches for metadata catalogs in Data Mesh, including a blockchain-based approach that could be beneficial in cross-organization implementations. However, such an approach presents limitations such as querying performance and storage capacity.

2.2.2. Data Products

Although strictly related to the concept of data mesh, scientific literature on *data products* (DPs) dates back to the 1990s. Wang et al. [52] observed that as data moves from component to component, it undergoes a series of transformations to reach its consumers as an information product. As the focus on data grew significantly in the 2000s and 2010s, the term *information product* was gradually replaced by *data product*. However, in scientific literature, the term *data product* often refers to products and services to final consumers obtained through the processing and the combination of data [34].

Dehghani [15][16] defines a DP as the *architectural quantum* of the data mesh. In platform architecture, the architectural quantum is the smallest architectural unit that can be independently deployed. In a DP, data and code co-exist as a single unit, where the code transforms, maintains, and serves the data.

According to Dehghani [14], in a data mesh the DPs should adhere to the DATSIS principles and should be designed with these principles in mind. More specifically, a DP should be:

- **Discoverable**: a DP should be easily found and should provide the information necessary to use it;
- **Addressable**: a DP should offer a unique and permanent address to users to access it;
- **Trustworthy**: a DP should provide quality and service standards guaranteeing its truthfulness;
- **Self-describing**: a DP should provide a description of its structure and updated documentation:
- **Interoperable**: a DP should follow standards and provide multiple interfaces to be linked and composed with other DPs;

- **Secure**: a DP should provide access control policies restricting access according to the various users' privileges.

In [16], Dehghani introduces other baseline attributes for DPs, such as being valuable and natively accessible. Majchrzak et al. [37] propose applying the FAIR principles (Findability, Addressability, Interoperability, and Reusability) [54] to DP design to maximize reusability and data integration.

Hasan and Legner [25] derive five characteristics of DPs emphasizing their purpose and their properties: i) DPs satisfy recurring information needs, ii) DPs have a well-defined consumer base, iii) DPs create measurable value for organizations, iv) DPs are produced through collating different data elements, and v) DPs are delivered in a consumable form.

According to Hasan and Legner [25], DPs can be categorized into Basic DPs, Analytical DPs, and Advanced Analytical DPs. The authors observe that, as organizations transition towards the adoption of a data mesh, they initially deploy basic DPs as they are simpler and with a larger spectrum of use-cases to then move to the deployment of analytical and advanced DPs.

To support the design and analysis of DPs, Hasan and Legner [24] propose the Data Product Canvas (DPC), based on the Business Model Canvas [40]. The DPC outlines the key elements for designing DPs, focusing on three perspectives: i) desirability from the customers, ii) technological feasibility, and iii) economic viability to organize its nine building blocks 2.1.

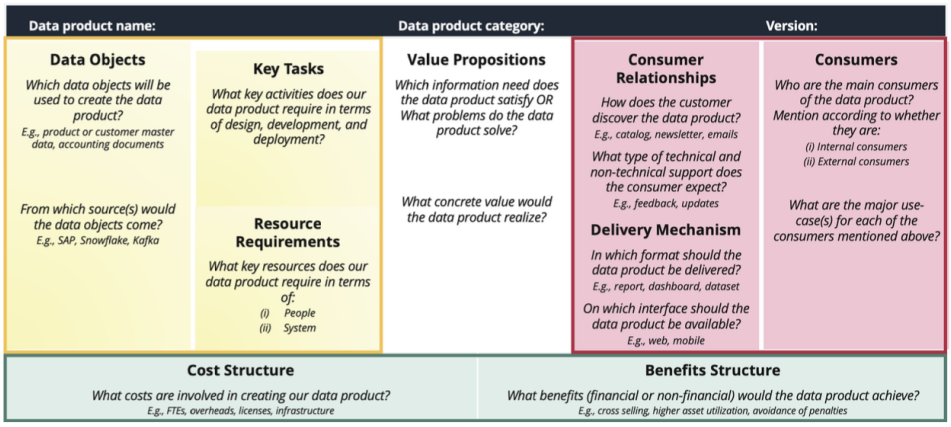

Figure 2.1: Data Product Canvas. From [24].

Majchrzak et al. [37] propose another DPC more focused on the DP's internal structure, describing aspects such as input and output ports, datasets, and internal security policies. While the two canvases offer different perspectives on a DP, they both serve as useful

tools to apply product thinking and designing DPs that satisfy the business needs of the organization.

While DPs are often analyzed with an intra-organizational perspective, adequately designed DPs can also be exchanged and traded inter-organizationally [12].

3 | Data Sharing in the Data Mesh

In this chapter, after providing an overview of the Data Mesh paradigm, its principles, and its model, we study how the current data mesh platform architecture enables and supports data exchanges between Data Products (DPs) and data consumers.

More specifically, the goal is to understand the process of setting up a data exchange between a DP and a data consumer, either another DP or a final user of the mesh, requesting a specific data object. Once the process has been defined, we move to study the frictions occurring at the interface between DP and consumer, with reference to the actual state of the data mesh paradigm.

3.1. Data Friction in Centralized Data Architectures

In centralized data architectures, data are stored in monolithic data platforms which ingest data from multiple sources and, through a series of transformations, make available those data to consumers.

As Dehghani argues [14], this monolithic approach fails to scale as data sources and data consumers increase over time. On one hand, data sources generate more and more data which is ingested and stored in the data platform in heterogeneous, non-standardized formats. On the other hand, data consumers require data in a growing span of use cases involving various transformations on the original data. In this way, the data team running the platform has to deal with bigger and bigger amounts of pressure from both ends of the organization's data pipeline, introducing bottlenecks and ultimately failing to scale along with data sources and data consumers. Furthermore, as the only duty of data sources is to generate data to be ingested into the data platform, data often comes with low overall data quality such as the absence of metadata, causing the data team to deal with high data friction when working with it.

One possible solution is to introduce more specialization in data teams, subdividing them according to the stages of ETL pipelines. However, this does not solve the issue, as the hyper-specialization of data teams introduces friction along the pipeline at the interfacing

of the different teams. In this way, the data processing team will have to align with the data ingestion team, and in the same way, the data serving team will have to align with the data processing team. Not only this solution does not reduce data friction, but, ultimately, hyper-specialized data teams do not align with business domains and needs, failing to deliver valuable data to data consumers.

3.2. Data Mesh Platform Architecture

In centralized data architectures, such as data lakes, oftentimes absence of data and metadata standards as well as poor data curation lead to high friction when sharing and using data [15]. To address these challenges, Dehghani proposes the data mesh paradigm [14–16] as a new approach for organizations to build their data architectures. The data mesh paradigm argues that re-organizing data according to business domains and decentralizing data ownership moving data closer to sources and consumers can reduce the frictions that hamper value-generation in centralized data architectures. Of course, this is only possible if adequate data governance policies are enforced to ensure high data quality and standardization across the business domains of the organization.

Dehghani [15, 16] models the logical architecture of the data mesh as a multi-plane architecture (Figure 3.1), consisting of three planes: i) Data Infrastructure Plane, ii) Data Product Experience Plane, and iii) Mesh Experience Plane. Each plane represents a set of logical capabilities and offers a set of interfaces to access them.

- The **Data Infrastructure Plane** is responsible for managing the low-level infrastructural components such as storage, CI/CD engines, etc. to build and run the Data Mesh, providing access to them with high-abstraction;
- The **Data Product Experience Plane** provides the necessary abstractions to interact with the data products on the mesh. For instance, through this plane, it is possible to create, deploy, and manage Data Products as well as access and combine them;
- The **Mesh Experience Plane** supports the mesh-level capabilities necessary to coordinate the Data Mesh and guarantee standardization and interoperability. Capabilities such as data governance and Data Product discoverability and observability are abstracted in this plane.

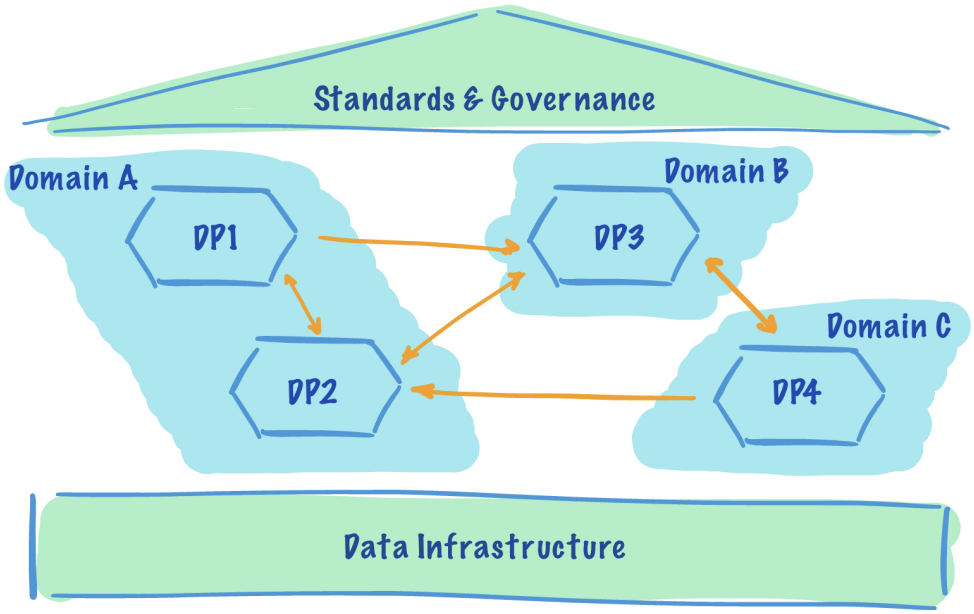

Figure 3.1: Logical architecture of a data mesh. Adapted from [41].

Note that, while in layered architectures often layers can be accessed only in hierarchical order, multi-plane architectures provide access to every plane depending on user needs. For instance, in layered architectures, database access is often possible only through object mappings in the layer above. On the other hand, in the Data Mesh multi-plane architecture the Infrastructure Plane can be accessed from the Mesh Experience Plane.

3.3. Data Product Model

To reduce data friction in data mesh, properly modeled Data Products (DP) are key to ensuring overall high data quality and standardization across the Data Product Experience Plane and DP interoperability.

The DP is the architectural quantum of the data mesh, hence the smallest architectural unit independently deployable. A DP is composed of three structural elements [15]:

- **Code**: including code to consume, transform, and serve data, code to provide access, discoverability, and observability and code to enforce global policies;
- **Data and metadata**: analytical data in heterogeneous formats, along with the metadata to document it and to allow federated governance;
- **Platform dependencies**: infrastructural components allowing creation, deploy-

ment, and management of the DP.

3.3.1. Data Product Ports

DPs interface with other DPs and with the other planes of the logical architecture through ports. According to Dehghani [16], a DP has four types of ports: *input* ports, *output* ports, *control* ports, and *discoverability and observability* ports (Figure 3.2). The first two types enable the interfacing between DPs, allowing them to consume data from operational sources and other DPs and to serve data to other DPs. Control ports enable the performing of governance operations on DPs, such as policy enforcement. Finally, discoverability and observability ports allow the DPs to be available on the data mesh, as well as provide information about them. These ports, in line with the DATSIS principles, are provided with unique addresses (URIs) to be accessed through APIs such as REST or GraphQL.

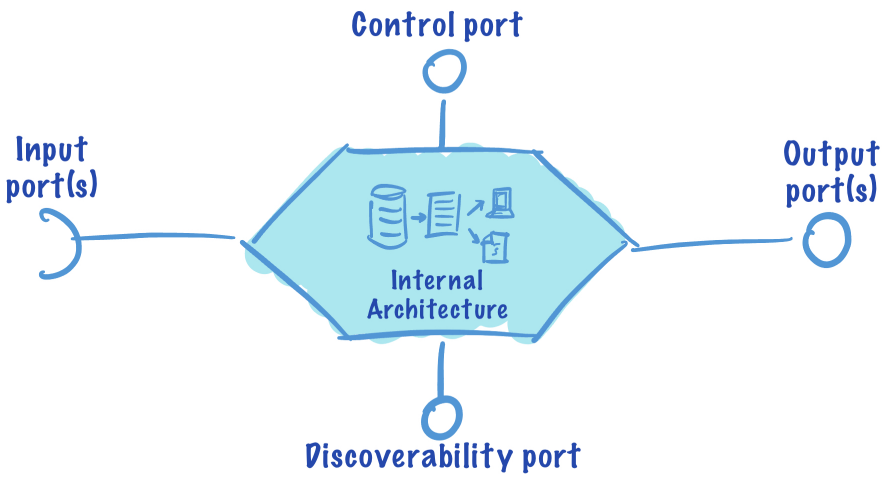

Figure 3.2: Graphical representation of a DP and its ports.

3.3.2. Data Product Types

In a data mesh, we can identify and classify DPs based on their characteristics.

According to Dehghani [14, 16], in a data mesh, we can distinguish between *source-oriented* DPs, closely aligned to the operational sources where the data originates, and *consumer-oriented* DPs, closely aligned with access models for final users. As an organization switches to a decentralized approach in data management, the data from operational sources will be ingested, processed, and served by source-oriented DPs. The output of these DPs will be then ingested by more consumer-oriented DPs which will serve the final users the information they need to make data-driven decisions.

We can further distinguish DPs of a data mesh according to the classification provided in [25], categorizing DPs based on the type of data they ingest and serve to the consumers:

- **Basic DPs**: DPs that provide basic knowledge of their domain. They ingest data from operational sources and lay the foundation for further data analytics;
- **Analytical DPs**: DPs obtained as the result of the application of data analytics techniques to basic DPs. They provide more dense insights than basic DPs;
- **Advanced analytical DPs**: DPs obtained as the result of advanced data analytics transformations on basic and analytical DPs. They provide the highest amount of value and can be used for specific and critical use cases.

The two classifications help us understand how, in a data mesh, different DPs interact with operational data sources, final users, and other DPs. Generally speaking, close to operational sources we will find source-oriented, basic DPs while, on the other end, data consumers are served by consumer-aligned, analytical and advanced analytical DPs.

Figure 3.3 provides a representation of how the three types of DPs inter-relate to one another in a data mesh, and how data flows across them in the value generation process.

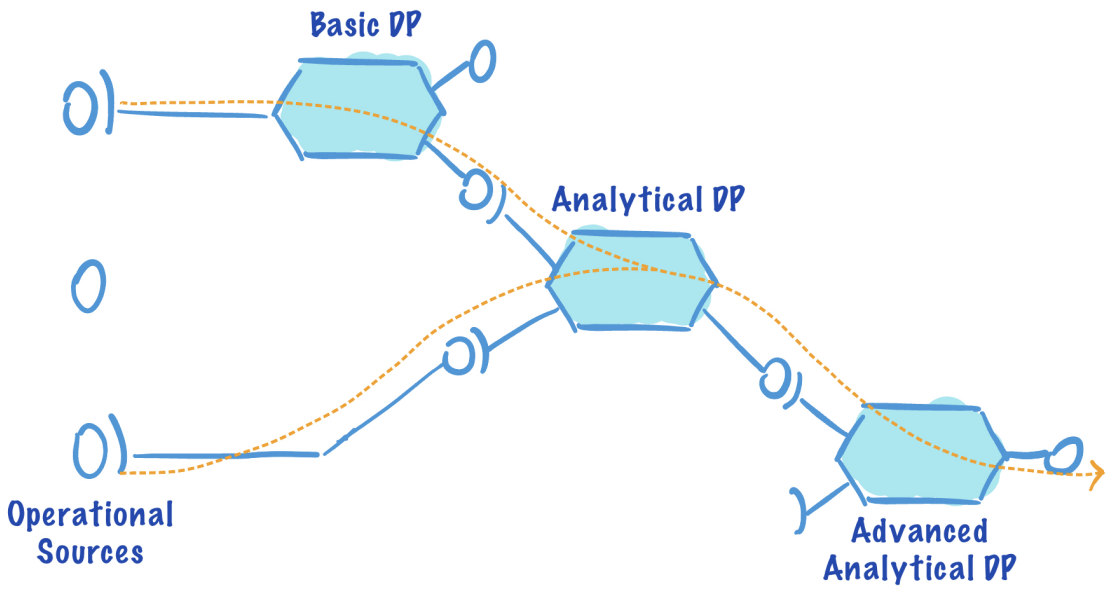

Figure 3.3: Graphical representation basic, analytical, and advanced analytical DPs and their relations.

As information flows through the DPs in a data mesh, the data becomes more processed

and refined. While Basic DPs serve lowly-refined data that cover a wide array of use-cases, Advanced Analytical DPs usually address narrow and specific use-cases. On the other hand, Advanced Analytical DPs provide data with a high value for BI units and data-driven decisions.

Usually, as an organization transitions towards the adoption of a data mesh, initially it will start by designing and deploying basic DPs to then use their output to deploy analytical and advanced analytical DPs at a later stage.

3.3.3. Data Product Manifest

The data mesh must provide capabilities supporting the discoverability and observability of DPs. Given a DP, a data mesh user should be able to find it on the mesh and understand its structure, hence its inputs, outputs, access methods, restrictions, internal policies, and governance standards. As stated previously, DPs have ports to enable their discoverability at the mesh governance plane. These ports have a URI address and can be accessed through standardized APIs. Through the discoverability ports, it is possible to access the *Data Product Manifest* (DPM): a document describing the DP, its configuration, and its state. The DPM should describe the DP, its output ports and their respective SLOs, its input ports and data sources, and its local policies [16]. By reading a DP's manifest, a potential user should be able to understand from where and how the DP gets the data and how it offers it to the consumers, as well as how the data is managed internally (i.e. privacy, retention, locality, etc.).

We extend and refine the definition of DPM by Dehghani [16] by providing further details on how the manifest should be structured and published.

First, the manifest should describe the DP's general information such as name, owner, and version. Moreover, it should provide the URIs of the discoverability and control ports:

```
data-product-name: PatientsDP
data-product-owner: john.doe@tech.abcmedical.com
data-product-description: |-
  This Data Product offers information about the patients
  registered in the ABC Medical databases.
version: 1.0.0
discovery-port-uri: https://mesh.abcmedical.com/patientsdp/discovery
control-port-uri: https://mesh.abcmedical.com/patientsdp/control
```

The second part of the manifest should describe each input port of the DP, stating the

data source (i.e. the output port from which the data is consumed) and the data format:

```
inputs:
  - uri: db:pg://abcmedical.com:5433/patients
    content-type: application/xml
  - uri: db:pg://abcmedical.com:5433/visits
    content-type: application/json
```

By stating its input ports, a DP allows potential users to obtain information about the source of the served data and its lineage.

Then, the manifest should describe, for each output port of the DP, how the data is offered to the consumers. The output port description should be standardized and allow automated parsing in order to verify whether or not the output port satisfies a consumer's requirements. We propose an OpenAPI-like format to describe the data objects served by output ports. However, given the heterogeneous nature of data objects in a data mesh, such a format may fail to capture the entire spectrum of a DP's possible outputs.

Assumption 3.1. *Every output port's object schema can be described with the OpenAPI specification. That is, every data object has a schema and its attributes have OpenAPI-compatible data types.*

By restricting the outputs of a DP with Assumption 3.1, a DP's output ports can be described as follows:

```
output-ports:
  - uri: https://mesh.abcmedical.com/patientsdp/api/v1/patients
    description: |-
      Returns a list of patients registered in the ABC
      Medical databases, ordered by date of insertion.
    response:
      content:
        application/json:
          schema:
            type: array
            items:
              $ref: '#/components/schemas/Patient'
      encryption-standard: none
      anonymized: false
```

Note that each output port has its own URI. Moreover, for each output port's response type, other information such as the encryption standard can be described in addition to the data object's schema. These parameters can vary depending on the organization's needs and internal policies.

As in OpenAPI specifications, DPMs can describe also internal schemas used in responses. For instance, the above output port returns an array of objects described in the "components" section of the DPM. In this case, the DPM's component section will be structured as follows, describing how a "Patient" object is represented:

```yaml
components:
  schemas:
    Patient:
      type: object
      properties:
        id:
          type: integer
        name:
          type: string
        surname:
          type: string
        dateofbirth:
          type: string
          format: date
        insertiondate:
          type: string
          format: date
```

Finally, the DPM should describe how the data is managed internally (i.e. privacy, retention, locality, etc.) and eventual access restrictions for its outputs. For instance, a specific output may be available only to certain domains or to users with a specific set of privileges. The definition of this section heavily depends on the organization's policies and internal regulations, and, for the sake of simplicity, its implementation and representation are outside the scope of this chapter.

3.4. Data Exchanges in Data Mesh

The goal of the data mesh is to enable data sharing and integration to ultimately get the maximum value from the organization's data architecture. For this reason, it is key that the data exchanges between DPs and consumers (either other DPs or final users) are as frictionless as possible.

With reference to the DP model presented in Section 3.3.1, each DP provides a set of output ports that can be accessed through globally standardized APIs. Note that some output ports may implement access control rules restricting access to the DP's output. For the sake of simplicity, in this study, we are not including access control rules in our data exchange overview. On the other side, the consumer has an input port to receive the data sent by the DP. Just like the DP's output ports have specific characteristics and formats, as discussed in Section 3.3.3, the consumer's input will have to satisfy a set of requirements in order to be successfully ingested.

To enable the data sharing between provider and consumer, the output port of the provider DP and the input port of the consumer must be interoperable. This means that the format and the other characteristics of the provider's output must match the requirements of the consumer's expected input. If the consumer's ingestion pipeline is designed for the CSV format but the provider's output is a JSON file, most likely the data exchange will end up with an error. A crucial aspect when modeling the two ports to ensure interoperability is deciding which port should be modeled after the other. In other words, if the consumer must adapt to the provider or vice versa.

Keeping in mind that the ultimate goal of data mesh is to deliver value through data, the more refined, processed, and aggregated data is, the more valuable it becomes. With this in mind, and in line with the classification of [25], we argue that the highest amount of value is delivered by advanced analytical DPs at the end of the data flows inside the data mesh. For this reason, it should be the provider's duty to serve data consumers with adequate outputs satisfying their needs. This is in line with the notion of Consumer-Driven Contracts [45]. The only exception to this observation can be made for basic DPs that ingest data from operational sources. Since operational sources are likely to be legacy components in an organization's data architecture, it can be difficult to re-model their output, hence requiring the basic DPs to adapt their input ports to them. This exception is outside the scope of this study.

According to the domain model proposed by Wider et al. [53], DPs' output ports should state their Service Level Objectives (SLO), while data consumers' input ports should state

their assumptions and their requirements. According to the same authors, this can be achieved through the use of executable contract tests. In this way, the consumer can state its requirements and the provider can model its output accordingly.

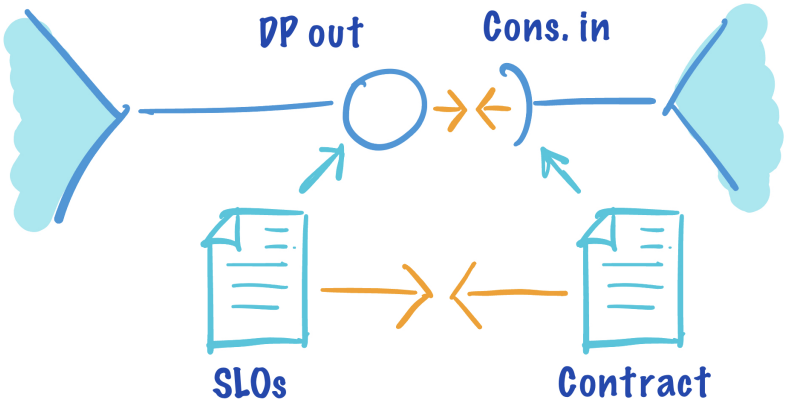

Figure 3.4: Interfacing between a DP and a data consumer.

If the provider DP output port and the consumer input port can interface, then the data exchange can take place.

This contract-based approach is particularly useful to manage the evolution of a DP in a consumer-oriented way. As a DP's lifecycle progresses and its internal structure evolves, its outputs must always satisfy the contract conditions of the served consumers. In this way, there is no risk that a new version of a DP's output port is released changing its characteristics in such a way that its consumers cannot ingest the data object anymore. On the other end, as a consumer's needs evolve, new versions of the contracts with its providers will be released. In this way, as the data mesh evolves and grows in size, its data flows keep working and do not interrupt the value generation processes.

However, as new DPs are deployed and new use cases arise, the number of interfaces between providers and consumers significantly increases over time. The risk is that, in large data mesh implementations, the number of interfaces and their inherent consumer contracts slow down the deployment and evolution of DPs.

3.5. Chapter Conclusions

The data mesh paradigm aims to overcome the frictions occurring in centralized data architectures such as data warehouses and data lakes proposing a decentralized approach in managing and owning data. To do so, data are aligned with business domains and

organized in DPs closer to data sources and data consumers.

To reduce the friction in data flows across the organization, it is crucial to design proper DPs and adopt standardized data management practices. In this sense, we propose a representation of DPs aligned with the model introduced by Dehghani [16] and enriched with a few additional components. More specifically, we provide a standardized representation of a DP's structure and output ports with the introduction of the Data Product Manifest (Section 3.3.3).

However, as an organization shifts towards decentralized data ownership and data management and deploys various DPs, without a proper data sharing framework it becomes increasingly difficult to manage the data exchanges in the mesh.

The data mesh at its current state does not provide a structured method to approach data exchanges. Given a data consumer requesting a specific data object to a provider DP, the data mesh should support the creation of a new interface to enable the data exchange between the provider and the consumer. Moreover, given a consumer's request and the current state of the provider, it should be possible to define and quantify the effort needed to enable the interfacing between the two actors. In this way, data sharing in the data mesh can be approached in a structured way with the ultimate goal of managing the evolution of DPs in a friction-aware manner.

The first step towards this achievement is the definition of a data exchange model and the inherent definition of data friction. In the next chapter, we will define such a model to then apply it to data exchanges in the data mesh.

4 | Modeling Data Friction in Data Exchanges

To address the challenges of setting up data exchanges in data meshes, and reduce the data friction in such a scenario, we need first a model that allows us to structure a data exchange and, within such a model, provide a definition of data friction.

The goal of this chapter is to define an abstract model for designing data exchanges between a data provider and a data consumer. Such a model should be able to effectively represent, with a high enough level of abstraction, the process of setting up a data exchange between provider and consumer. It should cover a wide enough array of cases to be applied in real-world scenarios, including data exchanges in data meshes.

We will then derive from the model our definitions of effort and data friction, and we will introduce a mathematical method to calculate them in a data exchange.

4.1. List of Symbols

In this chapter, we will introduce some symbols as a shorthand to refer to various components of our model.

Table 4.1 serves as a reference to the reader who, along with the reading, may need to look back to the meaning of a certain notation.

Symbol	Description
d	Data object
$C = c_i$	Universe of data pipeline capabilities
c_t	Transmission capability
$C_P \subseteq C$	Capabilities already implemented by the data provider
$\overleftarrow{C} \subseteq C$	Capabilities that must be deployed by the provider
$\overrightarrow{C} \subseteq C$	Capabilities that must be deployed by the consumer
p_α	Data pipeline between provider and consumer α
$\overleftarrow{p}_{\alpha,x}$	Portion of the pipeline p_α deployed by the provider
$\overrightarrow{p}_{\alpha,x}$	Portion of the pipeline p_α deployed by the consumer
$\widehat{p}_{\alpha,x}$	Deployment configuration x of the pipeline p_α
$\widehat{p}'_{\alpha,x}$	Subset $[1, m], m \leq n$ of the deployed pipeline $\widehat{p}_{\alpha,x}$ already executed
$E_I(c)$	Implementation effort of capability c
$E_{I,P}$	Total implementation effort of the provider
$E_E(c)$	Execution effort of capability c
$E_{E,P}$	Total execution effort of the provider
$E_{E,\alpha}$	Total execution effort of the consumer α
$e(c)$	Unitary execution effort of capability c
N_d	Size of the data object d
μ_I	Implementation friction coefficient
μ_E	Execution friction coefficient

Table 4.1: List of symbols introduced in this chapter.

4.2. Model Definition

In this section we present our data exchange model, defining its main components and the underlying assumptions. The presented model reduces a data exchange to its basic components, with a high enough level of abstraction to be adapted to the various real-world scenarios.

4.2.1. Data Exchange Components

We can identify three main components in the basic model that we are going to analyze:

- **Data Provider**: the entity that serves the data object (i.e. the owner of the data object);
- **Data Consumer(s)**: the entity/ies that receive the data object served by the provider;
- **Data Object**: the payload that is transmitted from the provider to the consumer in the data exchange.

Example. Let's consider as a data provider a hospital server containing biographical information about the hospital's patients, stored in some format and served through REST APIs. Data consumers can connect to the server via the HTTP protocol and start exchanging data objects. □

The data exchange between the provider and consumer(s) happens over a pre-established communication channel between the two actors. In the example above, the communication channel is represented by an internet connection between the server and consumers and the HTTP protocol. We assume that the communication channel is already present and functioning: the establishment of the communication channel between the two parts is outside the scope of this study.

A representation of our basic model is depicted in Figure 4.1. Note that, in this basic model there is a single data provider and one or more data consumers, each setting up a data exchange with the data provider. It is also possible that a data exchange involves multiple data providers. However, such an $M : N$ model is outside the scope of this study and a data exchange with M data providers can be simplified by decomposing it into M data exchanges, each with a single provider.

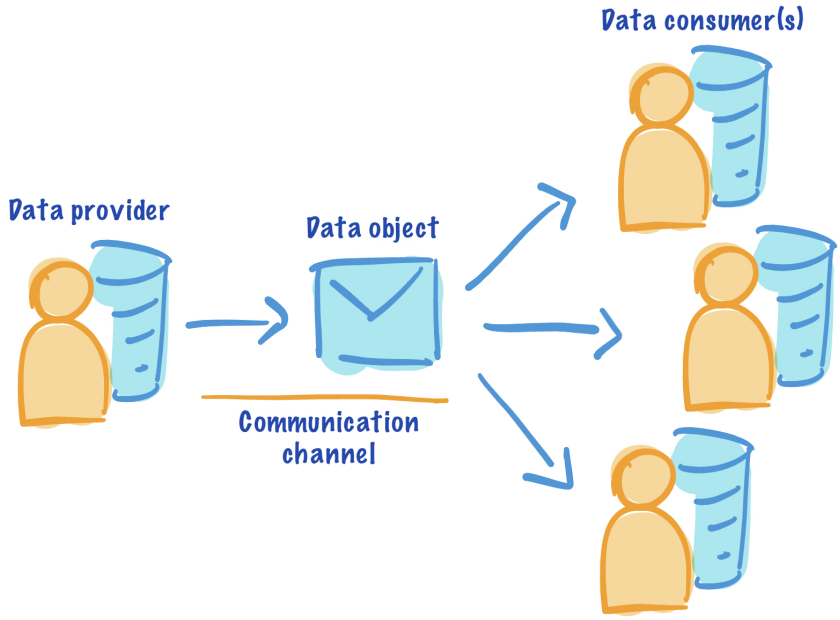

Figure 4.1: Basic model for data exchange between two parts.

4.2.2. Data Objects and Capabilities

The goal of the data exchange is to successfully transmit the data object from the data provider to the data consumer. A data object d is an item or a group of items containing relevant information related to a business object. For instance, considering the example above a possible data object d served by the provider could be a list containing patients registered in the database in CSV format.

A data object, to be served, can undergo one or more *transformations*. A transformation is performed by a *capability*. A capability can be represented as a function c that receives a data object d_{in} as input and applies to it a certain transformation returning a data object, thus:

$$d_{out} = c(d_{in}) \tag{4.1}$$

Example. Given the list of hospital patients, a possible transformation would be to aggregate the patients according to their area. The capability implementing the transformation can be, in this case, a simple Python script that, given a list of patients with their biographic information, returns the number of patients per ZIP code. □

For a capability to transform a data object, such capability must be first *implemented*

and then *executed*. The implementation of a capability consists of the realization of the piece of software that is able to perform the transformation. After a capability has been implemented, it can be then executed on the data object. Executing the capability c on a data object d means applying the transformation $c(d)$ to such data object.

The set C contains all the possible capabilities that can be deployed in the considered system. The boundaries of C may not be known a priori and grow as new capabilities are deployed.

A capability is implemented and executed on one or more resources provided by the organization or whoever is in charge of performing the transformation. The resource definition is outside the scope of this study, and we may assume that the resources needed to implement and execute such capability are already defined and always available.

Note that, while a capability has to be implemented only once, it can be then executed multiple times on various data objects. Obviously, the data object used as input of the capability must be compatible with the transformation implemented by the capability itself. The compatibility between a data object and a capability is outside the scope of this study and we may assume for now that, if a data object is fed as input to a capability, then data object and capability are compatible.

Given two capabilities $c_1, c_2 \in C$, if their output for every input d is the same, then the two capabilities are *equivalent*. In symbols:

$$c_1 \in C, c_2 \in C, d : \forall d, c_1(d) = c_2(d) \Rightarrow c_1 = c_2 \tag{4.2}$$

4.2.3. Data Pipeline Model

It is unlikely that, in a data exchange, a data object undergoes only one transformation: a *Data Exchange Pipeline*, or *Data Pipeline*, can be modeled as a sequence of capabilities, each implementing a transformation to be applied to the input data object. In symbols, a pipeline is defined as

$$p = \{c_1, ..., c_n\} \tag{4.3}$$

thus, p is an ordered set of capabilities $\{c_1, ..., c_n\}$, $c_i \in C \ \forall i \in [1, n]$, where, given two capabilities c_i and c_j and $i, j \in [1, n] : i < j$, then the capability c_i has to be executed prior to the capability c_j. A data pipeline $p = \{c_1, ..., c_n\}$ can be represented as a function obtained through the composition function (represented with the operator $\circ$,

e.g. $g(f(x)) = g \circ f$) of the functions of its capabilities $c_n \circ ... \circ c_1$. Given a data pipeline p and a data object d, the result of the execution of the pipeline on the data object can be expressed as

$$p(d) = c_n \circ ... \circ c_1(d). \tag{4.4}$$

Example. To obtain the aggregated view of patients per ZIP area from the hospital server, it may be necessary a pipeline $p = \{c_1, c_2, c_3\}$ consisting of three capabilities: c_1 ingesting the list of patients, c_2 aggregating them according to their ZIP code, and c_3 formatting the output in the desired format. □

Note that, in our definition, the capabilities of a data pipeline p can be executed only in a linear way, one after another according to the order relationship of p. This pipeline model does not cover those cases where two capabilities can be executed simultaneously: in such cases, we may assume that the capabilities executed simultaneously can also be executed in series, in an arbitrary order.

Given two data pipelines p_1 and p_2, $p_1 \cap p_2$ corresponds to the capabilities that are shared by both pipelines. This means that, in a system, the capabilities used to implement and execute a data pipeline can be shared with other pipelines.

Example. Considering the data object d containing the biographic information of the hospital patients, there may be various pipelines deriving a variety of data objects from d. Among these pipelines, let's consider the two pipelines p_1 and p_2 such that $d_1 = p_1(d)$ contains the number of patients per ZIP area in a JSON array and $d_2 = p_2(d)$ contains the same information but as a CSV file. p_1 and p_2 will likely share part of their capabilities up to a certain point to obtain their respective outputs d_1 and d_2. □

4.2.4. Data Exchange Model

With reference to the basic model presented in Section 4.2.1, the data exchange between provider and consumer(s) is enabled by a data pipeline (Figure 4.2). Such a pipeline transforms the data object served by the provider into the one requested by the consumer.

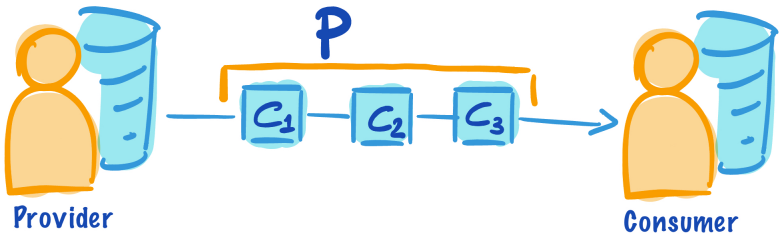

Figure 4.2: Data pipeline between a data provider and a data consumer. The capabilities of the pipeline transform the data object served by the provider into the object requested by the consumer.

Since in our basic model we consider a single provider and one or more data consumers, there will be one or more pipelines enabling the data exchanges between the provider and the various consumers. Generally speaking, considering a data consumer α, the notation p_α refers to the pipeline enabling the data exchange between the data provider and the consumer α.

Given a data exchange with a consumer α and a capability $c \in p_\alpha$, it can be deployed either on the data provider's side or on the data consumer's side. Generally speaking, there may be some capabilities that can be deployed only on one side of the data exchange: given C, the sets $\overleftarrow{C} \subseteq C$ and $\overrightarrow{C} \subseteq C$ contain, respectively, the capabilities in C that can be deployed only on the provider and on the consumer side.

For instance, if the biographic information of a patient needs to be anonymized before sharing, it is safe to assume that the capability implementing the anonymization of Personally Identifiable Information (PII) will have to be deployed on the provider side. It would make no sense to anonymize the data after having shared it with the consumer.

The other capabilities of the pipeline can either be deployed on the provider side or the consumer side, indifferently. We may assume that the capabilities that can be implemented only on one side can be, respectively, only at the beginning and at the end of the pipeline. That is, given a pipeline $p_\alpha = \{c_1, ..., c_x, ..., c_n\}$, if $c_x \in \overleftarrow{C}$, then $\forall c_i \in p_\alpha, 1 \leq i < x : c_i \in \overleftarrow{C}$. The same goes for the capabilities in $\overrightarrow{C}$: if $c_x \in \overrightarrow{C}$, then $\forall c_j \in p_\alpha, x < j \leq n : c_j \in \overrightarrow{C}$.

For the sake of simplicity, we may assume that, given a pipeline $p_\alpha = \{c_1, ..., c_n\}$ deployed partially on the provider side and partially on the consumer α side, all the capabilities up to a certain capability $c_x \in p_\alpha$ are deployed on the provider side and all the subsequent capabilities are deployed on the consumer side. In other words, if a capability c_y is

deployed consumer-side, all the capabilities $c_j \in p_\alpha, j \in (y, n]$ have to be deployed on the consumer side as well.

In symbols, given a data pipeline $p_\alpha = \{c_1, ..., c_x, ..., c_n\}$, the set $\overleftarrow{p}_{\alpha,x} = \{c_1, ..., c_x\}$ represents the portion of the pipeline that is deployed on the data provider side. Conversely, the set $\overrightarrow{p}_{\alpha,x+1} = \{c_{x+1}, ..., c_n\}$ represents all the capabilities deployed on the consumer side.

Example. Let's consider a pipeline $p_\alpha = \{c_1, c_2, c_3\}$ between provider and consumer α that, given, the list containing the hospital patients served by the provider, filters the patients based on their location (c_1), anonymizes their PII (c_2), and converts the list into a JSON array (c_3). A possible way to deploy such a pipeline would be to deploy on the provider side the capabilities up to the anonymization and to deploy the format conversion capability on the consumer side. In this case, $\overleftarrow{p}_{\alpha,2} = \{c_1, c_2\}$ and $\overrightarrow{p}_{\alpha,3} = \{c_3\}$. □

Given a pipeline p_α, since every capability $c \in p_\alpha$ must be deployed either provider-side or consumer-side, and given two capabilities $c_i, c_j \in p, i < j$, if c_j is deployed on the provider side then also c_i must be deployed on the same side, then:

$$p = \overleftarrow{p}_x + \overrightarrow{p}_{x+1} \tag{4.5}$$

Since the capabilities in $\overleftarrow{C}$ must be deployed by the provider, necessarily $\overleftarrow{C} \subseteq \overleftarrow{p}_{\alpha,x}$. In the same way, $\overrightarrow{C} \subseteq \overrightarrow{p}_{\alpha,x+1}$.

4.2.5. Data Transmission and Transmission Capability

In a data pipeline p deployed partially provider-side and consumer-side, the data object will be transmitted from provider to consumer after the last capability in $\overleftarrow{p}_{\alpha,x}$ and before the first capability in $\overrightarrow{p}_{\alpha,x+1}$. That is, given $\overleftarrow{p}_{\alpha,x} = \{c_1, ..., c_x\}$ and $\overrightarrow{p}_{\alpha,x+1} = \{c_{x+1}, ..., c_n\}$, after the execution of c_x the partially transformed data object will be sent to the data consumer.

The transmission of the data object can be represented as an additional capability in the pipeline. This additional capability is called *transmission capability* and is represented with c_t. Given data pipeline p and the sets $\overleftarrow{p}_{\alpha,x}$ and $\overrightarrow{p}_{\alpha,x+1}$, the capabilities in p will be executed by the provider up to c_x, then c_t will be executed, and then the consumer will execute the remaining capabilities.

Depending on the step of the pipeline in which the transmission occurs, the transmit-

ted data object will have certain characteristics. The characteristics of the data object transmitted over the communication channel will affect its transmission, making it easier or harder. For instance, a large, not aggregated data object will be heavier to transmit than the same data object after its aggregation. Another characteristic that can affect the transmission of a data object is the presence (or not) of cryptographic techniques to protect the communication. For the sake of simplicity, among the characteristics affecting the transmission we will only consider the size of the data object. A larger data object will be more difficult to transmit, and vice versa.

4.2.6. Deployment Configurations

The same data pipeline can be configured in different ways depending on which capabilities are deployed on the data provider side or the data consumer side. Depending on how the pipeline is configured, the transmission capability c_t will be executed at a certain point. A *deployment configuration* $\widehat{p}_x$ of a data pipeline p identifies the capabilities deployed provider-side and consumer-side in that specific configuration. In other words, a deployment configuration $\widehat{p}_x$ identifies at which point of the pipeline p the transmission c_t occurs:

$$\widehat{p}_{\alpha,x} = \overleftarrow{p}_{\alpha,x}, c_t, \overrightarrow{p}_{\alpha,x+1} \tag{4.6}$$

In the example of Section 4.2.4, the transmission of the data object will take place after the anonymization and before the format conversion. In this case, the deployment configuration will be $\widehat{p}_{\alpha,3} = \{c_1, c_2, c_t, c_3\}$.

Depending on the length of the pipeline and the presence of capabilities that have to be deployed on a specific side of the data exchange, there will be less or more possible deployment configurations. Given a pipeline p_α and the sets $\overleftarrow{C}$ and $\overrightarrow{C}$, there will be

$$1 + |p_\alpha| - |p_\alpha \cap \overleftarrow{C}| - |p_\alpha \cap \overrightarrow{C}| \tag{4.7}$$

possible deployment configurations.

According to Equation 4.7, a pipeline $p_\alpha = \{c_1, ..., c_n\}$ with $\overleftarrow{C} = \emptyset$ and $\overrightarrow{C} = \emptyset$ will have $n + 1$ possible deployment configurations:

- In the deployment configuration $\widehat{p}_{\alpha,0}$ all the capabilities will be deployed on the consumer side.

- in the deployment configuration $\widehat{p}_{\alpha,n}$ all the capabilities will be deployed on the provider side.
- In the deployment configuration $\widehat{p}_{\alpha,x}, x \in [1, n)$ the capabilities up to c_x will be deployed by the provider and the remaining ones by the consumer.

If there are capabilities that must be deployed on a specific side, then the possible deployment configurations diminish. For instance, if $\overleftarrow{C} = \{c_0\}$, then the deployment configuration $\widehat{p}_{\alpha,0}$ is not possible because by definition in $\widehat{p}_{\alpha,0}$ all the capabilities are deployed on the consumer side.

4.3. Setting up the data exchange

Once defined the main components, we move our analysis to understand how to set up a new data exchange between a data provider and a data consumer.

4.3.1. Data Provider Model

In its most basic form, the data provider serves a data object d, without any transformation, to potential data consumers.

As new data consumers set up new data exchanges with the provider, the latter will implement capabilities in the various data exchange pipelines. Note that the implementation of such capabilities is always performed by the provider, while their execution can either be on the provider's resources or the consumer's resources. The set C_P contains all the capabilities implemented by the provider in the various data exchanges with its data consumers.

At t_0, the data provider serves only d in its basic form and has not set up any data exchange, hence $C_{Provider} = \emptyset$. As a new data consumer α sets up a data exchange with the provider through the pipeline p_α, the capabilities in p_α will be then included in C_P for the following data exchanges. That is, if a new data exchange is set up and the corresponding data pipeline has capabilities already in C_P, the provider will not have to implement them.

4.3.2. Data Consumer Model

On the other side of the model, a data consumer ingests data from the data provider to use the data object according to its needs. A data consumer, in turn, may be a data provider to other data consumers. For instance, a data consumer may ingest data objects

from various data providers, aggregate and transform them, and serve the result as a new data object.

A data consumer ingests data objects from a number of data providers. For each of these data exchanges, the capabilities of the corresponding data pipeline may be deployed on the resources provided by the data consumer.

4.3.3. Data Exchange Phases

In this scenario, a data consumer α wants to set up a data exchange with the data provider. More specifically, the data consumer α requires a specific data object d_α satisfying a number of requirements. The data consumer must have knowledge, at least, of the data object d offered by the data provider, so that the requirements of d_α have a certain degree of feasibility. In other words, it must be possible to obtain d_α from d through a series of transformations. In symbols, there must be at least one data exchange pipeline p_α such that $d_\alpha = p_\alpha(d)$.

We can identify three phases in the data exchange setup: the *agreement* phase, the *implementation* phase and the *execution* phase (Figure 4.3).

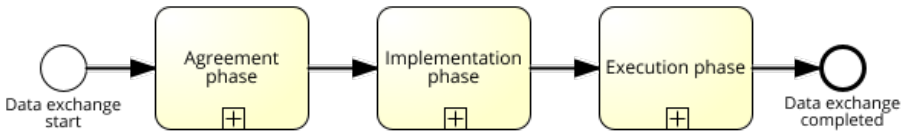

Figure 4.3: BPMN diagram of the three phases of a data exchange.

4.3.4. Agreement Phase

Goal of the *agreement* phase is to define the data pipeline needed to transform the data object d offered by the data provider into the data object d_α requested by the data consumer α and its deployment configuration. The structure of this process can vary depending on the context in which the agreement phase takes place. The BPMN choreography and process in Figure 4.4 and Figure 4.5 describe a possible agreement phase between data provider and data consumer.

At the beginning of the agreement phase, the data consumer α sends a request to the provider listing the requirements that the new data object d_α should satisfy. The data provider can accept the requirements, revise them, or abort the data exchange. If the requirements of the data object d_α are revised, then it is up to the data consumer to

accept the new requirements, revise them, or abort the data exchange. This procedure ends when the two actors agree on a final set of requirements of the requested data object d_α.

Once d_α is defined, data provider and data consumer α define the pipeline p_α such that $d_\alpha = p_\alpha(d)$. It is also possible that the pipeline definition is performed solely by the data provider. The process of defining such pipeline is outside the scope of this study and we may assume that, once d_α has been defined, there exists at least one pipeline p_α such that $d_\alpha = p_\alpha(d)$.

The next step of the agreement phase is to decide, among all the possible deployment configurations, the deployment configuration $\widehat{p}_{\alpha,x}$ to implement in the data exchange. That is, which capabilities in p_α provider and consumer will have to respectively deploy. Also this step is generally performed by the data provider but in some cases the deployment configuration may be defined collaboratively between data provider and data consumer.

How data provider and data consumer reach an agreement on data object, pipeline and deployment configuration varies from case to case. Generally speaking, the agreement phase involves an active message exchange between the two parts to define the overall structure of the data exchange. It is possible that, in some cases, a side of the data exchange is less involved than the other in the decision making process.

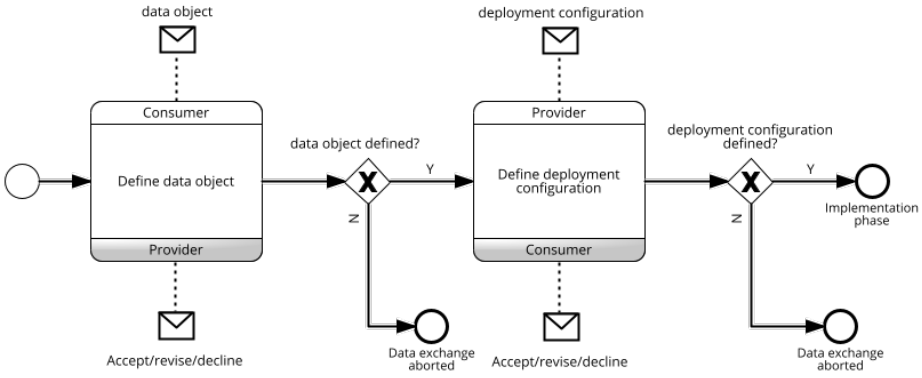

Figure 4.4: BPMN choreography of a possible agreement phase.

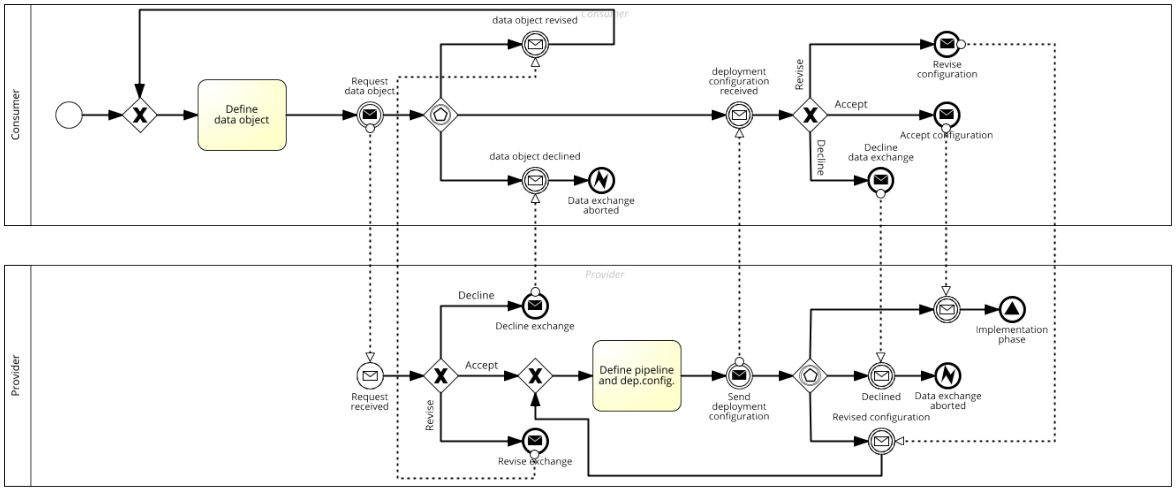

Figure 4.5: BPMN diagram of a possible agreement phase.

Once data provider and data consumer agree on the pipeline p_α and on the deployment configuration $\widehat{p}_{\alpha,x}$, the agreement phase is complete.

4.3.5. Implementation Phase

At the end of the agreement phase, provider and consumer agree on the pipeline p_α and on a deployment configuration $\widehat{p}_{\alpha,x}$.

Goal of the *implementation* phase is to implement those capabilities in p_α that have not been already implemented. The BPMN process of the implementation phase is represented in Figure 4.6.

By definition, p_α is composed by a series of capabilities $\{c_1, ..., c_n\}$. As said before, the implementation of the capabilities of the pipeline is up to the data provider. The data provider will have to implement the capabilities in p_α accordingly to the deployment configuration $\widehat{p}_{\alpha,x}$. That is, the capabilities in $\overleftarrow{p}_{\alpha,x}$ will be implemented on the provider side while the capabilities in $\overrightarrow{p}_{\alpha,x+1}$.

Of the capabilities in p_α, it is possible that some of them are already implemented in C_P. In symbols, given p_α and C_P, the data provider will have to implement only the capabilities in $p_\alpha \backslash C_P$.

It is possible that some capabilities in p_α cannot be implemented. If this is the case, either the two parts re-negotiate the agreement going back to the *agreement* phase or the data exchange cannot take place.

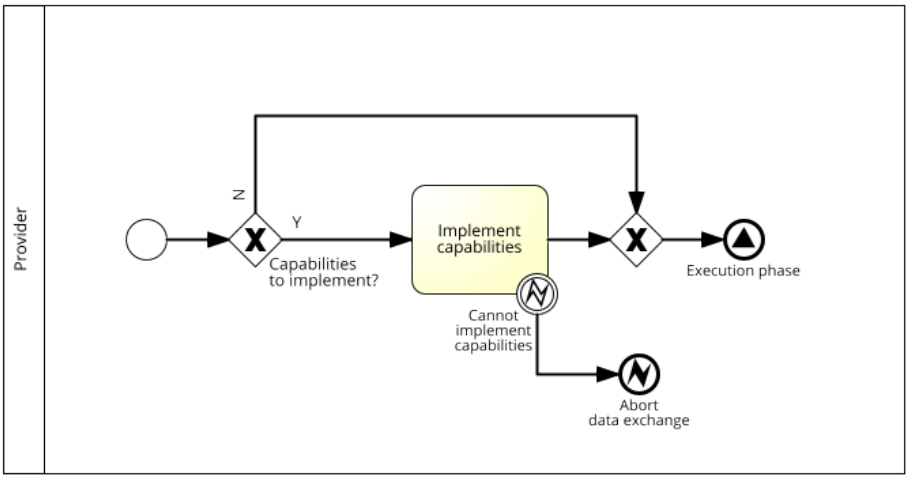

Figure 4.6: BPMN diagram of the implementation phase.

Once all the capabilities in p_α have been implemented, the data object d_α requested by the consumer is ready to be served and the implementation phase is complete and the data exchange moves to the *execution* phase. At the end of the implementation phase $p_\alpha \subseteq C_P$.

4.3.6. Execution phase

After p_α has been implemented, in order to perform the data exchange the capabilities in p_α have to be executed. Goal of the *execution* phase is to execute all the capabilities in p_α in order to effectively transform d in d_α and, depending on the chosen deployment configuration, transmit the data object from provider to consumer. The BPMN diagram of the execution phase is shown in Figure 4.7.

As said in Section 4.2.2, a capability has to be implemented only once but it can be executed multiple times. That is, if the original data object d changes, the various pipelines starting from it have to be executed once again in order to update the data object served to the various data consumers. Every time the data consumer α requests the data object d_α at the end of the pipeline p_α, if d has been updated the pipeline has to be executed again. Otherwise, d_α is already available and the execution phase it is not needed.

Given a data pipeline p_α, it is possible that its capabilities have been executed up to a certain point $c_i \in p_\alpha$ and d has not been updated. If this is the case, then only the capabilities after c_i have to be executed to complete the data exchange. This case may arise when the data pipeline shares part of its capabilities with other pipelines starting

from d, that have been previously executed.

Given a pipeline p_α and a deployment configuration $\widehat{p}_{\alpha,x}$, the set $\widehat{p}'_{\alpha,x}$ contains the capabilities of the pipeline that have already been executed at the beginning of the execution phase. For simplicity, we may assume that the capabilities in $\widehat{p}'_{\alpha,x}$, if any, start from the beginning of the pipeline up to a certain point and are all contiguous. That is, given a pipeline $p_\alpha = \{c_1, ..., c_n\}$ where, at the beginning of the execution phase, the capabilities have been already executed up to $c_x, x \in [1, n]$, then $\widehat{p}'_{\alpha,x}$ contains all and only the capabilities $c_i, i \in [1, x]$. In other words, it is not possible that $c_i \in p_\alpha, c_j \in p_\alpha, c_i \notin \widehat{p}'_{\alpha,x} \wedge c_j \in \widehat{p}'_{\alpha,x}, i < j$.

Please note that in certain environments there may not be the possibility to store the result of the partial execution of a data pipeline. Therefore, independently of the update rate of the starting data object, all the capabilities in p_α will have to be executed at every data exchange.

Also in the execution phase it is possible that, in a pipeline, a capability cannot be executed. This situation may arise in the case of execution errors in capabilities such as the inability to connect to the server. If an error during the execution phase occurs, then the execution phase will have to be performed once again. If some capabilities of the pipeline have been already executed prior to the occurrence of the error, and d has not been updated in the meantime, it will not be necessary to re-execute the capabilities before the error.

In the execution phase the capabilities are executed according to their order in the pipeline. Depending on the deployment configuration, the provider will execute the capabilities in the pipeline up to a certain point to then transmit the (partially) transformed data object to the data consumer, who will then execute the remaining capabilities to obtain the final data object.

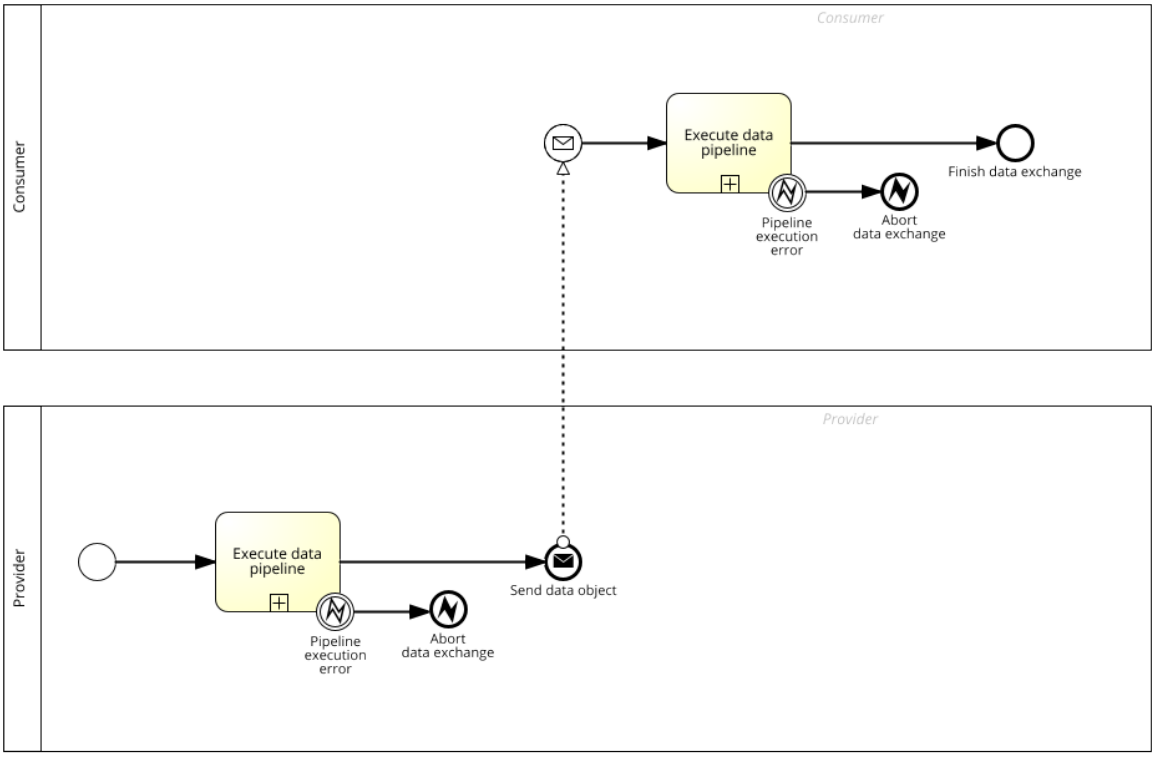

Figure 4.7: BPMN diagram of the execution phase.

Once the execution phase is completed, all the capabilities in p_α have been executed and d_α is ready to be used by the data consumer. At this point, the data exchange is concluded.

4.4. Effort in a Data Exchange

After having identified in the agreement phase the required capabilities to perform the data exchange, goal of the implementation and execution phases is to, respectively, implement and execute those capabilities.

The implementation and execution of such capabilities requires an *effort*. The data exchange between data provider and data consumer is enabled by their respective efforts (Figure 4.8).

Definition 4.1. *We define as* ***effort*** *E the amount of work that an actor has to do in order to perform a capability.*

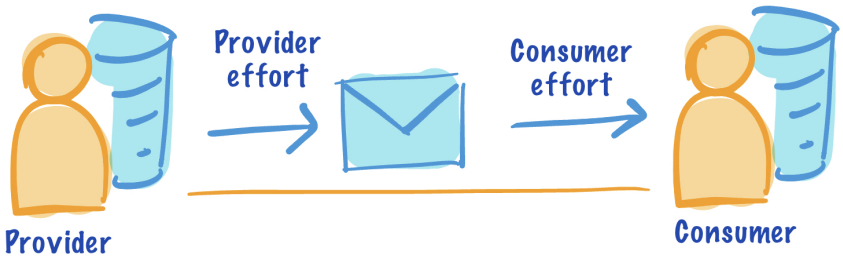

Figure 4.8: Graphical representation of provider and consumer effort.

Note that, in Definition 4.1, the capability is *performed*. Referring to the phases of the data exchange setup identified in Section 4.3, we can further distinguish the effort in *implementation* and *execution* effort.

4.4.1. Implementation Effort

With reference to the exchange phases discussed in the previous section, in the implementation phase the data provider has to make an effort in order to implement the capabilities of the pipeline.

Definition 4.2. *We define as* ***implementation effort*** E_I *the amount of work that an actor has to do in order to implement a capability.*

The implementation effort depends on the complexity of the capability: a more complex capability be more difficult to implement, requiring more effort. In symbols, given the capability c, we express the implementation to implement the capability successfully as $E_I(c)$

An estimate of the implementation effort required by a capability can be obtained by applying software cost estimation models such as SLIM or COCOMO [32].

4.4.2. Execution Effort

After the implementation phase is completed, the implemented capabilities have to be executed. Data provider and consumer have to make an effort in order to execute such capabilities.

Definition 4.3. *We define as* ***execution effort*** E_E *the amount of work that an actor has to do in order to execute a capability.*

The execution effort depends not only on the complexity of the capability but also on the size of the data object d on which the task is executed. The same task will require more effort when executed on a larger dataset.

In order to explicit the data product size in the definition, we define as *unitary effort* $e(c)$ the effort required to execute the capability c on a single data item of the data object. In this way, the total execution effort required by the capability c, $E_E(c)$, equals the unitary effort $e(c)$ needed for one data item times the total amount of data items of the data object N_d:

$$E_E(c) = e(c)N_d \tag{4.8}$$

The unitary effort required to execute a capability $e(c)$ can be estimated experimentally.

Note that the N_d varies along the pipeline: transformations such as filtering, aggregation and merging alter the structure of the data object, returning a data object with a different size and a different amount of data items. Usually, as data gets processed and refined through a pipeline, its size tends to diminish as result of transformations such as those just mentioned. With this in mind, given two capabilities of a pipeline $c_i, c_j \in p$, $i < j$ such that $e(c_i) \approxeq e(c_j)$, likely $E_E(c_i) > E_E(c_j)$. Note that there may be exceptions, such as conversions to larger formats or data augmentation capabilities.

With the definition of execution effort and Equation 4.8, it is possible to estimate the effort necessary to transmit the data object over the communication channel, hence the execution effort of the capability transmission $E_E(c_t)$. The unitary effort $e(c_t)$ is the effort required to transmit a single data item of the data object. The bigger the data object size N_d the higher the effort to transmit it, since, with reference to the assumptions made in Section 4.2.5, the only characteristic affecting the transmission of the data object is its size.

4.5. Data Friction in Data Exchanges

When setting up the exchange, the more capabilities have already been implemented the easier it is to start exchanging data. Conversely, if data provider and data consumer request capabilities yet to implement and execute, the data exchange will require more time to be set up.

Definition 4.4. *We define* ***data friction*** μ *the discrepancy between the total amount capabilities requested by the two parts and the amount of requested capabilities that have*

already been performed at the beginning of the data exchange preparation.

Going back to what we discussed in Section 4.3, the data exchange setup is composed of different phases. As stated in Section 4.4, the implementation phase depends on the complexity of the required capabilities while the execution phase depends on the data object and its dimension. Our definition of friction should take into account the differences between the various phases and should be further distinguished in different types of friction.

4.5.1. Implementation Friction

After having defined p_α and its deployment configuration $\widehat{p}_{\alpha,x}$, in the implementation phase the data provider implement the capabilities in p_α. Given the pipeline to implement p_α and the capabilities already implemented by the provider C_P, the more capabilities of p_α have already been implemented the lower the *implementation friction* of the data exchange. In other words, the more capabilities of p_α have already been implemented the easier it is to complete the implementation phase.

Definition 4.5. *We define* ***implementation friction coefficient*** μ_I *the ratio between the amount of capabilities that have already been implemented at the beginning of the implementation phase and the total amount of capabilities in the data pipeline.*

In symbols, given a pipeline p_α between the data provider and a consumer α and the set of previously implemented capabilities C_P, we can express what defined in Definition 4.5 as:

$$\mu_I = \frac{\sum_{c \in p_\alpha \backslash C_P} E_I(c)}{\sum_{c \in p_\alpha} E_I(c)}, \quad 0 \leq \mu_I \leq 1 \tag{4.9}$$

According to Equation 4.9, the implementation friction coefficient μ_I will be lower the more capabilities in p_α have already been implemented. Of course, in the definition of μ_I we have to take into consideration the implementation effort required by each capability: if a highly complex capability, hence with a high implementation effort, still has to be implemented, it is reasonable that the implementation friction will be high even though simpler capabilities have already been implemented.

To overcome the implementation friction, the total implementation effort of the data provider $E_{I,P}$ has to be equal or higher than the total effort required to implement the capabilities in $p_\alpha \backslash C_P$. In symbols:

$$E_{I,P} \geq \mu_I \sum_{c \in p_\alpha} E_I(c) \tag{4.10}$$

Drawing a parallel with the world of physics, the implementation friction (and the effort needed to overcome it) can be compared to the static friction force. Given a motionless object, to initiate its movement is needed a force at least equal to the object's normal force multiplied by the static friction coefficient of the surface on which the object is placed ($F \geq \mu_s N$). In Equation 4.10, to enable the movement of the data the provider needs to apply a force (the implementation effort) at least equal to the implementation friction of the pipeline, equal to the total implementation effort multiplied by the implementation friction coefficient μ_I

The implementation friction coefficient μ_I of a given pipeline p_α with a certain deployment configuration $\widehat{p}_{\alpha,x}$ depends on the system in which the pipeline is implemented. That is, depending on the characteristics of the system in which the implementation of the pipeline takes place, the value of μ_I will be different. The implementation friction is calculated at the instant of time t when the implementation phase begins. The characteristics of the system in which μ_I is calculated are evaluated at t. More specifically, to calculate μ_I, the capabilities already implemented by the provider are those in C_P at t.

4.5.2. Execution Friction

Once implemented, the capabilities in p_α have to be executed in order to obtain the data object d_α requested by the data consumer. Depending on the chosen deployment configuration $\widehat{p}_{\alpha,x}$ and the presence of already executed capabilities $\widehat{p}'_{\alpha,x}$, the difficulty to perform the data exchange will change.

Definition 4.6. *We define* ***execution friction*** *coefficient μ_E the ratio between the amount of capabilities that have already been executed at the beginning of the execution phase and the total amount capabilities in the pipeline.*

As said in Section 4.3.6, in some systems it is possible that not every capability of p_α has to be executed. Given a pipeline p_α and the set $\widehat{p}'_{\alpha,x}$, containing the capabilities already executed at the beginning of the execution phase, only those capabilities in p_α and not in $\widehat{p}'_{\alpha,x}$ have to be executed.

Given a pipeline p_α and a deployment configuration $\widehat{p}_{\alpha,x}$, the execution friction coefficient μ_E can be calculated as:

$$\mu_E = \frac{\sum_{c \in \widehat{p}_{\alpha,x} \setminus \widehat{p}'_{\alpha,x}} E_E(c)}{\sum_{c \in \widehat{p}_{\alpha,x}} E_E(c)}, \quad 0 \leq \mu_E \leq 1 \tag{4.11}$$

According to Equation 4.11, the more the capabilities already executed at the beginning of the execution phase, the lower the coefficient μ_E will be. Also in this definition we have to take into account the effort required by each capability to be executed. If a capability c has a high execution effort $E_E(c)$ and still has to be executed, μ_E will be substantially higher than if c has already been executed.

Note that in Equation 4.11 the capabilities are taken from the deployment configuration $\widehat{p}_{\alpha,x}$ and not from the data pipeline p. This is because we are also considering the transmission c_t of the data object, which is considered in the capabilities set of the deployment configuration and not in the data pipeline.

In highly dynamic contexts, where the basic data object d is constantly updated with new data, it is unlikely that some capabilities will already be executed, thus it will be likely that $\widehat{p}'_{\alpha,x} = \emptyset$. In these scenarios, μ_E will likely be close to 1. Conversely, the less frequent the updates to the data object d the higher the probability that some tasks have already been executed in previous data exchanges.

As in the previous phase, also in the execution phase provider and consumer effort has to be higher than the total effort required to execute the tasks in $\widehat{p}_{\alpha,x} \setminus \widehat{p}'_{\alpha,x}$:

$$\sum_{c \in \overleftarrow{p}_{\alpha,x} \setminus \widehat{p}'_{\alpha,x}} E_{E,P}(c) + E_E(c_t) + \sum_{c \in \overrightarrow{p}_{\alpha,x+1} \setminus \widehat{p}'_{\alpha,x}} E_{E,\alpha}(c) = \mu_E \sum_{c \in \widehat{p}_{\alpha,x}} E_E(c) \tag{4.12}$$

Going back to the physics parallel, if the implementation phase can be compared to the movement initiation, then in the execution phase the data objects needs to keep moving to complete the data exchange. In physics, the force applied to an object to continue its movement needs to be bigger than the object normal force multiplied by the dynamic friction coefficient ($F \geq \mu_d N$). Similarly, in the execution phase the joint effort of provider and consumer needs to be at least equal to the execution friction of the data pipeline, equal to the total execution effort multiplied by the execution friction coefficient μ_E.

If the system in which the data exchange takes place does not allow the partial execution of a pipeline, then there will not be a set $\widehat{p}'_{\alpha,x}$ and all the capabilities of $\widehat{p}_{\alpha,x}$ will have to be executed every time. In this situation the execution friction coefficient of the data exchange will always be $\mu_E = 1$.

As the implementation friction, the execution friction coefficient μ_E of a pipeline p with a

deployment configuration $\widehat{p}_{\alpha,x}$ depends on the characteristics of the system in which the pipeline is executed. μ_E is calculated at the instant of time t when the execution phase of the data exchange starts. That is, the already executed capabilities are those present in the set $\widehat{p}'_{\alpha,x}$ at t.

4.5.3. Simplifying Assumptions

The equation presented so far can be used to effectively calculate Data Friction in data exchanges. However, an easier estimation of Data Friction can be obtained by making some simplifying assumptions. In this way, although the obtained results will be less accurate, we will be able to calculate Data Friction also in absence of some information.

The assumptions made in this simplified model are the following:

Assumption 4.1. *All the capabilities have the same implementation effort* $E_I = 1$.

Although in real scenarios each capability requires different efforts for its implementation, it can be difficult and time consuming to estimate its effort through software cost models. In this way, each capability c has the same implementation effort E_I and the implementation friction coefficient μ_I is simply calculated as the ratio between the amount of capabilities yet to implement and the total amount of capabilities in the pipeline. In symbols, given a pipeline p_α and the capabilities already implemented by the provider C_P, the implementation friction coefficient μ_I can be calculated as:

$$\mu_I = \frac{|p_\alpha \backslash C_P|}{|p_\alpha|} \tag{4.13}$$

Assumption 4.2. *All the capabilities have the same execution effort* $E_E = 1$.

At the same time, it can be difficult in real scenarios to obtain the unitary effort $e(c)$ for a given capability c. In this way, each capability c has the same execution effort E_E and the execution friction coefficient μ_E is simply calculated as the number of capabilities yet to execute over the total number of capabilities in the pipeline. In symbols, given a pipeline p_α, a deployment configuration $\widehat{p}_{\alpha,x}$ and the already executed capabilities $\widehat{p}'_{\alpha,x}$, the execution friction can be calculated as:

$$\mu_E = \frac{|\widehat{p}_{\alpha,x} \backslash \widehat{p}'_{\alpha,x}|}{|\widehat{p}_{\alpha,x}|} \tag{4.14}$$

4.5.4. Example

With reference to the example introduced in Section 4.2.4, let's apply the equations presented in this chapter to estimate data friction and effort in the data exchange. We will make use the assumptions of Section 4.5.3 in order to keep calculations simple.

The pipeline $p_\alpha = \{c_1, c_2, c_3\}$ enabling the data exchange between provider and the consumer α is composed of three capabilities: c_1 filtering the patients, c_2 anonymizing their PII, and c_3 converting the result in a JSON array. With the deployment configuration $\widehat{p}_{\alpha,2}$ chosen for the data exchange, the capabilities deployed on the provider side will be $\overleftarrow{p}_{\alpha,2} = \{c_1, c_2\}$, while $\overrightarrow{p}_{\alpha,3} = \{c_3\}$ will be deployed consumer-side.

If, at the beginning of the implementation phase at t_0 the provider has no previously implemented capabilities, it will have to implement all the three capabilities in p_α. Since $C_P = \emptyset$ and $E_I(c) = 1 \forall c \in p_\alpha$, the implementation friction coefficient of the data exchange will be $\mu_I = 1$ and the total implementation effort required to the provider in order to make the exchange possible will be $E_{I,P} = 3$.

Similarly, in the execution phase $\widehat{p}'_{\alpha,2} = \emptyset$ and all three capabilities will have to be executed, with an execution friction coefficient of $\mu_E = 1$. The data provider will execute the capabilities in $\overleftarrow{p}_{\alpha,2}$ with an execution effort $E_{E,P} = 2$, then c_t will be executed and then the consumer α will execute the capabilities in $\overrightarrow{p}_{\alpha,3} = \{c_3\}$ with an execution effort $E_{E,\alpha} = 1$.

If, after completing the exchange with α, at t_1 another data consumer β requests the same data object but as a CSV file, then only part of the capabilities of p_β will have to be implemented and, assuming the system allows it, executed. Assuming $p_\beta = \{c_1, c_2, c_4\}$, where c_4 converts the result in a CSV file instead of a JSON list, the provider will only have to implement c_4.

The resulting implementation friction coefficient will be $\mu_I = \frac{1}{3}$, since $\{c_1, c_2\} \in C_P$ and only one out of the three capabilities in p_β will have to be implemented. In this case, the total implementation effort of the provider will be $E_{I,P} = 1$.

Similarly, assuming the original data object d is not updated between the two data exchanges, c_4 is also the only capability that has to be executed, with an execution friction coefficient of $\mu_E = \frac{1}{3}$. For instance, if the chosen deployment configuration for the data exchange is $\widehat{p}_{\alpha,3}$, then $E_{E,P} = 1$ and $E_{E,\beta} = 0$.

The already deployed p_α makes the deployment of p_β much faster and effortless, as proven by the low implementation friction coefficient $\mu_I = \frac{1}{3}$ and the low execution friction

coefficient $\mu_E = \frac{1}{3}$.

4.6. Friction-aware Evolution of Data Pipelines

If in the example of Section 4.5.4 the data provider and the data consumers were to implement a new pipeline from scratch for each data object, the friction in setting up the data exchange with β would be substantially higher.

Generally speaking, by creating new data pipelines (and their respective data objects) starting from existing ones, the effort necessary to overcome the frictions arising in new data exchanges can be significantly reduced both on provider and consumer side.

If in setting up new data exchanges providers and consumer do not reuse existing capabilities, the implementation friction will always be $\mu_I = 1$. This means that each data exchange will require a high implementation effort on both provider and consumer side. The result of this approach is that the time to set up new pipelines will be significantly high, possibily leading to failures in setting up new data exchanges.

On the other hand, by re-using capabilities in multiple data pipelines providers and consumers can reduce the effort in setting up new data exchanges. This is beneficial both in terms of workload, which will be significantly lower in the implementation phase, and in terms of time, since the data pipeline will be set up quicker.

Of course, in order to re-use capabilities in multiple pipelines, these have to be standardized and generalized enough to be used in more than a single use case. This can be obtained by breaking down complex, highly-specialized capabilities in simpler and smaller ones, each implementing a single atomic transformation. This approach to modeling data pipelines shares the principles of microservices, as well as the benefits brought by these architectures such as modularity, scalability and distributed development.

With reference to the example of Section 4.5.4, the capabilities c_1 and c_2 must have a level of standardization high enough to allow the provider to use them in multiple pipelines. With this, it is possible to implement only c_4 in p_β, without having to bother about the implementation of the previous capabilities, already in C_P since they have been implemented in a previous data exchange and are re-usable.

4.7. Chapter Conclusions

In this chapter we have defined a basic model that allows us to estimate data friction in a data exchange. We have defined the main components of the model and structured the

process to set up a data exchange. With this model, we are able to effectively calculate the implementation and execution friction and effort in a data exchange. A simple example proves how setting up a new data exchange can be made substantially easier if a friction-aware approach is adopted, resulting in a lower time to set up the pipeline and to effectively start sharing data.

During the chapter we have made some assumptions to keep our model simple and focus on the core aspects of data friction and effort: future works may further research those areas that have been left out of our study such as the establishment of the communication channel or the use of resources in the model.

Our model is able to represent data exchanges between a provider and various data consumers, but we did not consider the case in which a data consumer intends to set up a data exchange with multiple data provider. In an era where data integration is at the core of many data projects, future works may model data exchanges integrating multiple data objects from multiple providers. Similarly, our pipeline model does not consider the possibility of having multiple capabilities executed simultaneously: although this does not affect the effectiveness of our study, it could be interesting to enrich the model with the possibility of representing the parallelization of capabilities and studying its effects on data friction.

With the model defined in this chapter we are able now to move onto data sharing in the data mesh, being able to calculate data friction and identify how data exchanges can be enabled and optimized in such a scenario.

5 | Data Friction in the Data Mesh

We can now apply the model defined in Chapter 4 to data exchanges in a Data Mesh. By doing so, we extend the characteristic of the Data Mesh model presented in Chapter 3 to allow the set up of data exchanges with/between Data Products (DPs). We provide a model to establish data exchanges in the mesh in a friction-aware fashion and we study how it can ease the value generation along the DP chain.

5.1. Extended Data Mesh

The basic data mesh model introduced by Dehghani [14][15][16] and presented in Chapter 3 does not support the definition of data exchanges between data providers and data consumers. By modeling data exchanges with the structured approach of Chapter 4, a data mesh provides a method to enable the interfacing between provider and consumer. However, the current data mesh model must be extended to introduce the concepts of data pipelines and data exchanges.

The data mesh infrastructure and governance planes must support the definition and set up of data pipelines with reference to the model defined in Chapter 4. That is, data providers (hence DPs) and data consumers (either DPs or final users) should be able to set up data pipelines following the data exchange setup phases introduced in Section 4.3. Moreover, providers and consumers should be able to build those pipelines in a friction-aware way, enhancing value generation in the organization. This approach could be especially beneficial as the mesh grows, with an increasing number of deployed DPs and, consequently, an increasing number of data exchanges. By considering data friction when setting up new data exchanges, the time and effort required can be significantly reduced, resulting in easier data sharing and ultimately in higher value for the organization.

The model on which this chapter is based is an extended version of the Data Mesh, which we define as *Extended Data Mesh*. Note that in this chapter the terms Data Mesh and Extended Data Mesh will be used interchangeably, both referring to the latter.

5.1.1. Data Pipeline Model

In the extended data mesh, data pipelines between providers and consumers follow the model presented in Section 4.2, being composed by subsequent transformations following the structure of ETL pipelines. The capabilities introduced in our model are represented of *tasks*. Thus, in the extended data mesh data pipelines are composed by tasks, each representing an operation performed on data. A good example of the operations that can be implemented by a pipeline's tasks are those of Figure 5.1. However, depending on the mesh, and the pipeline, each task will have its peculiarities. It is worth considering that the more generalized a task is (i.e. by using parameters for its inputs), the higher its reusability will be in setting up new data pipelines.

With reference to the model presented in Chapter 4, when setting up data pipelines in the extended data mesh we must take into consideration the concept of domain [14]. More specifically, in a data exchange between a data provider and a data consumer, the data object must pertain to the data provider's business domain throughout the entire pipeline. The data pipeline ends when the data object does not belong to the data provider's domain.

With this in mind, the data pipeline is set up in the data provider's domain and the data consumer does not participate in the pipeline definition. However, the consumer can express some constraints on the pipeline, particularly on the location in which some tasks are deployed. For instance, a consumer may require that some tasks of the pipeline be deployed on specific resources. However, the provider handles the execution of all the tasks in the data pipeline, including those tasks deployed on the consumer's resources. If a task can be executed freely by the data consumer, then such a task belongs to the consumer's domain and thus does not belong to the data pipeline.

5.1.2. Data Pipelines Tasks

The tasks through which a data object in a pipeline is transformed may vary depending on the system in which the pipeline is deployed. In a data mesh, we can identify a set of tasks that covers most of the transformations of a data pipeline. By identifying this set, we will be able to better characterize data pipelines in a data mesh and track the transformations of a data object.

We propose a standardized set of tasks (Figure 5.1) based on the work of Raj et al. [43] and on Apache NiFi's processors [21]. The tasks are grouped into three macro-functions according to the three ETL stages of ingesting, transforming, and serving data. The tasks

are represented in a BPMN-like notation to allow the representation of data pipelines using BPMN diagrams.

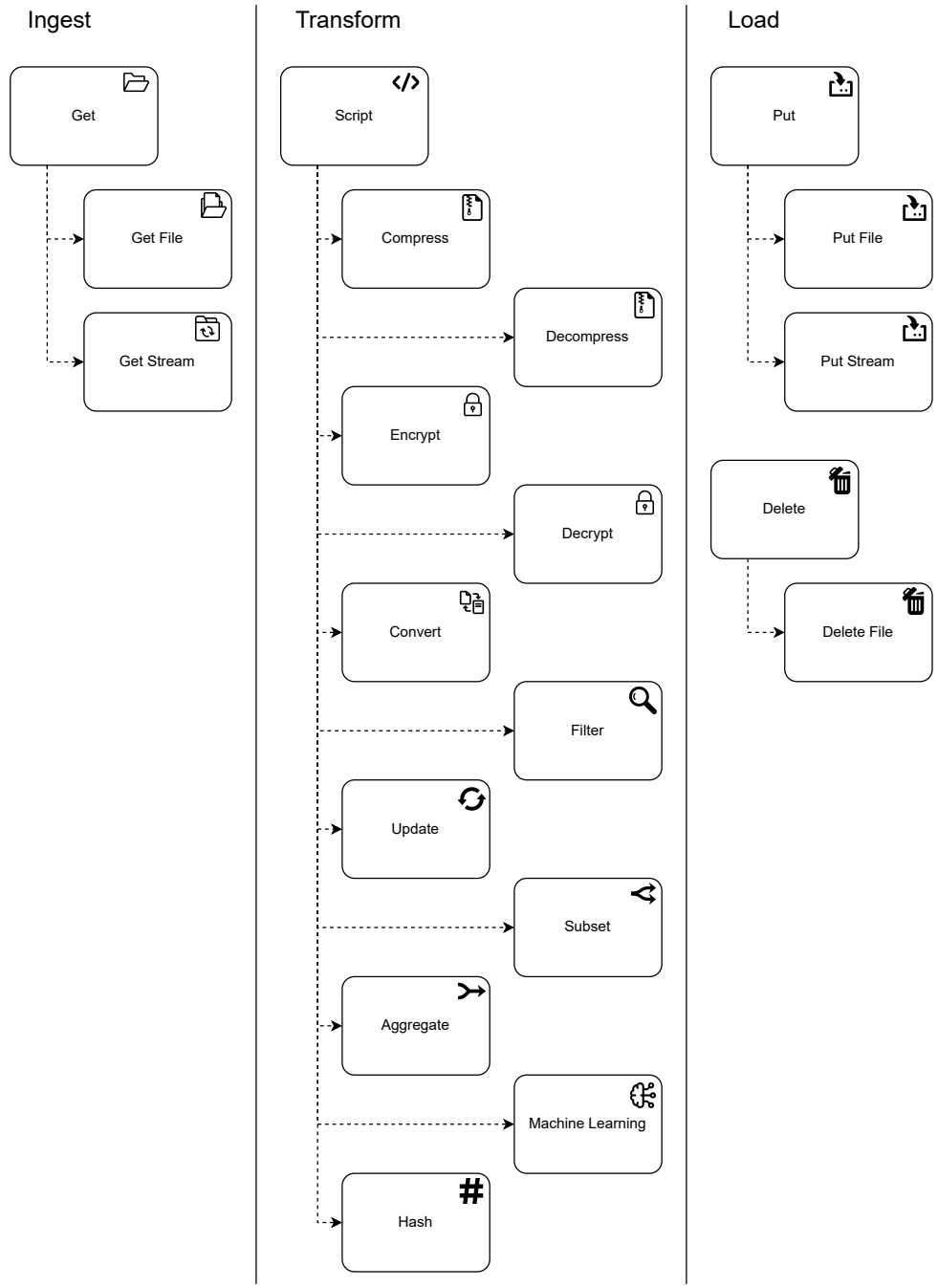

Figure 5.1: Data pipelines' task set.

The task set can vary across different DPs and different data meshes. However, we suggest that a uniform representation of capabilities can be helpful in terms of standardization and data integration. In a data mesh with a uniform set of tasks, data engineers can easily

understand how data are transformed throughout the various data pipelines. Moreover, a standard representation (and implementation) of tasks helps in terms of component implementation and re-usability.

The implementation of these capabilities and their parameters may vary between organizations and even between DPs of the same organization. For instance, the *encrypt* task may receive as input the data object and the encryption method and return the encrypted data object. The *encrypt* task and its respective transformation represented in Figure 5.2 can be expressed in pseudo-code as `data_object.encrypt(encryption_method='RSA')`.

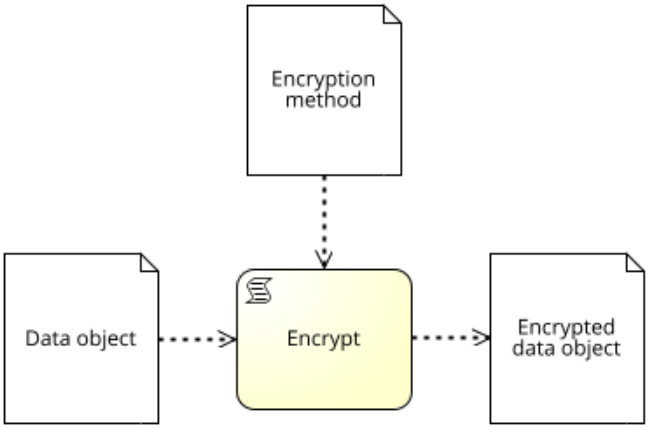

Figure 5.2: "Encrypt" task BPMN-like representation.

5.1.3. Data Product Model

With reference to the basic model presented in Chapter 4, in the Extended Data Mesh, a DP can be both a data provider and a data consumer. A DP ingests data from operational sources and other DPs, acting as data consumer, and at the same time serves data objects through its output ports to data consumers that can be either final users or other DPs. In this chapter, we will model data exchanges between DPs acting as data providers and a data consumer that can be either another DP or a final user, indifferently.

Each DP has its own *Data Steward*. According to Bode et al. [5], the data steward is the owner of the DP and is responsible for the overall data management. Moreover, the data steward should have complete knowledge of the existing use cases of the DP [5].

The characteristics of the data steward make it the most indicated figure to define the data exchange pipeline and choose the best configuration: having knowledge of the DP existing infrastructure, the data steward can define the tasks to implement in the pipeline

to obtain the desired data object.

5.2. Modeling Data Exchanges in the Extended Data Mesh

According to the model presented in Section 3.3, a DP serves data objects to data consumers through one or more output ports. For the sake of simplicity, we will start defining our model by considering as data provider a single DP with a single output port to then extend the model including DPs with multiple output ports.

With these assumptions, we can consider as the data provider a DP with a single output port, serving a basic data object d_0 to potential data consumers. The internal structure of the DP, its input ports, and its interaction with the other planes of the Data Mesh architecture are outside of the scope of this study.

On the other side of the data exchange, we consider a simple data consumer C_1 requesting a data object d_{C_1}. The internal structure of the data consumer is not relevant, as well as it is not relevant whether this actor is a DP or a final user of the Data Mesh.

The goal of the data exchange is to transform the data object d_0 in d_{C_1} to allow the communication between the provider DP and consumer C_1 (Figure 5.3).

Figure 5.3: Representation of the data exchange model between DPs.

It is possible that, in line with the provider model presented in Section 4.3.1, the provider DP has some previously implemented tasks in C_P. However, at t_0, we can assume that for the data provider its set of previously implemented tasks is initially empty $C_P = \emptyset$.

5.2.1. Pipeline Definition and Data Steward

In the agreement phase, once received the data consumer's request the provider defines the data pipeline p_{C_1}, the data object d_{C_1} resulting at the end of it, and the deploy-

ment configuration $\widehat{p}_{C_1,x}$. Depending on the mesh governance and its internal policies, the agreement process may be more or less structured. Similarly, it is possible that in some data meshes the deployment configuration will be more or less oriented towards the implementation of the entirety of capabilities by the provider or by the consumer. For instance, in line with the notion of Consumer-Driven Contracts [45], some data meshes may require that the implementation of the capabilities is performed entirely by the data provider.

The pipeline and deployment configuration must respect the eventual constraints imposed by the data consumer. Indeed, the data consumer may constrain the execution of a task to a certain location to have a better control of the process. However, the task execution is always controlled by the data provider, hence the owner of the data object. In a data exchange on the extended data mesh, it cannot happen that a task can be freely executed by the consumer.

In the Extended Data Mesh, the crucial figure in the agreement phase is the Data Steward. As said in Section 5.1.3, each DP of the data mesh has its own data steward. In the agreement phase, the request of d_{C_1} made by the data consumer C_1 is managed by the data steward of the provider DP. The data steward is in charge of designing the pipeline p_{C_1}. The data steward, having knowledge of the existing use cases, should be able to define the tasks to transform d_0 in an optimal way, promoting the reuse of already existing capabilities to minimize the friction in the data exchange. In this way, the data steward effectively reduces the effort needed to set up the exchange, in line with what discussed by Bode et al. [5].

The data steward of the DP acting as data provider defines also the deployment configuration $\widehat{p}_{C_1,x}$ accordingly to the constraints imposed by C_1. It is possible that a data pipeline is entirely implemented on the provider side or on the consumer side. However

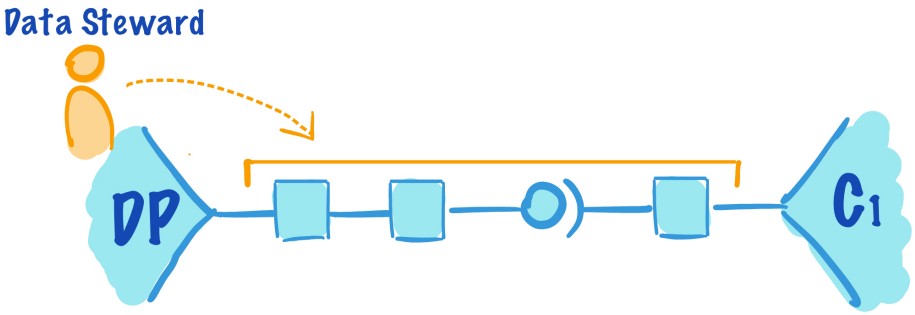

Figure 5.4: Representation of the pipeline between data provider and C1. The pipeline is defined by the provider DP's data steward.

Depending on the constraints on the execution location of its tasks, the pipeline will include more or less transmissions of the data object, affecting the total execution effort.

5.2.2. Implementation and Execution Phases

While the agreement phase is more subject to variations depending on the context, the implementation phase and the execution phase processes are more straightforward and aligned with their definition in Section 4.3.

Once p_{C_1} and $\widehat{p}_{C_1,x}$ are defined by the provider DP's data steward, in the implementation phase the data provider will implement the tasks that have not been already implemented, thus those tasks in $p_{C_1} \backslash C_P$. In the execution phase, the capabilities of the pipeline p_{C_1} will be executed in order to transform d_0 in d_{C_1} and to serve it to the data consumer C_1.

Note that, while in the model defined in Chapter 4 there is only one transmission from data provider to data consumer, this may not be the case in a data mesh. In fact, the tasks of the pipeline may be deployed on different resources with different locations, both on the provider and consumer side. If this is the case, then there will be more transmissions. However, in this study, we will consider only the data transmission from provider to consumer the only data transmission in the exchange. Future works may involve extending our model with multiple data transmissions.

5.3. Effort and Friction

The model presented in the previous sections allows the application of the equations of Chapter 4 to calculate or at least estimate the data friction in a data exchange between DPs and the effort required to overcome it. However, in a data mesh, we must consider how the system characteristics affect friction and effort in data exchanges, and how these characteristics can be taken into account.

5.3.1. Implementation Effort and Friction in Data Meshes

Calculating (or at least estimating) the implementation effort E_I of the task in a pipeline can be challenging when the data exchange takes place in a data mesh. The characteristics of the system can affect how E_I is calculated and the same capability may require different implementation efforts in different systems. It can also happen that different DPs in the same data mesh adopt different methods to estimate E_I, thus effort (and friction) may vary depending on which perspective is being considered. Moreover, in certain systems, it is possible that the implementation effort cannot be estimated at all.

Generally speaking, to effectively calculate/estimate the implementation friction coefficient μ_I in data exchanges in a data mesh requires a uniform view of the implementation effort. That is, all the actors must adopt a common method to estimate E_I, allowing them effectively calculate μ_I for a given data exchange. Several metrics can be adopted to calculate E_I. Software cost estimation models could be used as an estimate of the effort needed to implement a task or an entire pipeline. However, the application of these cost models requires time and effort and may not be possible in some circumstances. In these cases, simpler estimates of the effort could be obtained by referring to the implementation characteristics of the tasks. For instance, the effort can be estimated with the approximate number of lines of code needed for a given task or the approximate time needed for its implementation. In any case, the estimation of E_I is usually done in a qualitative way.

In those cases where the calculation of E_I is not possible or not effective, μ_I can be estimated by assuming constant implementation effort, in line with Assumption 4.1.

5.3.2. Execution Effort and Friction in Data Meshes

Depending on the system in which the data exchange takes place, the unitary execution effort $e(c)$ and, consequently, the execution effort E_E can be calculated experimentally.

Depending on the considered task *task*, different characteristics affect its unitary effort $e(task)$. As in the implementation effort, it is crucial that the system (i.e. the mesh) in which the data exchange takes place these characteristics are universally recognized and estimated in the same way. Moreover, in order to effectively use the equations provided in Chapter 4 the various efforts need to be dimensionally coherent.

Generally speaking, a universally coherent method to estimate the execution effort of a given task *task* could be to refer to its *execution time*. In this way, the execution effort $E_E(task)$ corresponds to the number of (milli)seconds needed to complete the execution of the task. For tasks that depend on the size of the data object N_d, the unitary effort can be estimated by dividing the execution time by the data object's size (Equation 5.1).

$$e(t) = \frac{E_E(task)}{N_d} \quad \left[\frac{s}{bits}\right] \tag{5.1}$$

In this way, the experimentally calculated execution effort $E_E(task)$ can be re-used to estimate the execution effort for the execution of the same task on different data objects.

5.4. Friction-Aware Evolution of Data Products

At the end of the data exchange, the set of previously implemented tasks C_P will contain the newly implemented ones from the pipeline p_{C_1}. As shown in Figure 5.5, after the data exchange has been set up the provider serves another data object $d_{0'}$ in addition to d_0. This data object, result of the partial transformation of d_0 into d_{C_1}, can be made available on the data mesh by publishing a new output port on the provider's Data Product Manifest (DPM), or it can be used only inside the pipeline p_{C_1}. If the partially transformed data object is added to the provider's DPM, then future data exchanges on the Data Mesh can be set up starting from it instead of d_0 (Figure 5.6).

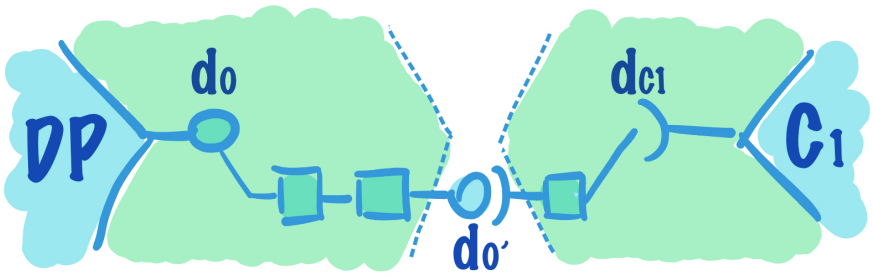

Figure 5.5: Internal representation of provider and consumer at the end of the data exchange set up.

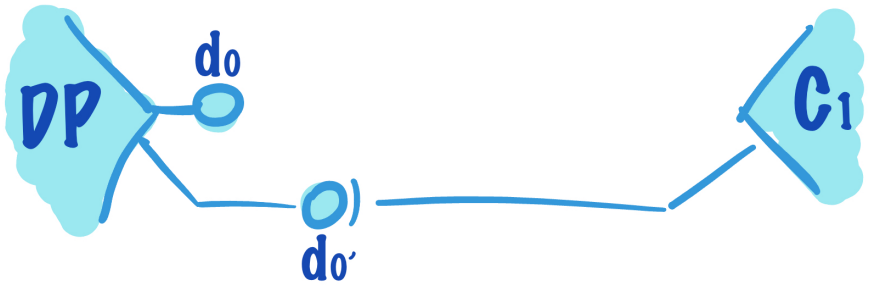

Figure 5.6: Provider and consumer at the end of the data exchange.

Depending on the presence of previously implemented capabilities, the data friction in setting up a new data exchange will be high or low. Of course, considering a data exchange between the provider and C_1 at t_0 where $C_P = \emptyset$, the implementation friction coefficient will be $\mu_I = 1$. As the provider DP implements new capabilities, added in C_P, these capabilities can be reused in setting up new data exchanges with data consumers, reducing the implementation friction coefficient μ_I in these forthcoming exchanges.

In the same way, the presence of partially transformed data objects can lower the execution friction of data exchanges between the provider and the various data consumers. Of course, for this to happen the data mesh implementation must allow the partial execution of data pipelines and store the partially transformed data objects.

5.5. Establishing Federations with the Extended Data Mesh

The extended data mesh allows setting up data exchanges between data providers and data consumers by defining data pipelines in accordance with the definition of business domain and with the imposed constraints. Moreover, these pipelines can be deployed in accordance with pre-agreed constraints such as the location of the execution of the pipeline's task. The data consumer may require that some tasks be executed on specific resources made available to the data provider to ensure that the task behavior respects the agreed SLOs. In this way, the behaviour and the results of the data pipeline can be monitored by provider and consumer, ensuring a trustable environment for the pipeline's execution.

These characteristics of data exchanges make the extended data mesh a suitable solution not only for intra-organizational data mesh implementations but also for setting up data meshes in environments with multiple organizations. By setting up inter-organizational data exchanges, the extended data mesh allows the establishment of a federated environment between organizations with a common goal achievable through data sharing. This scenario is particularly relevant in areas such as the healthcare sector, where data availability is low and data sharing is severely limited by regulatory policies such as privacy protection laws.

The *Federated Data Mesh* combines the principles and the platform architecture of the data mesh [14–16] with the principles of cloud federations [29]. In this way, separate organizations with a common goal are connected by a trusted and controlled data sharing environment where data can be exchanged in observance of regulatory frameworks and of the single organization's internal policies. By defining a data pipeline's tasks and specifying the pipeline's deployment configuration, providers and consumers belonging to different organizations can deploy such a pipeline according to the constraints at the organization and federation level. In this way, the federated data mesh allows controlled access to data objects to the members of the various organizations in a federation.

In this direction, Falconi and Plebani [20] propose the adoption of a federated data mesh

as a promising approach to enable data sharing in clinical trials. In such a federated approach, the data exchange model discussed in this chapter can be beneficial when defining the conditions under which data is shared among organizations and how the shared data is processed by both providers and consumers. Moreover, our friction-aware approach can effectively boost data sharing in such a federated data mesh, reducing the effort to establish connections between data providers and data consumers.

5.6. Chapter Conclusions

In this chapter, we have applied the data exchange model presented in Chapter 4 to data exchanges in data mesh. The application of our model allows the definition of data exchanges between provider and consumer in a structured and friction-aware approach.

By doing so, we have introduced an extended version of the traditional data mesh, namely the extended data mesh, which enables the interfacing between data providers and data consumers via the establishment of data pipelines. With this approach, not only do we provide a way to set up data exchanges in the data mesh but, by taking into consideration data friction, we provide also a method to deploy these data exchanges in an optimized way.

Finally, we observed how the extended data mesh can be deployed in inter-organizational contexts to enable data sharing inside federations. In this way, organizations with a common goal can set up a trusted environment for data exchanges. By modeling these data exchanges following our approach, providers and consumers in a federated data mesh can obtain better control of the shared data objects and how they are processed in data pipelines.

Future works in this area may deepen the application of our data exchange model in data meshes, exploring how such a model can be adapted to the various types of data exchanges taking place in a data mesh. While we explored how to model data exchanges in a data mesh, we did not explore how to manage data exchanges in the governance plane of the data mesh and how such a plane can be adapted and enhanced to ease data sharing in the mesh. Finally, future works may better explore the concept of federated data mesh, focusing on which key aspects of data sharing need to be considered when setting up friction-aware data exchanges in a federated environment.

6 | Implementation

In this chapter, we explore the practical applications of Data Friction and Effort within the context of a real-world scenario. The focus of our investigation revolves around the manipulation and management of a synthetic healthcare dataset. A series of data pipelines, orchestrated using the Apache Airflow framework, are built on top of the dataset in order to simulate the setup of new data exchanges. These pipelines will be used to estimate how data friction and effort vary depending on the used approach, highlighting best practices in the deployment of data pipelines.

6.1. Dataset

The DP and data pipelines implemented in this chapter are developed on top of the Synthea dataset for COVID-19 [51]. The Synthea project is an open-source synthetic patient generator that allows the creation of realistic, synthetic healthcare data [50]. The Synthea dataset is designed for several healthcare and medical research purposes, including software testing, data analysis, and machine learning applications. The synthetic data generated by Synthea is not derived from real patient data but is designed to resemble it, providing a suitable alternative for situations where regulatory frameworks and privacy protection laws limit access to real-world healthcare datasets.

The Synthea dataset includes a wide range of health-related information, such as patient demographics, medical conditions, medications, procedures, and more. Researchers and developers can use this data to test healthcare software, develop and evaluate healthcare algorithms, and perform data analysis and research without exposing real patient information.

6.1.1. Synthetic Data

The use of synthetic data can be particularly useful in scenarios where the access and usage of real-world datasets are highly restricted due to privacy and legal concerns.

A synthetic dataset is artificially generated rather than being collected from real-world

sources. Synthetic data mimics the characteristics and patterns of real data without containing any sensitive, confidential, or personally identifiable information (PII). The rise of generative AI technology has brought significant improvements in the generation of synthetic datasets, that can be used in several applications such as model testing and data augmentation.

Synthetic data has various applications across industries, including healthcare, finance, cybersecurity, and more, where data privacy and confidentiality are crucial concerns. In industries where data access is highly restricted, synthetic datasets can be used in experimental setups for research purposes and the development of new solutions [51].

6.1.2. Synthea Schema

The Synthea dataset is a synthetic dataset containing healthcare information, organized in 16 tables containing entries about patients, their encounters, medical conditions and observations, healthcare insurance plans, and more. Figure 6.1 contains a representation of the schema of the Synthea dataset, describing its tables and their attributes.

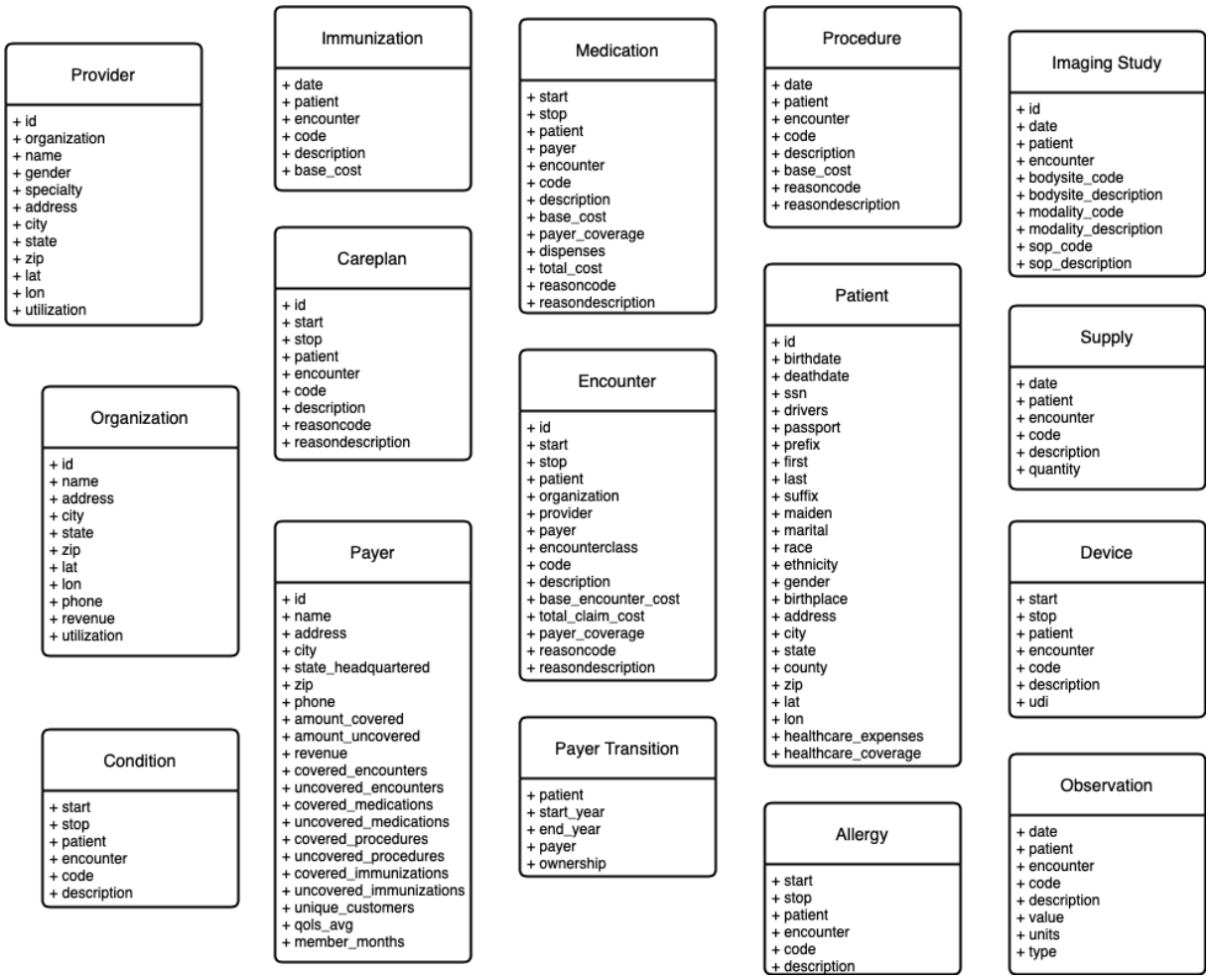

Figure 6.1: Synthea dataset schemas and attributes.

6.1.3. Dataset Implementation

Only some tables of the Synthea dataset have been used for the implementation of the project. Since the model considered in this study consists of one data provider and multiple data consumers, the data provider must handle data objects belonging to a single business domain in line with the data mesh principles.

With this in mind, only the `patients` and `encounters` tables have been effectively considered from the entirety of the Synthea dataset. These tables, initially stored as CSV files, have been imported into a MySQL database. This database acts as the operational foundation upon which the DPs of the data mesh will be implemented. Of course, in a real-world data mesh implementation, all the tables of the dataset will be used, served by various data providers according to the separation in business domains.

6.2. Data Product Definition

After building an operational data source, a simple DP serving data objects regarding patients has been developed in order to simulate the setup of data exchanges with data consumers. This DP will act as the provider of the various data exchanges, which will start from already served data objects.

The DP has been implemented to extract and serve data directly from the operational data source (the MySQL database containing the two tables). In line with the principles of the data mesh, the DP serves data to consumers via REST APIs accessible through HTTP requests. The API interfaces were implemented using the Spring framework, accessing the MySQL database via the Java Persistence API (JPA) and serving the extracted information using the Spring Web module.

The *Patients* DP extracts and serves information about the patients registered in the database. According to the classification provided in Section 3.3.2, this DP is a basic, source-aligned DP, since it ingests data from an operational source to serve it to advanced DPs in the mesh. The *Patients* DP exposes 4 API endpoints, described in Table 6.1.

Endpoint	**Description**
`/all`	Return a JSON array containing the anagraphic information of all the patients stored in the database.
`/csv`	Return a CSV file containing the anagraphic information of all the patients stored in the database.
`/{id}`	Return the anagraphic information of the patient with the id specified as parameter.
`/{id}/encounters`	Return the list of encounters of the patient with the id specified as parameter.

Table 6.1: Description of the *Patients* DP's API endpoints.

The API endpoints of the *Patients* DP can be described with the OpenAPI specification, in line with Assumption 3.1:

```
paths:
  /patients/{id}:
    get:
```

```
    tags:
      - patient-controller
    operationId: getPatientById
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: string
    responses:
      '200':
        description: OK
        content:
          '*/*':
            schema:
              $ref: '#/components/schemas/Patient'
/patients/{id}/encounters:
  get:
    tags:
      - patient-controller
    operationId: getPatientEncounters
    parameters:
      - name: id
        in: path
        required: true
        schema:
          type: string
    responses:
      '200':
        description: OK
        content:
          '*/*':
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Encounter'
/patients/all:
```

```
  get:
    tags:
      - patient-controller
    operationId: getAllPatients
    responses:
      '200':
        description: OK
        content:
          '*/*':
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Patient'
/patients/all/csv:
  get:
    tags:
      - patient-controller
    operationId: downloadCSV
    responses:
      '200':
        description: OK
```

The schemas referenced in the OpenAPI specifications for patients and encounters are those of the Synthea model, shown in Figure 6.1.

The implemented API endpoints represent the output ports of the *Patients* DP. Each of these output ports serves a certain data object to potential customers. In line with the model defined in the previous chapter, if a data consumer (either another DP or a final user) needs a data object already served by an output port, it can simply make an HTTP request to obtain it. Otherwise, a data pipeline can be built starting from the existing data objects to serve the data consumer's request.

6.3. Data Pipelines Implementation

If a data consumer needs a data object not already served by the DP output ports, a data pipeline can be built in order to obtain it. This phase of the data exchange setup follows the model presented in the previous chapter: a data consumer of the data mesh requests

a data object not already served by the provider (the DP) and a data pipeline is set up between the two parts in order to enable the data exchange.

The pipelines built in this section have been implemented using Apache Airflow. Apache Airflow allows the deployment of Directed Acyclic Graphs (DAGs) representing data pipelines. A DAG is composed of tasks, each representing, with reference to our model, a capability performing some kind of transformation to the input data object. The data sources used for the implemented DAGs are the APIs from the DPs defined in Section 6.2 and the pipelines are built on top of the data objects ingested from the sources, in line with the model presented in this study.

By implementing the data pipelines in Airflow, we can study how data friction and effort can be calculated in a real-world scenario. Furthermore, we can understand how different approaches can lead to lower, or higher, frictions in setting up new data exchanges between provider and consumer.

Although Airflow does not allow the separation of a pipeline between the data provider and data consumer, we can still study how different deployment configurations affect effort and friction in data exchange by considering the output of the different tasks of a pipeline. In other words, by looking at the size of the output of a given task from a data pipeline, we can obtain an estimate of the effort required to transmit such data object from provider to consumer. Since in our model the execution effort of transmitting a data object, represented by $E_E(c_t)$, depends only on the size of the data object N_d (Section 4.2.5), the bigger the data object the higher the effort in transmitting it from provider to consumer.

6.3.1. Definition of the use-case pipelines

In the current example, let's imagine that the *Patients* DP, as the data provider, sets up a series of data exchanges with data consumers. For each data exchange, the data consumer requests a data object not already served by the data provider, hence requiring a data pipeline to transform a data object into the requested one.

For the sake of simplicity, the pipelines will start from the `/csv` endpoint returning a CSV list of all the patients in the database, representing the data object d_0. Depending on the data object requested by the data consumers, this data object d_0 will be transformed according to the various pipelines.

We will consider three data consumers, C_1, C_2, and C_3, each requiring a variation of d_0 obtainable through a data pipeline. The three data pipelines p_{C_1}, p_{C_2}, and p_{C_3} between

provider and, respectively, C_1, C_2, and C_3 are described in Figure 6.2. The three pipelines present some similar tasks and slightly different outputs. The definition of such pipelines is outside the scope of this example and the data pipelines p_{C_1}, p_{C_2}, and p_{C_3} should be assumed already defined. In a real-world scenario, the data steward of the *Patients* DP and the various data consumers will define the respective data pipelines in line with the steps of the agreement phase discussed in Section 4.3.

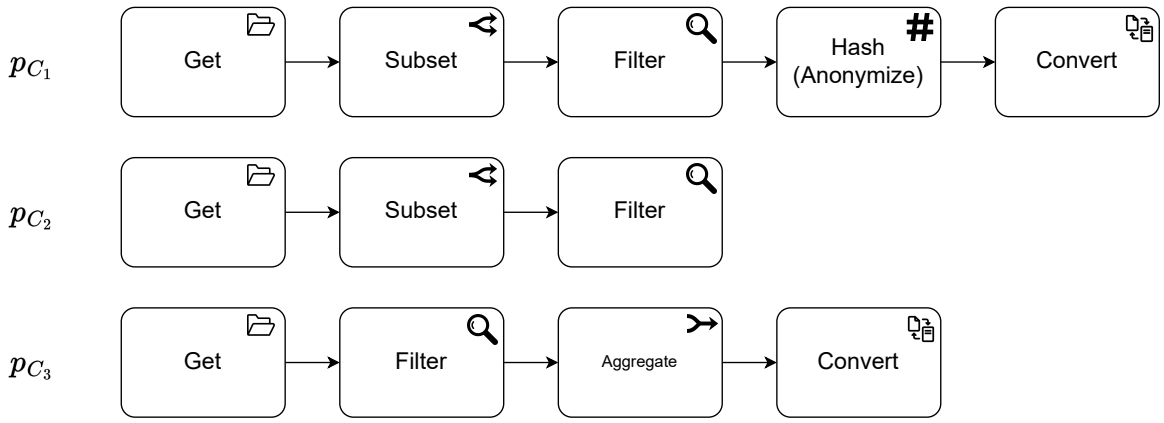

Figure 6.2: BPMN-like representation of the three pipelines p_{C_1}, p_{C_2}, and p_{C_3}.

More specifically, the three data consumers require the following data objects:

- The data consumer C_1, given the data object d_0, requires the data object d_{C_1} containing a list of the patients living in a certain area. The list must be in JSON format and the patients' PII must be anonymized. d_{C_1} can be obtained from d_0 by removing unnecessary patients' information, then filtering the patients living in the desired area, anonymizing with a simple hash function those patients' PII and finally converting the CSV list into a JSON array;
- The data consumer C_2, given the data object d_0, requires the data object d_{C_2} containing a list of the patients living in a certain area. The list must be in CSV format and the patients' PII does not need to be anonymized. The steps needed to obtain d_{C_2} are those needed to obtain d_{C_1}, with the exception of the anonymization and conversion tasks;
- Finally, the data consumer C_3, given d_0, requires the data object d_{C_3} containing, for each ZIP code in a list of ZIP codes, the number of patients living in that area as an XML file. d_{C_3} can be obtained from d_0 by filtering the patients who live in the specified ZIP codes, then counting the patients for each ZIP code, and finally converting the result into an XML file.

To estimate the effort and friction of each pipeline (and each data exchange), we have first of all to define the metrics used to estimate the efforts of each task of the three pipelines.

For the task implementation, we can use the number of lines of code (LoC) required by each task to estimate the implementation effort. Note that this technique does not provide a realistic estimate of the effort required to implement a task, as the complexity of a task depends on a much larger variety of factors. However, in the implemented pipeline the single tasks are fairly easy and LoC provides a fair estimate of the implementation effort E_I required by each of them. By counting the LoC of each task and summing them we can obtain the total implementation effort of the pipeline.

For what concerns the execution, we can refer to the execution time of each task as a proxy for its execution effort, in line with what discussed in Section 5.3.2. By monitoring the pipeline execution on Apache Airflow we can obtain the execution time, hence the execution effort, of each task and of the entire pipeline.

Finally, concerning the execution effort of the data object transmission $E_E(c_t)$, we will consider the size of the data object as the only characteristics affecting the effort. We will assume that the data object is transmitted on a communication channel with bandwidth $B = 1MBps$. Thus, to calculate $E_E(c_t)$ we will use the following equation:

$$E_E(c_t) = \frac{N_d}{B} \tag{6.1}$$

where N_d is the size of the transmitted data object.

Note that, in this proof of concept implementation, the implementation and execution effort of a given task are not estimated a priori. In real-world implementations, data friction should be calculated before and not after the completion of the data exchange. In this study, we retrieve the implementation/execution effort of the tasks a posteriori, for the sake of simplicity. We argue that, given a large enough observation basis, it should be then possible to estimate a task's implementation/execution effort based on the previously implemented/executed tasks.

6.3.2. Pipeline #1

In the pipeline p_{C_1} between provider and C_1 the task of anonymizing the dataset can only be deployed on the provider side. In symbols, given the anonymization task $Hash$ and the set of that tasks that can only be deployed provider-side $\overleftarrow{C}$, then $Hash \in \overleftarrow{C}$. Furthermore, we will assume that also the task Get must be deployed provider-side,

but given the presence of $Hash$ this does not affect the amount of possible deployment configurations.

With this in mind, there are only two possible deployment configurations for p_{C_1}, $\widehat{p}_{C_1,4}$ and $\widehat{p}_{C_1,5}$, represented in Figure 6.3. Depending on the chosen deployment configuration, effort and friction arising in the data exchange will vary.

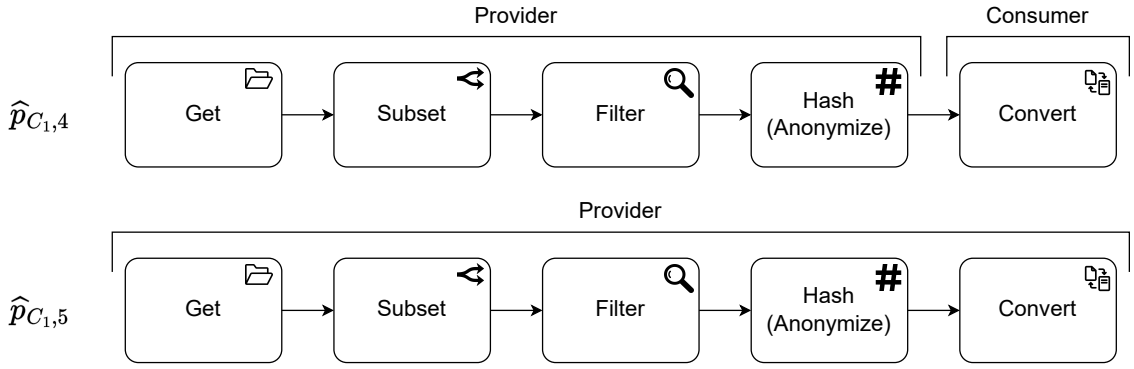

Figure 6.3: Possible deployment configurations for the pipeline p_{C_1}.

Assuming the data exchange between provider and C_1 begins at t_0, the data provider has no previously implemented tasks ($C_P = \emptyset$). Thus, in the implementation phase all the tasks in p_{C_1} will have to be implemented.

Table 6.2 contains, for each task in p_{C_1}, its implementation effort $E_I(task)$ measured as the number of LoC of the task.

Task	$E_I(task)$
`get_patients`	16
`subset_patients`	8
`filter_patients_by_zip`	8
`anonymize_pii`	12
`csv_to_json`	12
$E_{I,P}$	**56**

Table 6.2: Implementation effort $E_I(task)$ of each task of the pipeline p_{C_1}.

After having implemented its tasks, the pipeline p_{C_1} is successfully deployed on Airflow (Figure 6.4) and ready to be executed.

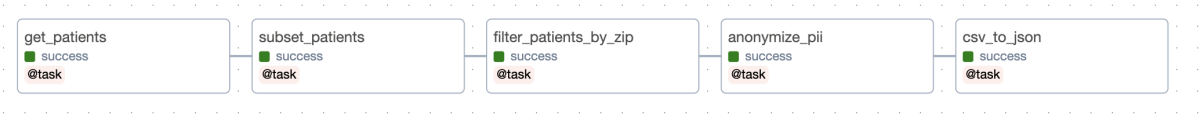

Figure 6.4: Airflow DAG of pipeline p_{C_1}.

By looking at the execution time of each task in p_{C_1} we can obtain the execution effort $E_E(task)$ of the tasks in p_{C_1}. The execution efforts of the tasks are reported in Table 6.3

Task	$E_E(task)$
`get_patients`	1.117161
`subset_patients`	0.391777
`filter_patients_by_zip`	0.284446
`anonymize_pii`	0.222661
`csv_to_json`	0.270467
$E_{E,P}$	**2.286512**

Table 6.3: Execution effort $E_E(task)$ of each task of the pipeline p_{C_1}.

Being p_{C_1} the first pipeline to be implemented, by definition it will have maximum implementation friction $\mu_I = 1$ and maximum execution friction $\mu_E = 1$. However, the choice of the deployment configuration affects the total execution effort of the data exchange: the two possible deployment configurations, $\widehat{p}_{C_1,4}$ and $\widehat{p}_{C_1,5}$, transmit different data objects from provider to consumer (Figure 6.3). In $\widehat{p}_{C_1,4}$ the transmission of the data object takes place after the PII anonymization, while in $\widehat{p}_{C_1,5}$ the data object is transmitted after being converted to JSON.

Based on our model, the transmission c_t of the data object depends on its size. By looking at the two objects' size, we can identify which object would require less effort $E_E(c_t)$ for its transmission. Table 6.4 reports the size of the data object to be transmitted in the two deployment configurations.

Dep. config.	Task before c_t	Output size	$E_E(c_t)$
$\widehat{p}_{C_1,4}$	`anonymize_pii`	23 KB	0.23
$\widehat{p}_{C_1,5}$	`csv_to_json`	54 KB	0.54

Table 6.4: Size of the data object to be transmitted in the two deployment configurations of p_{C_1} and respective execution effort $E_E(c_t)$.

Intuitively, the deployment configuration $\widehat{p}_{C_1,4}$ requires less execution effort because the size of the object to be transmitted is significantly lower than the one of $\widehat{p}_{C_1,5}$. With this in mind, in absence of other factors the deployment configuration $\widehat{p}_{C_1,4}$ should be preferred as it has a lower total execution effort E_E.

6.3.3. Pipeline #2

Assuming that the data exchange between provider and C_1 takes place successfully and ends at t_1, the provider when setting up future data exchanges will have already implemented the tasks of p_{C_1}. In symbols, $C_P = \{get, subset, filter, hash, convert\}$.

After completing the data exchange between provider and C_1, C_2 wants to set up a data exchange with the provider. Given the data object d_{C_2} requested by C_2 and the pipeline p_{C_2} such that $d_{C_2} = p_{C_2}(d_0)$, there are different deployment configurations that can be chosen (Figure 6.5) since the only task in $p_{C_2} \cap \overleftarrow{C}$ is *get* and there are no tasks in $p_{C_2} \cap \overrightarrow{C}$.

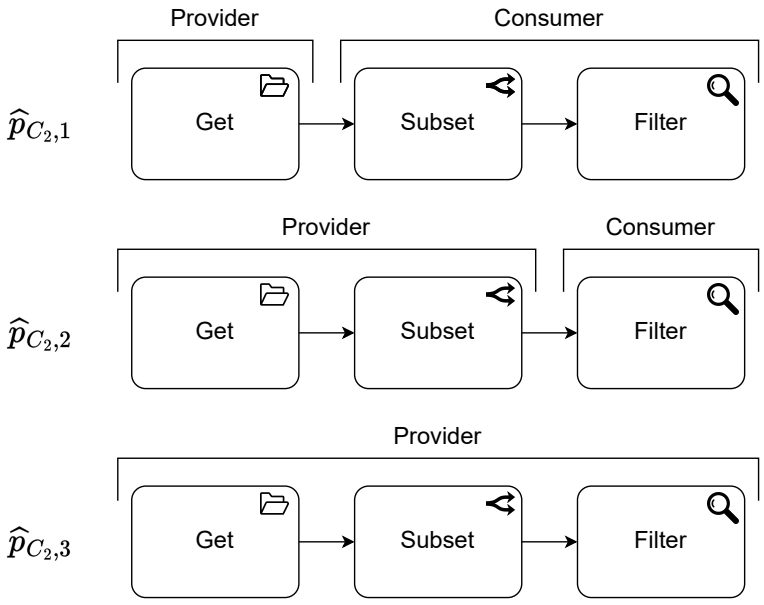

Figure 6.5: Possible deployment configurations for the pipeline p_{C_2}.

The implementation effort of the various tasks in p_{C_2} is the same of the corresponding tasks in p_{C_1} and can be found in Table 6.2. However, since the three tasks have been already implemented in p_{C_1} and are in C_P, there is no need to implement them again. With this in mind, the total implementation effort of the provider is null $E_{I,P} = 0$ and there is implementation friction $\mu_I = 0$ in this data exchange.

After having implemented its tasks, the pipeline p_{C_2} is successfully deployed on Airflow (Figure 6.6) and ready to be executed.

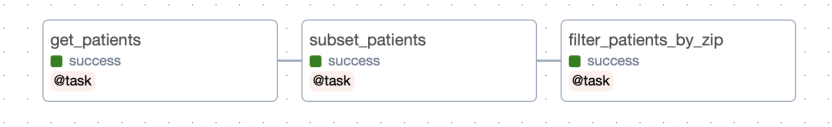

Figure 6.6: Airflow DAG of pipeline p_{C_2}.

For what concerns the execution phase, the execution effort E_E and friction μ_E depend on the chosen deployment configuration.

First of all, assuming the system in which the data exchange takes place allows to store the partial result of a pipeline, and that d_0 has not been modified since t_1, it may be possible that not every task has to be executed. In fact, the data provider has already available the output data object of all the three tasks in p_{C_2}. All the tasks deployed provider side will not have to be executed since their output is already available. With this in mind, regardless of the deployment configuration, the execution effort of the provider will always be null $E_{E,P} = 0$.

On the other hand, the consumer C_2 has not already executed any of the tasks in p_{C_2} and, regardless of the chosen deployment configuration, its execution effort will always be equal to the sum of the execution efforts of the task deployed on its side.

Table 6.5 reports the execution effort for each task in p_{C_2}, measured as its execution time.

Task	$E_E(task)$
`get_patients`	1.506339
`subset_patients`	0.632422
`filter_patients_by_zip`	0.258526

Table 6.5: Execution effort $E_E(task)$ of each task of the pipeline p_{C_2}.

Finally, depending on the chosen deployment configuration the data object that will be transmitted with the transmission task c_t will have a different size. Table 6.6 reports the size of the data object to be transmitted in the available deployment configurations.

Dep. config.	**Task before c_t**	**Output size**	$E_E(c_t)$
$\widehat{p}_{C_2,1}$	`get_patients`	3.9 MB	3.9
$\widehat{p}_{C_2,2}$	`subset_patients`	826 KB	0.826
$\widehat{p}_{C_2,3}$	`filter_patients_by_zip`	23 KB	0.023

Table 6.6: Size of the data object to be transmitted in the three deployment configurations of p_{C_2} and respective execution effort $E_E(c_t)$.

To sum up, depending on the chosen deployment configuration, there will be different total execution efforts and different execution frictions in the data exchange. The different execution efforts and execution frictions are reported in Table 6.7.

Dep. config.	$E_{E,P}$	$E_E(c_t)$	E_{E,C_2}	E_E	μ_E
$\widehat{p}_{C_2,1}$	0 (1.506339)	3.9	0.890948	4.790948	0.76
$\widehat{p}_{C_2,2}$	0 (2.138761)	0.826	0.258526	1.084526	0.34
$\widehat{p}_{C_2,3}$	0 (2.397287)	0.023	0	0.023	0.01

Table 6.7: Execution efforts and execution friction of the various deployment configurations of p_{C_2}.

Note that in Table 6.7 the execution effort of the provider $E_{E,P}$ is always 0 since it does not have to execute any task. However, the execution friction μ_E is calculated as E_E over the total effort of the pipeline without any previously executed task, in line with Equation 4.11.

According to the results in Table 6.7, the deployment configuration $\widehat{p}_{C_2,3}$ is by far the best one since it does produce an almost null execution friction. Intuitively, this makes sense since the provider has already stored the data object requested by C_2 as a partial transformation of the previously executed p_{C_1} and it only needs to transmit to C_2 a very small data object.

6.3.4. Pipeline #3

At t_2 the data exchange between provider and C_2 is successful and completed. After implementing and executing p_{C_2}, the status of C_P does not change since the tasks in p_{C_2} were already implemented.

After completing the data exchange with C_2, another data consumer C_3 sets up a data exchange with the provider. Given the data object d_{C_3} and the pipeline p_{C_3} such that $d_{C_3} = p_{C_3}(d_0)$, there are different deployment configurations that can be chosen (Figure 6.7) since there is only the *get* task in $p_{C_3} \cap \overleftarrow{C}$ and no tasks in $p_{C_2} \cap \overrightarrow{C}$.

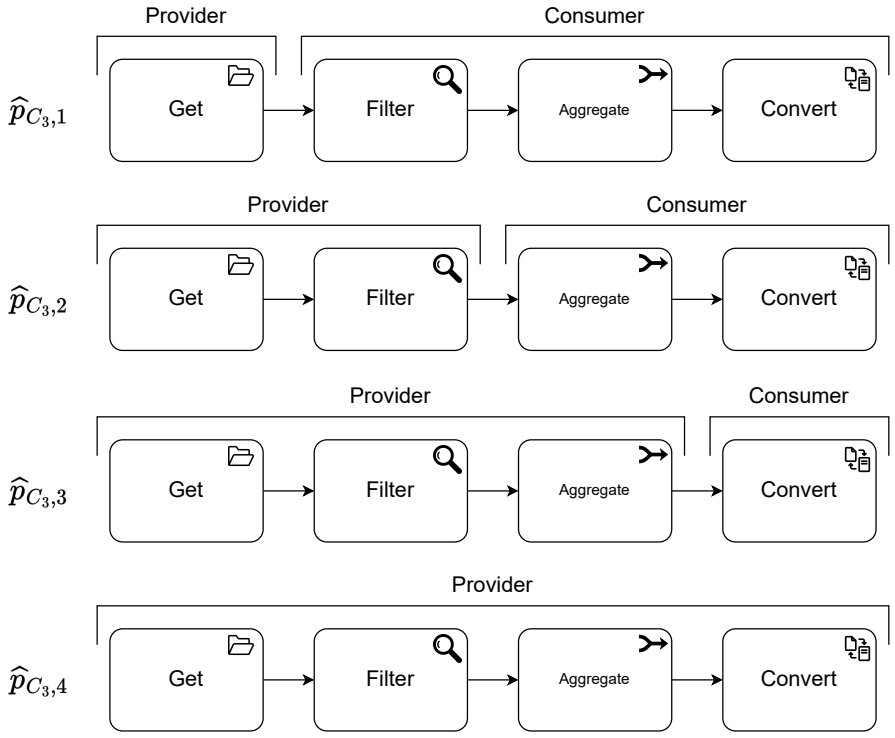

Figure 6.7: Possible deployment configurations for the pipeline p_{C_3}.

In the implementation phase, the data provider has to implement two of the four tasks in p_{C_3} since $get, filter \in C_P$. The implementation effort of the tasks of p_{C_3} is reported in Table 6.8.

Task	$E_I(task)$
`get_patients`	16
`filter_patients_by_zip`	8
`count_patients_by_zip`	8
`csv_to_xml`	15
$E_{I,P}$	**47**

Table 6.8: Implementation effort $E_I(task)$ of each task of the pipeline p_{C_3}.

However, the provider only has to implement the *aggregate* and *convert* (since the *convert* in p_{C_1} converts from CSV to JSON while this one converts to XML). Thus, the total implementation effort is $E_{I,P} = E_I(aggregate) + E_I(convert) = 23$ and the implementation friction is $\mu_I = \frac{23}{47} = 0.49$.

Once implemented, the pipeline p_{C_3} is deployed in Airflow and ready to be executed (Figure 6.8).

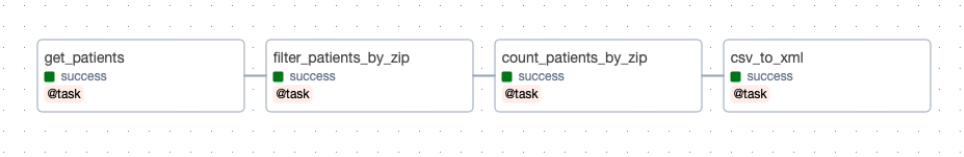

Figure 6.8: Airflow DAG of pipeline p_{C_3}.

Once again, depending on the chosen deployment configuration, in the execution phase there will be different execution efforts for provider and consumer and different execution friction.

In this case, the provider has already available the result of the *get* task. Note that the output of the *filter* task in this pipeline is different than the one of the previously executed pipelines, thus the execution of such task is mandatory. On the other hand, the consumer will have to execute all the tasks deployed on its side.

Table 6.9 reports the execution effort for each task in p_{C_3}, measured as its execution time.

Task	$E_E(task)$
`get_patients`	1.059806
`filter_patients_by_zip`	0.324634
`count_patients_by_zip`	0.231703
`csv_to_xml`	0.207368

Table 6.9: Execution effort $E_E(task)$ of each task of the pipeline p_{C_3}.

As in the previous pipelines, depending on the chosen deployment configuration the data object to be transmitted will have different sizes. Table 6.10 reports the size of the data object to be transmitted in the available deployment configurations.

Dep. config.	Task before c_t	Output size	$E_E(c_t)$
$\widehat{p}_{C_3,1}$	`get_patients`	3.9 MB	3.9
$\widehat{p}_{C_3,2}$	`filter_patients_by_zip`	101 KB	0.101
$\widehat{p}_{C_3,3}$	`count_patients_by_zip`	94 B	≈0
$\widehat{p}_{C_3,4}$	`csv_to_xml`	539 B	≈0

Table 6.10: Size of the data object to be transmitted in the three deployment configurations of p_{C_3} and respective execution effort $E_E(c_t)$.

According to the deployment configuration there will be different execution efforts for provider and consumer and different execution friction. The results of the different deployment configurations are reported in Table 6.11.

Dep. config.	$E_{E,P}$	$E_E(c_t)$	E_{E,C_3}	E_E	μ_E
$\widehat{p}_{C_3,1}$	0 (1.059806)	3.9	0.763705	4.663705	0.81
$\widehat{p}_{C_3,2}$	0.324634 (1.38444)	0.101	0.439071	0.864705	0.45
$\widehat{p}_{C_3,3}$	0.556337 (1.616143)	≈0	0.207368	0.763705	0.42
$\widehat{p}_{C_3,4}$	0.763705 (1.823511)	≈0	0	0.763705	0.42

Table 6.11: Execution efforts and execution friction of the various deployment configurations of p_{C_3}.

Based on the results in Table 6.11, the deployment configurations $\widehat{p}_{C_3,3}$ and $\widehat{p}_{C_3,4}$ yield the

minimum amount of execution effort $E_E = 0.76$ and execution friction $\mu_E = 0.42$. This is due to the fact that, as shown in Table 6.10, the two transmitted data object are very small in size and the execution effort for their transmission $E_E(c_t)$ is approximately 0. With this in mind, either $\widehat{p}_{C_3,3}$ or $\widehat{p}_{C_3,4}$ are the deployment configurations to be preferred in this data exchange. However, for data exchanges involving larger amount of data, $\widehat{p}_{C_3,3}$ should be preferred since its transmitted data object is significantly smaller than the transmitted one in $\widehat{p}_{C_3,4}$.

6.4. Discussion

The obtained results show that, by adopting a friction-aware approach in setting up data exchanges, the setup process discussed in Section 4.3 can be made easier and require less effort from the two parts. The results show that by being able to estimate the implementation and execution effort of a pipeline and its various deployment configurations, data providers and data consumers can choose the best configuration to set up the data exchange. In a world where the ability to extract value from data is key, being able to set up data exchanges at a faster rate can give an organization a significant competitive advantage.

If the pipelines p_{C_1}, p_{C_2} and p_{C_3} were to be deployed without considering the implementation and execution efforts and frictions, data provider and data consumer may choose a configuration that requires higher effort to complete the data exchange, resulting in a waste of time and resources. For instance, if in the pipeline p_{C_3} had been chosen the deployment configuration $\widehat{p}_{C_3,1}$, the effort in completing the data exchange would be significantly higher: every time that C_3 requests d_{C_3}, the time required to execute p_{C_3} would be much higher and C_3 would have to wait longer to be able to work with the requested data object.

Overall, the implementation shows how the model and the equations defined in Chapter 4 can be used effectively to estimate friction and effort in real-world data pipelines and exchanges. The obtained results and values are in line with what one would expect when setting up a data exchange and with data sharing's best practices.

6.5. Limitations and Future Works

Although the implementation shows the validity of our model, there are a few limitations derived from the implementation choices that are worth considering. Given the implementation in Apache Airflow, there are aspects of the model presented in Chapter 4 that

cannot be fully implemented in such a framework.

First of all, while Apache Airflow provides a framework to define pipelines in a similar way to our data pipeline definition, it lacks the distinction between provider and consumer. Thus, a pipeline in Airflow is deployed only by one actor, and the separation between provider and consumer has to be virtually simulated on top of the deployed pipeline. In our case, for the obtained results we simulated the various deployment configurations mathematically although the pipelines were entirely deployed by a single actor.

With this in mind, the transmission of the data object has to be simulated. Since there is no separation between the two sides, the transmission c_t and its execution effort were manually calculated based on the data object dimension and added mathematically to the experimentally obtained results.

Finally, although we considered the possibility of partially executing a pipeline, reducing the total execution effort according to the previously executed tasks, this feature is not available in Apache Airflow. In fact, in Airflow pipelines are always executed from beginning to end, independently of their previous runs. This is in line with the observation that not every system allows the partial execution of a pipeline and the storing of its partial results. However, we mathematically simulated the partial execution of data pipelines in order to validate our model on the execution side.

Future works may focus on implementing our data exchange model on systems different than Apache Airflow. Such systems may be data pipeline frameworks allowing the deployment of a data pipeline in a distributed way in order to simulate the presence of a provider and a consumer, trying different deployment configurations. Such systems may also provide the possibility of partial executions of data pipelines to obtain experimental results on execution efforts.

6.6. Chapter Conclusions

In this chapter, we have provided a real-world implementation of a DP and data pipelines starting from it. We have deployed a simple DP offering data through a REST API, implemented in Spring, and data pipelines starting from its API endpoint with Apache Airflow. Finally, we have executed these data pipelines to gather experimental data on effort and friction. This implementation allowed us to apply our data exchange model in a real scenario to verify its validity.

The successful deployment of the data pipelines defined in this chapter and the results obtained with their execution validate our proposed model, showing how data friction

and effort can be estimated in a practical context. Moreover, the obtained results help us identify guidelines in configuring and deploying data pipelines to minimize data friction, enhancing data sharing and the value generated from it.

Although the implemented model suffers from the limitations imposed by the used frameworks, it offers a valuable overview of how the concepts introduced in this thesis can be applied to real-world use cases, offering practical guidance on data pipelines' deployment strategies that yield optimal outcomes.

7 | Conclusions and Future Developments

As data becomes increasingly central in today's economy, setting up data exchanges in a friction-aware way can make data sharing ecosystems flourish, maximizing the value generated through data processing. However, the proliferation of data sources and consumers poses a significant challenge, leading to higher levels of data friction when setting up new data exchanges. As the benefits of data sharing ecosystems become increasingly evident, this friction impedes the seamless exchange of data, slowing down advancements in the industry and the scientific community.

In this thesis, we have provided a model to structure a data exchange between a data provider and a data consumer, identifying the main components enabling the exchange and the phases of the setup process, from the first consumer request to the completion. On top of this model, we have provided a mathematical method to quantify data friction in a given data exchange, distinguishing between *implementation* friction, occurring during the data exchange setup, and *execution* friction, occurring during the execution of the data exchange pipeline.

We then used this model to provide a structured and friction-aware approach to set up data exchanges in a data mesh. As the data mesh paradigm shift advocates the decentralization of data ownership and management to reduce data friction and maximize value generation, currently the data mesh lacks practical guidance in setting up data exchanges. Our model provides a method to connect data consumers and their requests to data products available in the mesh and quantify the effort needed to enable the data exchange among the two actors. Our friction-aware approach can optimize the management of data pipelines in the mesh, reducing the time to obtain the requested data objects in a context where efficiency and agility are paramount.

We implemented a simple proof of concept to verify the validity of our hypotheses and our model, developing a basic data product and using Apache Airflow to deploy data pipelines starting from it. We then calculated friction and effort in these pipelines according to the

equations previously defined. The results suggest that our model can be effectively used to quantify data friction and effort in a data exchange. Moreover, our model can help identify the optimal configuration of a data pipeline when trying to minimize the effort to deploy it.

Although our model covers the fundamental aspects to be considered in data exchanges, we did not address a number of factors such as resource management, set up of the communication channel, graph-structured data pipelines, etc. Throughout this thesis, we explicitly identified those areas that were outside the scope of this study. Future works may better detail how these relate to and extend our model, in an effort to make it as generalized as possible. Moreover, further research applying the model to various real-world use cases, including data mesh-based ones, can help in gaining insights into data frictions in different data sharing scenarios. As technology advances and data ecosystems become increasingly sophisticated, ongoing refinement and validation of the model against emerging frameworks and tools will be imperative. This will ensure the sustained relevance and applicability of our approach in the ever-evolving landscape of data management.

To the best of our knowledge, this thesis is the first body of work to address data friction in a structured, mathematical approach. We believe that the proposed model can be effectively adapted to real-world use cases to approach data sharing in a friction-aware manner, identifying and promoting those best practices to reduce the efforts needed to set up new data exchanges, ultimately contributing to the efficient and seamless functioning of data ecosystems.

Bibliography

[1] J. L. Alcala. Adidas data mesh journey: Sharing data efficiently at scale, 2022. URL `https://medium.com/adidoescode/adidas-data-mesh-journey-sharing-data-efficiently-at-scale-c50ee671fbd7`. Accessed on September 5th, 2023.

[2] G. C. Alter and M. Vardigan. Addressing global data sharing challenges. *Journal of Empirical Research on Human Research Ethics*, 10(3):317–323, 2015.

[3] P. Ataei and A. Litchfield. The state of big data reference architectures: A systematic literature review. *IEEE Access*, 2022.

[4] J. Bates. The politics of data friction. *Journal of Documentation*, 74(2):412–429, 2018.

[5] J. Bode, N. Kühl, D. Kreuzberger, and S. Hirschl. Data mesh: Motivational factors, challenges, and best practices. *arXiv preprint arXiv:2302.01713*, 2023.

[6] M. Bonde, C. Bossen, and P. Danholt. Data-work and friction: Investigating the practices of repurposing healthcare data. *Health informatics journal*, 25(3):558–566, 2019.

[7] C. L. Borgman. The conundrum of sharing research data. *Journal of the American Society for Information Science and Technology*, 63(6):1059–1078, 2012.

[8] C. L. Borgman. *Big Data, Little Data, No Data: Scholarship in the Networked World*. MIT Press, 2015.

[9] C. L. Borgman, J. C. Wallis, and M. S. Mayernik. Who's got the data? interdependencies in science and technology collaborations. *Computer Supported Cooperative Work (CSCW)*, 21:485–523, 2012.

[10] V. K. Butte and S. Butte. Enterprise data strategy: A decentralized data mesh approach. In *2022 International Conference on Data Analytics for Business and Industry (ICDABI)*, pages 62–66. IEEE, 2022.

[11] D. Carr. Giving viewers what they want, 2013. URL `https://www.nytimes.com/2013/02/25/business/media/for-house-of-cards-using-big-data-to-guarantee-its-popularity.html`. Accessed on September 3rd, 2023.

[12] P. Chen, J. Yang, A. Beheshti, and J. Su. Towards data economy: are products and marketplaces ready. In *Proceedings of the 48th International Conference on Very Large Data Bases (VLDB)*, pages 1–4, 2022.

[13] J. Cunningham. Netflix data mesh: Composable data processing, 2020. URL `https://www.youtube.com/watch?v=TO_IiN06jJ4`. Accessed on September 5th, 2023.

[14] Z. Dehghani. How to move beyond a monolithic data lake to a distributed data mesh. *Martin Fowler's Blog*, 2019.

[15] Z. Dehghani. Data mesh principles and logical architecture. *Martin Fowler's Blog*, 2020.

[16] Z. Dehghani. *Data Mesh*. Marcombo, 2022.

[17] A. Dolhopolov, A. Castelltort, and A. Laurent. Exploring the benefits of blockchain-powered metadata catalogs in data mesh architecture. 2023.

[18] P. N. Edwards. *A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming*. MIT Press, 2010.

[19] P. N. Edwards, M. S. Mayernik, A. L. Batcheller, G. C. Bowker, and C. L. Borgman. Science friction: Data, metadata, and collaboration. *Social studies of science*, 41(5): 667–690, 2011.

[20] M. Falconi and P. Plebani. Adopting data mesh principles to boost data sharing for clinical trials. In *2023 IEEE International Conference on Digital Health (ICDH)*, pages 298–306. IEEE, 2023.

[21] T. A. S. Foundation. Apache nifi documentation, 2023. URL `https://nifi.apache.org/docs.html`. Accessed on September 11th, 2023.

[22] J. Gilbertson and A. Salzber. Introducing uber movement, 2017. URL `https://www.uber.com/newsroom/introducing-uber-movement-2/`. Accessed on September 3rd, 2023.

[23] A. Goedegebuure, I. Kumara, S. Driessen, D. Di Nucci, G. Monsieur, W.-j. v. d. Heuvel, and D. A. Tamburri. Data mesh: a systematic gray literature review. *arXiv preprint arXiv:2304.01062*, 2023.

[24] M. R. Hasan and C. Legner. Data product canvas: A visual inquiry tool supporting data product design. In *International Conference on Design Science Research in Information Systems and Technology*, pages 191–205. Springer, 2023.

[25] M. R. Hasan and C. Legner. Understanding data products: Motivations, definition and categories. 2023.

[26] M. T. R. I. in association with Databricks. Building a high-performance data and ai organization, 2021. URL `https://www.technologyreview.com/2021/04/15/1022754/building-a-high-performance-data-and-ai-organization/`. Accessed on September 4th, 2023.

[27] D. Joshi, S. Pratik, and M. P. Rao. Data governance in data mesh infrastructures: the saxo bank case study. 2021.

[28] Y. Kim and J. M. Stanton. Institutional and individual influences on scientists' data sharing practices. *Journal of Computational Science Education*, 3(1):47–56, 2012.

[29] C. A. Lee, R. B. Bohn, and M. Michel. The nist cloud federation reference architecture 5. *NIST Special Publication*, 500:332, 2020.

[30] K. Lee and I. N. Sener. Strava metro data for bicycle monitoring: a literature review. *Transport reviews*, 41(1):27–47, 2021.

[31] S. Leonelli, N. Smirnoff, J. Moore, C. Cook, and R. Bastow. Making open data work for plant scientists. *Journal of Experimental Botany*, 64(14):4109–4117, 2013.

[32] H. Leung and Z. Fan. Software cost estimation. In *Handbook of Software Engineering and Knowledge Engineering: Volume II: Emerging Technologies*, pages 307–324. World Scientific, 2002.

[33] A. Loukiala, J.-P. Joutsenlahti, M. Raatikainen, T. Mikkonen, and T. Lehtonen. Migrating from a centralized data warehouse to a decentralized data platform architecture. In *International Conference on Product-Focused Software Process Improvement*, pages 36–48. Springer, 2021.

[34] M. Loukides. *The evolution of data products*. " O'Reilly Media, Inc.", 2011.

[35] I. Machado, C. Costa, and M. Y. Santos. Data-driven information systems: the data mesh paradigm shift. 2021.

[36] I. A. Machado, C. Costa, and M. Y. Santos. Data mesh: concepts and principles of a paradigm shift in data architectures. *Procedia Computer Science*, 196:263–271, 2022.

[37] J. Majchrzak, S. Balnojan, and M. Siwiak. *Data Mesh in Action*. Simon and Schuster, 2023.

[38] P. Murray-Rust. Open data in science. *Nature Precedings*, pages 1–1, 2008.

[39] B. T. O'Neill. Failure rates for analytics, ai, and big data projects = 85yikes!, 2019. URL `https://designingforanalytics.com/resources/failure-rates-for-analytics-bi-iot-and-big-data-projects-85-yikes/`. Accessed on September 4th, 2023.

[40] A. Osterwalder and Y. Pigneur. *Business model generation: a handbook for visionaries, game changers, and challengers*, volume 1. John Wiley & Sons, 2010.

[41] N. J. Podlesny, A. V. Kayem, and C. Meinel. Cok: A survey of privacy challenges in relation to data meshes. In *International Conference on Database and Expert Systems Applications*, pages 85–102. Springer, 2022.

[42] J. A. Praena, N. Verdian, and O. Torres Fernandez. Data mesh at glovo, 2022. URL `https://www.thoughtworks.com/en-ca/insights/blog/data-engineering/data-mesh-at-glovo`. Accessed on September 5th, 2023.

[43] A. Raj, J. Bosch, H. H. Olsson, and T. J. Wang. Modelling data pipelines. In *2020 46th Euromicro conference on software engineering and advanced applications (SEAA)*, pages 13–20. IEEE, 2020.

[44] P. Rankin. Roche diagnostics data mesh and data vault journey, 2022. URL `https://www.scalefree.com/wp-content/uploads/2022/09/Roche_DDL22.pdf`. Accessed on September 5th, 2023.

[45] I. Robinson. Consumer-driven contracts: A service evolution pattern. *Martin Fowler's Blog*, 2006.

[46] M. Schultze and A. Wider. Data mesh in practice: How europe's leading online platform for fashion goes beyond the data lake, 2020. URL `https://www.youtube.com/watch?v=eiUhV56uVUc`. Accessed on September 5th, 2023.

[47] R. Shaurya, J. Tang, P. Lanners, and S. Kern. Bmw cloud data hub: A reference implementation of the modern data architecture on aws, 2022. URL `https://aws.amazon.com/it/blogs/industries/bmw-cloud-data-hub-a-reference-implementation-of-the-modern-data-architecture-on-aw` Accessed on September 5th, 2023.

[48] C. Tenopir, E. D. Dalton, S. Allard, M. Frame, I. Pjesivac, B. Birch, D. Pollock, and

K. Dorsett. Changes in data sharing and data reuse practices and perceptions among scientists worldwide. *PloS one*, 10(8):e0134826, 2015.

[49] K. Vestues, G. K. Hanssen, M. Mikalsen, T. A. Buan, and K. Conboy. Agile data management in nav: a case study. In *International Conference on Agile Software Development*, pages 220–235. Springer International Publishing Cham, 2022.

[50] J. Walonoski, M. Kramer, J. Nichols, A. Quina, C. Moesel, D. Hall, C. Duffett, K. Dube, T. Gallagher, and S. McLachlan. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. *Journal of the American Medical Informatics Association*, 25(3): 230–238, 2018.

[51] J. Walonoski, S. Klaus, E. Granger, D. Hall, A. Gregorowicz, G. Neyarapally, A. Watson, and J. Eastman. Synthea™ novel coronavirus (covid-19) model and synthetic data set. *Intelligence-based medicine*, 1:100007, 2020.

[52] R. Y. Wang, Y. W. Lee, L. L. Pipino, and D. M. Strong. Manage your information as a product. *MIT Sloan Management Review*, 39(4):95, 1998.

[53] A. Wider, S. Verma, and A. Akhtar. Decentralized data governance as part of a data mesh platform: Concepts and approaches. *arXiv preprint arXiv:2307.02357*, 2023.

[54] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

List of Figures

List of Tables

Acknowledgements

Grazie innanzitutto al mio relatore, il Prof. Pierluigi Plebani, e al mio correlatore, il Dott. Matteo Falconi, per la loro disponibilità e per il prezioso contributo alla realizzazione di questa tesi.

Grazie alla mia famiglia, a mia mamma Roberta, mio padre Matteo e mia nonna Angela, per aver creduto in me e per avermi sempre sostenuto in questo percorso.

Grazie ai miei amici di Genova, e quindi a Carlo Andrea, Lucia, Lorenzo (B.), Lorenzo (D.), Filippo, Davide, Andrea (C.), Andrea (T.), Ramiro, e l'elenco va avanti, per esserci sempre stati. Grazie per tutte le esperienze vissute insieme: sono fortunato ad essere cresciuto al vostro fianco e a poter continuare a farlo.

Grazie ai miei amici di Milano, e quindi a Tommaso, Mauro, Giulia, Alessandra, Valeria, e l'elenco va avanti, per avermi fatto sentire a casa lontano (quasi due ore di macchina!) da casa. Grazie per le innumerevoli cene, le serate, le ore su divani vari, le domeniche sportive e chi più ne ha più ne metta.

Grazie ai miei amici di Barcellona, e quindi Leonardo, Alessia, Andre, Jorre, e l'elenco va avanti, per un'esperienza unica di crescita e di vita (e di svago, e di Razzmatazz, e di tequile).

Grazie a Sofia per il suo sostegno incondizionato. Grazie per esserci sempre e per spingermi a essere la versione migliore di me ogni giorno.

Infine, grazie a chi non ho ringraziato in questi brevi ringraziamenti. È difficile racchiudere in poche righe le esperienze e le persone che mi hanno accompagnato in questi cinque anni, ma sono grato a loro per averli resi indimenticabili. Citando un saggio, "grazie a chi c'è sempre stato, a chi c'è stato ma non da sempre, a chi ci sarà, a chi vorrebbe esserci ma non c'è" e poi non mi ricordo come prosegue.