POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Data Friction in Data Sharing: a Physics Inspired Model

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Giacomo Lombardo

Advisor: Prof. Pierluigi Plebani

Co-advisor: Matteo Falconi

Academic year: 2022-2023

1. Introduction

In the Big Data era, the increasing volumes of data data represent a crucial asset for organizations. Data-driven organizations can gain a competitive advantage by extracting value from data processing. In this context, data sharing has become a pivotal factor in driving progress, especially in those areas where data availability is low and heavily constrained by regulatory frameworks.

Despite the perception of data seamlessly flowing through devices, databases, and servers, data faces significant barriers and challenges opposing its natural movement. Coined by Paul Edwards [4], the term *Data Friction* refers to those efforts required to store, move, and access data. In today's scenario, given the heterogeneity of data sources and formats, data sharing ecosystems deal with increasing amounts of data friction. Traditional data architectures such as data warehouses and data lakes fail to scale along the proliferation of data sources and consumers as well as to fully exploit data-driven value generation.

In this scenario, a new paradigm in data architecture emerges: the *Data Mesh*. The data mesh is a socio-technical concept based on decentralized data ownership, allowing organizations to manage data at scale [3]. In the data mesh, centralized data repositories are reorganized into data products following domain-driven design principles. By decentralizing data management, data teams can work on specialized data products, ensuring higher quality and governance standards.

While the data mesh paradigm offers a novel approach to data management, it lacks a structured approach to set up data exchanges between data products. Moreover, at the state of the art, there is no model to define and quantify 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 to maximize value generation.

This thesis, which contributes to the European TEADAL research project[1], introduces a model to evaluate data friction in data exchanges, defining their main components and structuring their setup process. This model is then applied to the data mesh paradigm, extending its current implementation and addressing its shortcomings in data sharing. The model is validated through a proof of concept implementation, con-

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

cluding with observations and future research directions.

2. Related Work

2.1. Data Friction

The term *data friction* refers to "the costs in time, energy, and attention required to collect, check, store, move, receive, and access data" [4]. As in physics, data friction occurs between two data interfaces (e.g. from Edge to Cloud).
According to scientific literature, data sharing in scientific research is subject to high friction levels. In scientific disciplines, data sharing is enabled by a complex interconnected network of technological infrastructures, social and cultural relationships, and institutional support [1]. Neglecting these factors and their interaction creates data friction, hindering data flow. While fields such as astronomy and genomics have adopted data sharing and open data practices, others, such as healthcare, rarely reuse and share collected data for other experiments.
From a technology perspective, the main driver of data friction is the lack of a proper data sharing infrastructure and data management practices [1]. Data movement is restricted by the lack of licenses and standards and poor quality (or absence of) metadata.
The socio-cultural context also contributes to data friction. Especially in multidisciplinary settings, different research goals and data management practices can lead to friction in collecting and using data [2]. Despite acknowledging the advantages of data sharing, researchers frequently refrain from sharing their data or restrict data access [5].

2.2. Data Mesh

Traditional centralized data architectures are subject to high data friction in using and sharing data. To address these challenges, Dehghani proposes the Data Mesh [3] as a new paradigm for organizations to build their data architectures. The Data Mesh paradigm argues that reorganizing data according to business domains and decentralizing data ownership, moving data closer to sources and consumers, can reduce the frictions hampering value generation in centralized data architectures.
The data product (DP) is the architectural quantum of the data mesh, hence the smallest architectural unit that is independently deployable. At its core, a data mesh is a network of connected DPs. A DP is composed of three structural elements [3]:

- 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 governance;
- Platform dependencies: infrastructural components allowing creation, deployment, and management of the DP.

A DP has four types of ports: *input* ports, *output* ports, *control* ports, and *discoverability and observability* ports. The first two types enable the interfacing between DPs, allowing them to consume data and to serve it to the mesh. Control ports enable governance operations such as policy enforcement. Finally, discoverability and observability ports allow DPs to be available on the mesh and provide information about them. These ports are provided with unique URIs to be accessed through common APIs such as REST or GraphQL.

3. Data Sharing in Data Mesh

DPs input and output ports allow them to communicate with other DPs on the mesh. Such ports must be interoperable to enable data exchanges and sharing on the data mesh. That is, given a provider and a consumer on the mesh, the consumer must be able to ingest the output served by the provider. A DP's output ports are described on its Data Product Manifest (DPM): a document describing the DP, its configuration, and its state. Via the DPM, a DP specifies, for each output port, its URI and the characteristics of the served data object in an OpenAPI-like fashion. Looking at the DPM, a consumer should understand how to ingest the served data.
As the goal of the data mesh is to deliver value through data processing, we argue that the provider has to adapt to the consumer's request. In line with our observation, Wider et al. [6] introduce consumer contracts and output ports SLOs in the data mesh domain model. This contract-based approach helps manage the

evolution of DPs in a consumer-oriented way, ensuring interoperability throughout the entire DP lifecycle.
However, the number of interfaces between providers and consumers significantly increases as the mesh grows, and such an approach may slow down the deployment and evolution of DPs. Currently, the data mesh does not provide a structured method to set up data exchanges. Moreover, it should be possible to quantify the effort needed to enable the interfacing between provider and consumer. In this way, data sharing in the mesh can be approached in a friction-aware way to manage and ease the evolution of DPs.

4. Modeling Data Friction in Data Exchanges

To address the challenges of setting up data exchanges in the data mesh and reduce the data friction in such a scenario, we first need a model to structure a data exchange and define its inherent data friction.

4.1. Data Exchange Components

We can identify three main components in our model:

- Data Provider: the entity that serves the data object;
- Data Consumer: the entity that receives the data object;
- Data Object: the payload transmitted from provider to consumer.

The goal of the data exchange is the successful transmission of the data object d from provider to consumer. A data object can be transformed through a capability, represented by a function c that receives a data object d_{in} as input and returns a transformed data object $d_{out} = c(d_{in})$. For instance, a transformation could be translating the data object from .csv to .json format or anonymizing a dataset.
Likely, a data object has to be transformed multiple times in a data exchange. A data pipeline p is an ordered set of capabilities:

$$p = \{c_1, \ldots, c_n\}$$

From a functional standpoint, a pipeline can be represented as a function p with respect to the data object d. In fact, by leveraging the composition operator $\circ$, the result of the transformations applied to d is:

$$d_{out} = c_n \circ \ldots \circ c_1(d) = p(d)$$

Assuming we are considering a single provider offering a data object, different consumers could require different transformations, thus defining different pipelines. Given two pipelines p_α and p_β, related to different consumers α and β, if $p_\alpha \cap p_\beta \neq 0$ then the two pipelines share a set of capabilities. This shared set will be implemented only once and executed when they are required for one of the two exchanges.

4.2. Data Pipeline Deployment

Given a data pipeline's capability $c_i \in p$, it can be deployed either on the provider or on the consumer side. It is possible that a capability can be deployed only on one side (e.g, an anonymization can be performed only on the provider side). Given C the universe of all the capabilities, the sets $\overleftarrow{C} \subseteq C$ and $\overrightarrow{C} \subseteq C$ identify, respectively, the capabilities that can be deployed only on the provider and on the consumer side. If a pipeline has a capability $c_x \in \overleftarrow{C}$, then all the capabilities $c_i \in p, i \leq x$ must be deployed on the provider side. The same goes for the other side of the pipeline.
With this in mind, a pipeline p can be represented as $p = \overleftarrow{p}_x + \overrightarrow{p}_{x+1}$, where p is divided in two sub-pipelines, one comprising all the capabilities up to c_x deployed by the provider ($\overleftarrow{p}_x$) and one comprising the remaining capabilities deployed by the consumer ($\overrightarrow{p}_{x+1}$).
Given a pipeline $p = \overleftarrow{p}_x + \overrightarrow{p}_{x+1}$, the data object is transmitted from provider to consumer between the capabilities c_x and c_{x+1}. The transmission of the data object is represented as an additional capability c_t in the deployed data pipeline. Depending on the last capability implemented on the provider side, a pipeline can have different *deployment configurations*:

$$\widehat{p}_x = \overleftarrow{p}_x + c_t + \overrightarrow{p}_{x+1}$$

Given a pipeline p and the sets $\overleftarrow{C}$ and $\overrightarrow{C}$, there are $1 + |p| - |p \cap \overleftarrow{C}| - |p \cap \overrightarrow{C}|$ possible deployment configurations for p.

4.3. Effort

Once agreed on a pipeline p and a deployment configuration $\widehat{p}_x$, the provider implements the capabilities that have not been previously implemented. The implementation of a capability c requires an *implementation effort* $E_I(c)$. The implementation effort can be obtained by applying cost estimation models such as COCOMO.
Once implemented the missing capabilities, the pipeline is deployed and has to be executed. The execution of a capability c requires an *execution effort* $E_I(c)$. The execution effort depends on the inherent difficulty of the capability, identified by the unitary effort $e(c)$, and on the size of the data object N_d:

$$E_E(c) = e(c)N_d$$

The execution effort of a capability can be estimated experimentally.

4.4. Friction

In a pipeline, the more capabilities to implement the more difficult the setup of the data exchange. Given a pipeline p and the set of previously implemented capabilities C_P, the *implementation friction coefficient* can be calculated as

$$\mu_I = \frac{\sum_{c \in p \setminus C_P} E_I(c)}{\sum_{c \in p} E_I(c)}, \quad 0 \leq \mu_I \leq 1$$

To overcome the implementation friction, the provider will have to provide a total implementation effort $E_{I,P}$ such that

$$E_{I,P} \geq \mu_I \sum_{c \in p} E_I(c)$$

Similarly, the more capabilities to execute the more difficult the completion of the data exchange. Given a deployment configuration $\widehat{p}_x$, the execution friction coefficient can be calculated as

$$\mu_E = \frac{\sum_{c \in \widehat{p}_x \setminus \widehat{p}'_x} E_E(c)}{\sum_{c \in \widehat{p}_x} E_E(c)}, \quad 0 \leq \mu_E \leq 1$$

where $\widehat{p}'_x$ is the portion of $\widehat{p}_x$ that has already been executed. Note that the partial execution of a pipeline is possible only in some systems: otherwise, the entire pipeline is executed and $\mu_E = 1$ by definition.

To overcome the execution friction, provider and consumer will have to provide a total execution effort E_E such that

$$E_E \geq \mu_E \sum_{c \in \widehat{p}_x} E_E(c)$$

5. Data Friction in Data Mesh

We can now apply the model defined in Section 4 to data exchanges in the data mesh. That is, data providers (DPs) and consumers (either DPs or final users) should be able to set up structured data exchanges, following a friction-aware approach to enhance value generation. By considering data friction, the time and effort required to set up a data exchange can be reduced, resulting in easier data sharing in the mesh.
In the data mesh, a DP can be a both a provider and a consumer. In this study, we model data exchanges between a DP acting as data provider and a data consumers that can either be DPs or final users. The interfacing between provider and consumer is enabled by a data pipeline p composed of *tasks*, each representing a capability. The pipeline p transforms the data object d served by one of the provider's output ports into the various data objects requested by the consumers.
The pipeline is defined by the provider's *data steward*, according to the separation in different domains of the data mesh. The data object must belong to the provider's domain throughout the entire pipeline. The consumer can put constraints on the pipeline definition by requesting the deployment of a capability on a specific location. For instance, a consumer may require that a capability is deployed on a certain resource to monitor the task's behavior.
All the actors must adopt a uniform perspective on implementation and execution effort in order to effectively calculate the inherent friction of a data exchange. That is, provider and consumer must use the same metrics when calculating the implementation/execution effort of a given task. The definition of these metrics must be defined at governance level in order to ensure a uniformed view throughout the entire data mesh.
We argue that following this structured approach for data exchanges in a data mesh can be beneficial for multiple reasons:

- By considering data friction, the efforts in setting up data exchanges can be minimized;
- The friction-aware evolution of DPs can maximize the value generated by data sharing;
- The data mesh can be extended to federated deployments by allowing inter-organizational data exchanges.

6. Implementation

We provide a proof of concept implementation to validate our model in a real-world scenario. More specifically, we deploy a simple DP serving a list of hospital patients in *.csv* format. We then deploy using Apache Airflow three data pipelines starting from the DP's output.

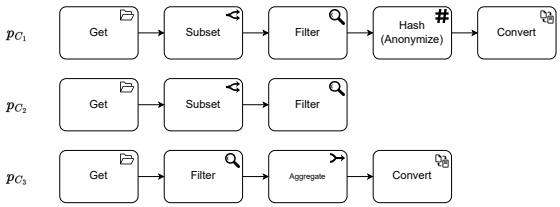

Figure 1: Implemented pipelines.

To estimate the effort and friction of each pipeline, we first define the metrics to calculate a task's effort. The implementation effort is calculated as the lines of code (LoC) and the execution effort as the execution time of the task.

6.1. Pipeline I

The pipeline p_{C_1} consists of the following tasks: retrieve the data from the data source, select a subset of the data, filter the data based on the ZIP code of the patients, anonymize the data, and convert the data from *.csv* to *.json*.
The *anonymize* task can be deployed only on the provider side, thus there are only two possible deployment configurations: $\widehat{p}_{C_1,4}$ and $\widehat{p}_{C_1,5}$.
Given that the implementation of p_{C_1} begins at t_0, all the tasks have to be implemented and $\mu_I = 1$. The total implementation effort of the pipeline is $E_{I,P} = 56$ (Table 1). Similarly, all the tasks have to be executed and $\mu_E = 1$.
Similarly, all the tasks have to be executed and $\mu_E = 1$. The execution effort of the tasks in p_{C_1} is reported in Table 2.
The only factor to consider when choosing the optimal deployment configuration is the effort

Task	$E_I(c)$
`Get`	16
`Subset`	8
`Filter`	8
`Anonymize`	12
`Convert`	12
$E_{I,P}$	**56**

Table 1: Implementation effort $E_I(c)$ of pipeline p_{C_1}

Task	$E_E(task)$
`Get`	1.12
`Subset`	0.39
`Filter`	0.28
`Anonymize`	0.22
`Convert`	0.27
E_E	**2.29**

Table 2: Execution effort $E_E(c)$ of pipeline p_{C_1}

required by c_t (Table 3): $\widehat{p}_{C_1,4}$ is to be preferred since the total execution effort is lower.

$\widehat{p}$	**Task before** c_t	N_d	$E_E(c_t)$
$\widehat{p}_{C_1,4}$	`Anonymize`	23 KB	0.23
$\widehat{p}_{C_1,5}$	`Convert`	54 KB	0.54

Table 3: Execution effort for c_t in p_{C_1}

6.2. Pipeline II

The pipeline p_{C_1} consists of the following tasks: retrieve the data from the data source, select a subset of the data, and filter the data based on the ZIP code of the patients. The *get* task can be deployed only on the provider side: the possible deployment configurations are $\widehat{p}_{C_2,1}$, $\widehat{p}_{C_2,2}$ and $\widehat{p}_{C_2,3}$.
All the tasks in p_{C_2} have already been implemented and $\mu_I = 0$. Regarding the execution, not all tasks have to be executed depending on the configuration. As evident in Table 4, $\widehat{p}_{C_2,3}$ requires the lower execution effort among the possible deployment configurations.

$\widehat{p}$	$E_{E,P}$	$E_E(c_t)$	E_{E,C_2}	E_E	μ_E
$\widehat{p}_{C_2,1}$	0 (1.51)	3.90	0.89	4.79	0.76
$\widehat{p}_{C_2,2}$	0 (2.14)	0.83	0.26	1.08	0.34
$\widehat{p}_{C_2,3}$	0 (2.40)	0.02	0	0.02	0.01

Table 4: Execution effort and friction of the deployment configurations of p_{C_2}.

6.3. Pipeline III

Finally, the pipeline p_{C_3} consists of the following tasks: retrieve the data, filter the data based on the ZIP code of the patients, aggregate the data based on the ZIP code, and convert the data from *.csv* format to *.xml*.
Only two tasks in p_{C_3} have to be implemented and the implementation friction is $\mu_I = 0.49$ (Table 5).

Capability	$E_I(c)$
`Get`	0 (16)
`Filter`	0 (8)
`Aggregate`	8
`Convert`	15
$E_{I,P}$	**23 (47)**

Table 5: Implementation effort $E_I(c)$ of pipeline p_{C_3}

The possible deployment configurations are $\widehat{p}_{C_3,1}$, $\widehat{p}_{C_3,2}$, $\widehat{p}_{C_3,3}$ and $\widehat{p}_{C_3,4}$. Depending on the chosen configuration, the execution effort and friction changes (Table 6).

$\widehat{p}$	$E_{E,P}$	$E_E(c_t)$	E_{E,C_3}	E_E	μ_E
$\widehat{p}_{C_3,1}$	0 (1.06)	3.90	0.76	4.66	0.81
$\widehat{p}_{C_3,2}$	0.32 (1.38)	0.10	0.44	0.86	0.45
$\widehat{p}_{C_3,3}$	0.56 (1.62)	≈0	0.21	0.76	0.42
$\widehat{p}_{C_3,4}$	0.76 (1.82)	≈0	0	0.76	0.42

Table 6: Execution effort and friction of the deployment configurations of p_{C_3}.

According to Table 6, the optimal deployment configurations are $\widehat{p}_{C_3,3}$ and $\widehat{p}_{C_3,4}$ as they yield the best results in terms of execution friction and effort.

7. Conclusions and Future Works

In this thesis, we defined a model to structure a data exchange between a data provider and a data consumer, providing a method to quantify the data friction of the exchange.
We then use such a model to provide a structured and friction-aware approach in setting up exchanges in the data mesh. We argue that our friction-aware approach can effectively optimize data sharing in the mesh, reducing the time needed to obtain the requested data objects.
The results obtained from the proof of concept implementation suggest that our model can be effectively used to quantify data friction and effort in a data exchange, identifying the optimal configuration of a data pipeline.
To the best of our knowledge, this is the first body of work addressing data friction with a structured, mathematical approach. We believe the proposed model can be adapted to real-world use cases to approach data sharing in a friction-aware way, ultimately contributing to the efficient and seamless functioning of data ecosystems.
Future works may delve deeper into the definition of the components of our model and further explore real-world applications, gaining insights into data friction in different data sharing ecosystems.

8. Acknowledgements

Special thanks go to Professor Pierluigi Plebani, Doctor Matteo Falconi, and everyone who helped with the writing of this thesis, directly or indirectly.

References

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

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

[3] Zhamak Dehghani. *Data Mesh*. Marcombo, 2022.

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

[5] Carol Tenopir et al. Changes in data sharing and data reuse practices and perceptions among scientists worldwide. *PloS one*, 10(8):e0134826, 2015.

[6] Arif Wider, Sumedha Verma, and Atif Akhtar. Decentralized data governance as part of a data mesh platform: Concepts and approaches. *arXiv preprint arXiv:2307.02357*, 2023.