# Data Friction: Physics-Inspired Metaphor to Evaluate the Technical Difficulties in Trustworthy Data Sharing

Matteo Falconi[1], Giacomo Lombardo[1], Pierluigi Plebani[1], and Sebastian Werner[2]

[1] Politecnico di Milano, Milan, Italy
{matteo.falconi, pierluigi.plebani}@polimi.it
giacomo.lombardo@mail.polimi.it
[2] Information Systems Engineering, Technische Universität Berlin, Germany
werner@tu-berlin.de

**Abstract.** Data sharing between organizations is becoming an ever-increasing necessity. Data sharing allows organizations to improve business processes that depend on what happens in other organizations, just as having data from other organizations can enrich data analysis models. However, even though data is seen as the new oil, it does not move like oil. There are several technical and organizational factors that make data sharing difficult.
In this work, inspired by the definition of friction in physics, we want to provide a first friction model that is able to capture the elements that hinder data sharing. The proposed model hypothesizes that through the adoption of data mesh in conjunction with a service-oriented sharing approach, we can utilize this model to reduce and reuse the effort for sharing data.

**Keywords:** Federated Data Sharing, Modeling Data Friction

## 1 Introduction

Although companies recognize data as an important asset, and it is proven that sharing this resource can improve the business, data sharing is often hindered by technical and organizational aspects [9]. In fact, every organization would like to have access to other organizations' data, there is a certain distrust in sharing their own data. Among the main problems, there is the fear of losing control over critical information and the costs of making the sharing process compatible with current regulations (for example, GDPR for the personal information of European citizens). From a technical point of view, data sharing implies the definition of data governance policies that make access to data safe, limited only to the data to which a potential consumer has the right to access, and efficient in both construction and maintenance of the technological elements necessary for data sharing. There is therefore a growing need to have solutions capable of

guaranteeing sovereignty over one's data, i.e., full control over them even when they are shared with other organisations [2].

In this context, the adoption of data mesh [4] as an approach for managing data for analytics purposes is proving useful for efficiently managing data within an organization [10]. However, when you intend to expand data sharing to other organizations you need to have a data sharing process in place that ensures controlled and observable access to the data. Control refers to the fact that sharing is limited to only the data that the specific consumer has the right to access. Observability refers to the need to know what happens when data leaves the organization and is no longer directly managed. The combination of these two elements enables the creation of trustworthy data sharing, and the effort required to create them somehow opposes data sharing. Taking inspiration from what happens in physics, we can say that there is a force that opposes the movement of data and which can be equated to the concept of friction. Then, the following research question is defined: *How can we evaluate data frictions in data sharing in a fair and manageable way?*

To answer this question, this work starts from the assumption that the organizations involved in data sharing belong to the same federation. This allows us to guarantee that some rules are shared and adopted by all participants. On this basis, a data sharing process is proposed in a federated context that allows the definition of friction. As in physics, this friction is based on two components, static and dynamic, which are linked to the effort necessary to build the data sharing system, modeled as data pipelines, and to perform data sharing also including the effects of the network and observability mechanisms.

The remainder of this paper is organized as follows: First, we present related work in Section 2, followed by the introduction of the problem of federated data meshes (Section 3). Then, we present the trustworthy data friction model in Section 4, before presenting a preliminary validation in Section 5 and concluding in Section 6.

## 2 Related work

Coined by Paul Edwards [6], 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* occurs at the interface of two "surfaces," two points where data moves. Data Friction restricts and impedes the natural movement of data and requires cost and effort to overcome and it emerges when data, primarily due to low levels of data governance and curation, is not properly collected and managed.

From a technology perspective, the main driver of data friction is the lack of a proper data-sharing infrastructure and data management practices [1]. It has been observed [14] that the technological advances that have facilitated data collection have also hindered the sharing and reuse of data in the scientific community due to a lack of licensing and standards. Unified models and platforms to support data sharing are often lacking, even within the same domain [12].

Focusing on the platforms, in centralized data architectures such as data lakes, oftentimes, the absence of data and metadata standards, as well as poor data curation, lead to high friction when sharing and using data [3]. To address these challenges, Dehghani proposes the data mesh paradigm [3, 4] 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.

Although data mesh can reduce the effort in sharing the data inside an organization, some problems remain when organizations want to share this data [8]. The goal of this paper is to deal with these problems by extending the idea of the *data product*, which is central in the data mesh approach, in a different setting inspired by the data spaces ecosystems that are characterizing the European arena [15]. In fact, while a data product is designed to be managed and consumed internally by an organization, the situation changes when it is assumed that the data product can be shared with external organizations. In this case, the adoption of a service oriented approach allows to offer data products as services. Exploiting the well established service orientation principles defined by Erl [7], a data product is designed to be consumed by a class of consumers which are interested in different aspects of the data that is managed in that product. Although this could facilitate the initial engagement between the data provider and consumer, unlike the usual service provisioning, the data that can be shared must be properly tailored with respect to the real needs and rights that the consumer has. In this sense, the added effort of exposing the right data to the right consumer affects the data friction as it is defined in this paper.

## 3 Federated Data sharing model

Based on the work of Jussen et al. [11], data sharing is defined as the domain-independent process of giving third parties access to data sets of others within the framework of the agreement between the involved parties. In our setting, we assume that two parties are involved: a data provider that wants to share the data and a data consumer that is interested in these data. It might happen that the data provider shares the same data with different data consumers, thus, a specific data sharing process is required to manage the relationship between the data provider and each of the data consumers. Five main steps constitute this process (see left of Fig. 1):

- Dataset preparation: data are collected and organized by the data provider.
- Data sharing agreement: data consumer and data provider, according to a negotiation process, which is out of the scope of this paper, agree on which data are actually shared, as well as, the format (e.g., JSON, CSV) and the possible mandatory transformations (e.g., anonymization).

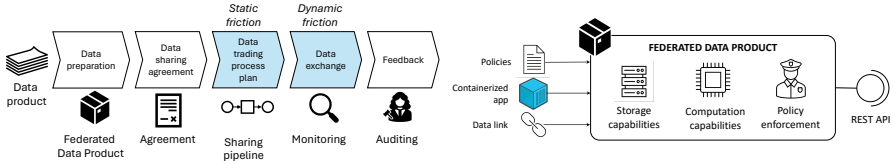


**Fig. 1.** Federated data sharing process (left) and FDP architecture (right).

- Planning of the data trading process: the requirements dictated by the agreement are translated into technical elements that ensure that only the agreed data are exchanged.
- Data exchange: here, the actual data movement occurs from the data provider to the data consumer.
- Feedback to the data provider and data consumer: assuming that the data exchange is partially or fully observable (e.g., through a monitoring system), data about how the data exchange occurred and to which extent the agreement has been respected are collected.

Based on the steps above, in this section, we introduce the approach adopted to implement a trustworthy data sharing process assuming that: (i) the data sharing occurs in a federated setting; (ii) data mesh approach is used as a foundation of the data sharing; (iii) data mesh is extended to support cross-organizational data sharing; (iv) blockchain is adopted as the mechanism to attest the correctness of the data exchange.

To better introduce the aspects related to the data sharing and to better motivate the proposed approach to improve the effectiveness of data sharing, we refer to a common scenario in the healthcare domain: hospitals that want to share data about their patients with other hospitals in the realm of a multi-centric clinical trial [5]. Conducting a clinical trial requires both properly anonymized individual data and already aggregated data from the two hospitals. The patient data will be used by the other hospital for further aggregation or by the medical center for research purposes. Data sharing is, therefore, a major component of a successful clinical trial. When the multi-centric clinical trial is set up a set of rules is identified and the participating hospitals must adhere with. This is a sort of federation. Possible rules could refer to how to collect data, who has the right to read the data, the format, and whether the data are anonymized or not. In particular, we assume that an hospital of this network is willing to share a dataset of its patients with the other hospitals. Each of these hospitals could be interested in all the patients, in a portion of them, in a data set organized in a specific format, or a subset of data obtained by aggregation.

***Data preparation:*** Starting from the successful use of the data mesh approach, the data preparation is related to generating the *data product* is well supported in organizations. However, the data mesh focuses mainly on data management within an organization without considering the possibility of sharing data externally. Here, we extend the concept of data mesh to a federated perspective. In

particular, our attention focuses on the extension of the data product model by proposing the so-called *Federated Data Product (FDP)* (see right of Fig. 1). In order to support the data governance policies, an FDP has the capability to run code, store data, and to enforce policies. Not all capabilities must be included if not needed by the governance policies.

An FDP is, therefore, an architectural element designed for sharing the data it contains. Inspired by the principles of service orientation [7], this element must also be loosely coupled and reusable. About the former principle, it is expected that the lifecycle of the FDP is as much as possible independent of its consumer. About the latter, the logic encapsulated by the service is associated with a context that is sufficiently agnostic to any one usage scenario so as to be considered reusable by different consumers.

Therefore, the design and creation of a FDP must be targeted to a consumer type and not by a particular consumer. In this way, the same FDP can be used for different consumers. At the same time, each consumer may have particular needs, or data governance policies may restrict the amount or the level of data aggregation depending on the type of consumer.

Referring to the example, an FDP could be the list of patients whose data has been collected in a clinical trial within a hospital. This FDP contains different data types, e.g., biographical, and clinical. This does not mean that all this data can be visible to everyone, but rather, visibility must be consumer-dependent. For example, a particular doctor from another hospital can have visibility into the entire data set as long as it is anonymized. Or, the ethics committee of another hospital that is studying the possibility of using this data for another clinical trial has access to a subset of the data, always subject to anonymization.

***Data sharing agreement:*** Here the producer and consumer must decide what part of the data product is allowed, needed, and wanted. Consumer identification, processing purposes, and data quality guarantees can be negotiated here.

Adapting to these needs can be seen as a series of transformations on the output of the FDP. The execution costs of these transformations could and should be divided between the consumer and producer. Hence, part of the agreement must clearly describe transformations and the resources both parties are willing to provide. Here, we should offer both parties the capability to judge the potential friction an exchange might cause and, thus, how much effort in terms of resources, implementation and execution capacity is needed for that exchange.

***Data trading process plan:*** The specific agreement between a provider and consumer can stipulate mandatory or desired transformations executed in a pipeline between the FDP port and the customer. During the data trading process, the specificities of this pipeline are defined (e.g., how and where to execute). Naturally, this concrete specification presents several Data Frictions, e.g., in the volume of data to transport across the network. Besides the execution plan, the data provider must be able to verify and observe the correct execution of the pipeline. These observations must be accessible to the provider and must be trustworthy.

Between the implicit decision on where to perform each part of the data sharing pipeline and the decision on when to move data across the network, observing that execution also becomes crucial explicit friction. Hence, the earlier the data is moved across the network, the more steps need to be observed to ensure that the consumer receives only the agreed data in an agreed format. On the other hand, the later data is moved, the more computing and storage resources the provider must use. The decision of when to move data is influenced by resource availability, projected data volume, technical capabilities, implementation capacity, and the reliability of approaches to processing data without leaking confidential information. We can address the first part by carefully modeling the friction between each decision point. For the second part, we must rely on best-effort approaches, i.e., combining blockchain properties with infrastructure and process observation approaches and modeling the cost of interacting with these technologies. This paper presents a concrete model to evaluate these frictions, which can be used to find a fair or acceptable distribution.

***Data exchange:*** Once the execution plan is defined, the pipeline and accompanying components for observation and potentially needed identity or policy enforcement are deployed and started. How this happens depends strongly on the used resources and implementations provided by both the consumer and producer. One can envision sharing access to Kubernetes namespaces across networks, sharing prepared VMs, or deploying serverless functions that react based on consumption events. Similarly, the transformation itself could be custom Python code or transformations defined in a workflow engine such as Kubeflow or Apache Spark. For this paper, the automatic management of the execution across the resources of two organizations is out of scope.

***Feedback:*** Once a data exchange is completed or has sufficiently been used, both the consumer and provider may want or need to review the interactions of the data exchange. This feedback of the exchange has several goals. It allows both parties to review data movement decisions, which may inform future exchanges. A more crucial goal is related to the partially personal or confidential data exchange; here, providers are obligated to report all processing and used procedures. Similarly, customers may want to validate that the data was provided truly from raw sources, e.g., sensor measurements.

These needs imply the transfer of observation data as evidence between the consumer and provider after a data exchange and the retention of such data for the legal reporting periods or alternatively means to prove correctness even without the original logging data. To that end, we can either enable contractual means to provide access to that data after the exchange or utilize public repositories to store relevant evidence. However, as this evidence can also become crucial in disputes, we must ensure that the access can not be revoked later and that publicly shared evidence is verifiable and self-contained. We envision an evidence-based infrastructure as part of the FDP connected to immutable distributed storage solutions, such as IPFS. Combined with distributed ledgers, it can function as a verifiable public repository.

| Symbol | Description | Symbol | Description |
|---|---|---|---|
| $d$ | Data object | $\overrightarrow{p}^k_{x+1}$ | Part of $p_k$ deployed by the consumer |
| $c$ | Capability | $\widehat{p_x}^k$ | Deployment configuration $x$ of $\widehat{p}^k$ |
| $C$ | Set of capabilities implemented by the provider | $E^I(c)$ | Implementation effort of $c$ |
| $C'$ | Set of capabilities already executed on the same data | $E^I_{\widehat{p_x}^k}$ | Implementation effort of $\widehat{p_x}^k$ |
| $c_t$ | Transmission capability | $E^E(c,d)$ | Execution effort of $c$ on $d$ |
| $c_o$ | Observability capability | $E^E_{\widehat{p_x}^k}$ | Execution effort of $\widehat{p_x}^k$ |
| $\overleftarrow{C} \subseteq C$ | Set of $c$ that must be deployed by the provider | $e(c)$ | Unitary execution effort of $c$ |
| $\overrightarrow{C} \subseteq C$ | Set of $c$ that must be deployed by the consumer | $sizeof(d)$ | Size of the data object $d$ |
| $p^k$ | Data pipeline between provider and consumer k | $\mu^I_{\widehat{p_x}^k}$ | Implementation friction of $\widehat{p_x}^k$ |
| $\overleftarrow{p}^k_x$ | Part of $p_k$ deployed by the provider | $\mu^E_{\widehat{p_x}^k}$ | Execution friction of $\widehat{p_x}^k$ |

**Table 1.** Reference table for the notation

## 4 Evaluating data friction

In physics, friction $F$ is the force that opposes the movement of an element when it scrapes a certain surface. Therefore, to move the element it is necessary to apply a force $F_{apply}$ that is greater than the force that friction generates. This is represented by the equation:

$$F_{apply} > F \quad \text{where} \quad F = \mu \cdot N$$

$N$ is the force that opposes the movement (usually associated with the gravity of an element) and $\mu$ is a coefficient that takes into account the material of the involved surfaces.

In case of data friction, naming $d_{fdp}$ the data exposed by an FDP and $d_{cons}$ the data that is requested by the consumer, the forces that oppose to the movement are related to the effort that is required to transform and observe the data from the original format and content as it is created by the data provider, to the format and content as required by the user. Like in physics, where there are static and dynamic friction, also in our case, we have two components of the data friction: (i) the *static friction* $F_S$ related to the effort $E^I$ required to *implement* the pipeline that performs the transformation required, and (ii) the *dynamic friction* $F_D$ related to the effort $E^E$ in *moving* data from the provider to the consumer:

$$F_S = \mu^I \cdot E^I \qquad F_D = \mu^E \cdot E^E$$

As discussed in the next paragraphs, the higher the reuse of transformation and observation capabilities, the lower the coefficient of static friction $\mu^I$. Similarly, the higher the amount of transformation applied to the same data, the lower the coefficient of the dynamic friction $\mu^E$. This is because of the adoption of a data mesh approach, where the same FDP can be offered to different consumers and several of them could require one or more common transformation capabilities in the related pipelines.

The computation of $F_S$ and $F_D$ can be used during the agreement phase to make (i) the data provider aware of the effort that should be put in place to

implement the pipeline, and (ii) the data consumer aware of the effort required to move and observe the data from the provider side.

This value can be used to enrich the description of an FDP and it can be included in the set of metadata used to classify the FDP in a data catalog.

As a guideline for the rest of this section, Table 1 lists all the elements used to introduce the proposed approach to quantify $F_S$ and $F_D$.

***Data transformation capability.*** A transformation is carried out by a *capability*, that is represented by a function $c$ that receives a data object $d_{in}$ as input and returns as output a data object $d_{out}$ that has undergone a certain transformation: $d_{out} = c(d_{in})$. For instance, a transformation could be converting the data object from *.csv* to *.json* format or anonymizing a data set.

The set of capabilities required for a data exchange must be first *implemented* by the data provider, and then *executed* on the data object $d_{in}$ to produce the $d_{out}$. The implementation phase consists of the realization of a software script or program that is able to perform the required transformation (e.g., using Spark or Airflow). As a basic assumption, the data provider is in charge of the implementation of the capabilities to ensure the proper control on the way in which the data is offered to avoid possible data leakage. In fact, the consumer has only the role of consuming the data produced by the set of transformations.

The execution phase, as the name suggests, consists of the execution of the implemented program on the data object, thus applying the transformation $c(d)$ on the data object. Each capability can be implemented only once, but they can be executed multiple times, e.g. if in the future another data consumer will request a data object in *.json* format, the hospital on focus in our example scenario will not need to implement again the script, but will only need to execute it on the new data object required for the exchange. On this basis, we define $C$ as the set of capabilities that a provider has already implemented. If a new capability will be required, this set will increase.

***Data pipeline.*** A data pipeline $p$ concerns an *ordered sequence* of data transformations supported by a related set of capabilities, thus:

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

The order is related to the execution of the capabilities, thus, $c_i$ will be executed before $c_j$, if $i < j$.[3] Referring to our case study, a possible pipeline could be: data filtering, anonymization, conversion to *.json* format.

Considering the set of capabilities already implemented by the provider $C$, therefore $p - C$ indicates the set of capabilities that are defined in the pipeline that are required to be implemented by the data provider.

From a functional standpoint, a pipeline can be also represented as a function with respect to the data object on which the pipeline operates. In fact,

[3] For the sake of simplicity, only sequential pipelines are considered. In the case of branches or loops, their execution traces can be considered as pipeline variants, but this is out of the scope for this work.

by leveraging the composition operator (represented with $\circ$), the result of the transformations applied to $d_{fdp}$ is:

$$d_{cons} = c_n \circ \ldots \circ c_1(d_{fdp}) = p(d_{fdp}).$$

Thus a pipeline is the tool that allows the data provider to ensure that the data offered through an FDP is exposed to the consumer as agreed in terms of format and content. For the sake of clarity, we indicate as $p^k$ the pipeline used to transform the data offered by an FDP according to the agreement reached with a given consumer $k$, thus $d_k = p^k(d_{fdp})$. As several consumers might be interested in the data offered by the same FDP, this notation makes it possible to distinguish the related pipelines.

Once the FDP is created, we assume that $C = \emptyset$ as no capabilities have been implemented at that time by the provider. Every time a new pipeline is requested due to the arrival of a new customer, this set is updated as follows:

$$C = C \cup (p^k \cap C)$$

In fact, $p^k \cap C$ indicates the capabilities required in the pipeline but not yet implemented. Assuming that they will be implemented by the provider, $C$ will include these new capabilities. As a result, $C$ evolves during the time and the capabilities required to be implemented for a given pipeline depend on the history of that FDP.

For instance, in our reference scenario, we can have an agreement with a medical research center that results in a pipeline $p^1 = \{$*filter, anonymize, convert*$\}$, while the pipeline required to expose data to another hospital is $p^2 = \{$*anonymize, convert, compress*$\}$. Thus, starting with $C = 0$, with the definition of $p^1$ and the related implementations, we have $C = \{$*filter, anonymize, convert*$\}$, Then, when $p^2$ comes into play, so that $p^2 \cap C = \{$*compress*$\}$ which will set $C = \{$*filter, anonymize, convert, compress*$\}$.

***Data pipeline deployment.*** While the implementation of a capability is under the responsibility of the provider, a peculiar aspect of the proposed approach concerns the possibility to deploy this capability on resources at provider or consumer side. The decision on where to deploy can sometimes be forced by the nature of the transformation (e.g., it is reasonable that an anonymization can be performed only at the provider side) or the availability of the resources.

More formally, being $C$ the set of capabilities implemented by the provider, $\overleftarrow{C} \subseteq C$ and $\overrightarrow{C} \subseteq C$, respectively, indicate the capabilities that must be deployed only at the data provider side and at the data consumer side. These constraints are set during the agreement phase, where the provider and the consumer consider their available resources and the nature of the transformations. Once the $\overleftarrow{C}$ and $\overrightarrow{C}$ are defined, considering a pipeline $p^k$, it might exist an index $x$ so that:

$$\forall c_i \in p^k,\ 1 \leq i \leq x : c_i \in \overleftarrow{C} \quad \text{and} \quad \forall c_j \in p^k,\ x < j \leq n : c_j \in \overrightarrow{C}$$

In this way, a pipeline $p^k$ can be seen as $p^k_x = \overleftarrow{p}^k_x, \overrightarrow{p}^k_{x+1}$. In other words, the pipeline can be divided into two sub-pipelines: the first, composed of $x$ capabilities, deployed on the provider side ($\overleftarrow{p}^k_x$), while the second, composed of $n - x$ capabilities deployed on the consumer side ($\overrightarrow{p}^k_{x+1}$). In case the index $x$ does not exist, the pipeline is meant to be deployed entirely at provider side. It is also possible that more than one $x$ can be found for a given pipeline. This generates, from $p^k$, a series of different configurations $\{p^k_x\}$.

In general, if $\overleftarrow{C} = \emptyset$ and $\overrightarrow{C} = \emptyset$, then there are $n+1$ possible configurations. Otherwise, the number of possible configurations is given by the formula $1 + |p^k| - |p^k \cap \overleftarrow{C}| - |p^k \cap \overrightarrow{C}|$.

The ability to divide the pipeline gives the opportunity to understand when the transmission from the provider to the consumer occurs. This data transmission affects the friction as it is a fundamental component of the effort required to move the data. To capture the effect of the network, a specific *transmission capability* $c_t$ is introduced in the pipeline. Similarly, we must consider the observability of the part of the pipeline running on the customer side to ensure the trustworthiness and verifiability requirements of the data provider. For this, we introduce the *observability capability* $c_o$, which must be executed at the end of the pipeline. Thus, given a pipeline $p^k_x$ configuration, then the correspondent deployable configuration is:

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

As said, considering the constraints expressed by $\overleftarrow{C}$ and $\overrightarrow{C}$, there could be different configurations. During the agreement phase, the consumer and the provider have the objective of identifying which is the best alternative. For example, consider $p^1 = \{$*filter, anonymize, convert*$\}$, where $\overleftarrow{C} = \{anonymize\}$, the only possible pipeline deployable configuration will be $\widehat{p_2}^1 = \{$*filter, anonymize,* $c_t$, *convert,* $c_o\}$.

The goal of the data friction is to compute, for a given deployable configuration, which is the effort that must be put in place in order to move the data from the initial FDP to the destination after applying all the transformations and the network transmission. *This effort is a combination of implementation effort and execution effort.*

***Implementation effort.*** Given a deployable configuration $\widehat{p_x}^k$, the implementation effort required to implement the pipeline is defined as the sum of the effort to implement the composing capabilities, thus:

$$E^I_{\widehat{p_x}^k} = \sum_{c_i \in \widehat{p_x}^k \ C} E^I(c_i)$$

where $E^I(c_i)$ is defined as the amount of work required by the provider to implement a capability. The implementation effort depends only on the complexity of the capability: a complex capability will require more effort to be implemented with respect to a simpler capability. An estimate of the implementation effort

required by a capability can be obtained by applying software cost estimation models such as SLIM, COCOMO [13] or, as used in our validation the lines of code. As some of the capabilities could be already in $C$ because of previous implementations, then the implementation effort does not count their effect.

As said, the implementation effort corresponds to the **static friction**. Only for aesthetic reasons, to make it aligned with the formulation of the static friction in physics, the static friction coefficient $\mu^I_{\widehat{p_x}^k}$ is introduced and defined as the ratio between the number of capabilities of $\widehat{p_x}^k$ that have already been implemented at the beginning of the implementation phase and the total amount of capabilities in the data pipeline, thus

$$\mu^I_{\widehat{p_x}^k} = \frac{\sum_{c_i \in \widehat{p_x}^k \setminus C} E^I(c)}{\sum_{c_i \in \widehat{p_x}^k} E^I(c)}, \;\; 0 \leq \mu^I_{\widehat{p_x}^k} \leq 1 \text{ , so that, } E^I_{\widehat{p_x}^k} = \mu^I_{\widehat{p_x}^k} \cdot \sum_{c_i \in \widehat{p_x}^k} E^I(c_i)$$

The formula is inversely proportional between the number of capabilities already implemented and the value of implementation friction, as the more capabilities have been already implemented, the lower the resulting implementation friction will be.

On this basis, the static effort does not only depend on the deployment configuration, but also on the history of the customers that are attached to the same FDP because of the evolution of $C$ as previously discussed. Thus, if a consumer is lucky, it can select an FDP and agrees on a pipeline where all the capabilities are included in $C$ so that $\mu^I_{\widehat{p_x}^k}$ is 0 and the corresponding effort $E^I$, thus the static friction, will be 0.

***Execution effort.*** Similarly to the implementation effort, the execution effort is defined as

$$E^E_{\widehat{p_x}^k}(d_{fdp}) = \sum_{c_i \in \widehat{p_x}^k} E^E(c_i, d)$$

With respect to the implementation effort, the execution effort of a capability $E^E(c_i, d)$ does not depend only on the complexity of the capability, but it might also depend on the size of the data object $d$ on which the capability is executed. For this reason, in case this dependency occurs:

$$E^E(c_i, d) = u(c_i) \cdot sizeof(d) \;\; \text{otherwise} \;\; E^E(c_i) = u(c_i)$$

where $u(c_i)$ is *unitary effort* $e(c)$ as the effort required to execute the capability $c$ on a single data item of the data object $d$. The value of the unitary effort $u(c_i)$ can be estimated experimentally or based on existing benchmarks and at this stage we assume it is linearly dependent on the size of the data.

For instance, the execution effort of the transmission capability $E^E(c_t, d)$ is the effort required to transmit a single data item of the data object. The bigger the data object size, the higher the effort to transmit it. While the execution

effort for the observability capability $E^E(c_o)$ is the effort required to observe all capabilities in $\overrightarrow{p}^k$. Thus, this effort is always proportional to the number of capabilities running not on the provider side.

As per the implementation effort, also for the execution effort the reuse deeply affects the friction. In fact, given a pipeline $\widehat{p}_x$, if some of the capabilities are shared with other pipelines, it might be possible that those capabilities have been already executed with the same data as input, thus producing the same data as output. Assuming that sort of caching mechanism is possible[4], $C'$ indicates the set of capabilities that have been executed.

For example, if the hospital has its data already filtered from a previous request, then only the capabilities for anonymization, transmission, and conversion have to be executed to complete the data sharing process.

On this basis, we introduced the coefficient of execution friction $\mu^E_{\widehat{p_x}^k}$ as the ratio between the amount of capabilities of $\widehat{p_x}^k$ that have already been executed and the total amount of capabilities in the pipeline that must be executed. Thus:

$$\mu^E_{\widehat{p_x}^k} = \frac{\sum_{c_i \in \widehat{p_x}^k \setminus C'} E^E(c)}{\sum_{c_i \in \widehat{p_x}^k} E^E(c)}, \;\; 0 \leq \mu^E_{\widehat{p_x}^k} \leq 1$$

Based on this coefficient, the **execution friction** can be expressed as:

$$E^E_{\widehat{p_x}^k} = \mu^E \cdot \sum_{c_i \in \widehat{p_x}^k} E^E(c_i)$$

## 5 Validation

***Assumptions and Setup:*** In this section, we present a preliminary validation of the model based on the running example. The goal of this validation is to prove the feasibility of the method and to show how it can be adopted. As, to the best of our knowledge, our approach is the first one that proposes a systematic way to measure the friction, comparisons with other approaches are not possible at this time. The validation is being performed on a synthetic dataset of patient data, that is stored as a *.csv* file that contains 124,150 rows of patient data with a total size of 36 MB. The implementation and execution of the data friction model are not bound to the utilization of certain software. For our validation, the capabilities will be implemented in Apache Airflow. All experiments are performed on a MacBook Pro with 16 GB of RAM and an Intel chip.

To estimate the effort and friction in this example, we define the metrics to calculate the efforts as: (i) for capability implementation as the lines of code (LoC) required and (ii) for the execution effort as the execution time in AirFlow. Consequently, the implementation effort is expressed as a pure number, while the execution effort in seconds.

[4] At this stage, only a binary situation is considered. A more articulated analysis that considers the possibility that only a subset of data have been considered will be investigated, as well as the definition of the caching policies.

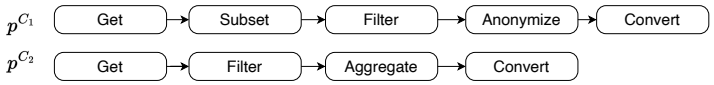


**Fig. 2.** Example Pipelines

| **Capability** | $E^I(c)(LoC)$ | $E^E(c)(s)$ |
|---|---|---|
| `get` | 16 | 9.01 |
| `subset` | 8 | 2.07 |
| `filter` | 8 | 0.44 |
| `anonymize` | 12 | 0.27 |
| `convert` | 12 | 0.28 |
| $\Sigma$ | **56** | **12.07** |

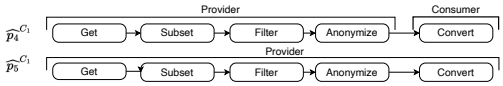


**Fig. 3.** $E^I(c)$ and $E^E(c)$ of the capabilities (left) and deployment conf. (right) of $p^{C_1}$ .

While LoC may not be the most accurate in a real-world scenario because the complexity of some capabilities depends on many factors, this metric can provide a first fair estimate of the implementation effort. Similarly, for the execution effort, more complex metrics could be used.

To estimate $c_t$, we assume a network bandwidth of $5MB/s$, thus $u(c_t) = 0.2s/MB$. About $c_o$, we utilize verifiable credentials stored on a blockchain (Ethereum) and IPFS and the execution effort depends on the size in IPFS and transaction cost. In our implementation, each generated credential is $\sim 36KB$, and $\sim 48310$ in gas costs. For this evaluation, we only consider the time to transfer of all observation credentials to the provider assuming the same bandwidth as above.

***Validation:*** In a multi-centric clinical trial, a hospital needs to exchange data about its patients with a medical center ($C_1$) and the other hospital ($C_2$) that takes part in the clinical trial. We assume that the hospital and the two data consumers have already agreed on the pipelines defined in Fig. 2 where the first to be considered is $p^{C_1}$, while the second is $p^{C_2}$.

Starting with $p^{C_1}$ for its capabilities, the left of Fig. 3 reports the implementation effort $E^I(c)$ required measured as Line of Codes (LoC) and the execution effort $E^E(c)$ as the time needed to execute on Apache Airflow.[5]

Assuming that $c_4 \in \overleftarrow{C}$, i.e., the capability $c_4 = anonymize$ must be executed at the provider side, thus there are only two possible deployment configurations: $\widehat{p_4}^{C_1}$, and $\widehat{p_5}^{C_1}$, (see the right of Fig. 3).

Since $p^{C_1}$ is the first pipeline to be implemented and executed the $C_P = \varnothing$. As result, both the implementation and execution friction coefficients for each deployment configuration, i.e, $\mu^I_{\widehat{p_4}^{C_1}}, \mu^I_{\widehat{p_5}^{C_1}}, \mu^E_{\widehat{p_4}^{C_1}}, \mu^E_{\widehat{p_5}^{C_1}}$, will be 1. The resulting implementation friction, i.e., the static friction, will be the same for both

[5] For the sake of brevity, instead of reporting the values of $u(c)$ for all the capabilities, which have to be multiplied by the size of the data, the resulting $E^E(c)$ is reported.

| Capability | $E^I(c)(LoC)$ | $E^E(c)(s)$ |
|---|---|---|
| `get` | 16 | 8.42 |
| `filter` | 8 | 2.33 |
| `aggregate` | 8 | 0.36 |
| `convert` | 15 | 0.49 |
| $\Sigma$ | **47** | **11.60** |

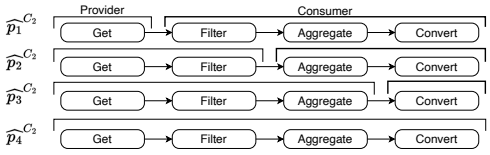


**Fig. 4.** $E^I(c)$ and $E^E(c)$ of the capabilities (left) and deployment conf. (right) of $p^{C_2}$ .

| Dep. config. | Output size | $E^E(c_t)$ | $E^E(c_o)$ |
|---|---|---|---|
| $\widehat{p_4}^{C_1}$ | 205 KB (from $c_4$) | 0.04 | 0.01 |
| $\widehat{p_5}^{C_1}$ | 477 KB (from $c_5$) | 0.09 | 0.0 |

**Table 2.** $E^E$ for $c_t$ and $c_o$ in $p^{C_1}$

| Dep. config. | Output size | $E^E(c_t)$ | $E^E(c_o)$ |
|---|---|---|---|
| $\widehat{p_1}^{C_2}$ | 3.9 MB (from $c_1$) | 3.9 | 0.02 |
| $\widehat{p_2}^{C_2}$ | 101 KB (from $c_2$) | 0.101 | 0.01 |
| $\widehat{p_3}^{C_2}$ | 94 B (from $c_3$) | ≈0 | ≈0 |
| $\widehat{p_4}^{C_2}$ | 539 B (from $c_4$) | ≈0 | ≈0 |

**Table 3.** $E^E$ for $c_t$ and $c_o$ in $p^{C_2}$

configurations as they are composed of the same capabilities:

$$E^I_{\widehat{p_4}^{C_1}} = E^I_{\widehat{p_5}^{C_1}} = 1 \cdot 56 = 56 \text{ LoC}$$

As a consequence, the choice of the deployment configuration depends on the execution effort that depends on the amount of data to be transmitted by $c_t$, and the effort related to the observation measured by $c_o$, as reported in Table 2. For the deployment configuration $\widehat{p_5}^{C_1}$ the value of $c_o$ is zero because all the capabilities are executed at provider side. Applying the execution friction formula, we obtain that $\widehat{p_4}^{C_1}$ requires less execution effort ($E^E_{\widehat{p_4}^{C_1}} = 1 \cdot (12.07\text{s} + 0.04\text{s} + 0.01\text{s}) = 12.12\text{s}$) than $\widehat{p_5}^{C_1}$ ($E^E_{\widehat{p_5}^{C_1}} = 1 \cdot (12.07\text{s} + 0.09\text{s} + 0.0\text{s}) = 12.16\text{s}$).

Moving to the data pipeline $p^{C_2}$, the implementation and execution effort of its four capabilities are reported in Fig. 4 (left). For the capabilities in common with the other pipeline, the values of $E^I(c)$ are equal to the previous case, while the $E^E(c)$ are different due to the different amounts of data.

Assuming that $c_1 \in \overleftarrow{C}$, thus the only capability required to be executed at the provider side is $c_1 = get$, the resulting possible deployment configurations are represented at the right of Fig. 4.

Since the pipeline $p^{C_2}$ happens after the pipeline $p^{C_1}$, then the set $C$ is no longer empty, but $C = \{get, subset, anonymize, convert\}$. Thus, the provider only has to implement the *aggregate* and *convert* capabilities, thus

$$\mu^I_{\widehat{p_1}^{C_2}} = \mu^I_{\widehat{p_2}^{C_2}} = \mu^I_{\widehat{p_3}^{C_2}} = \mu^I_{\widehat{p_4}^{C_2}} = \frac{8 + 15 \text{ LoC}}{47 \text{ LoC}} = 0.489$$

and the related $E^I = 23$ LoC. About the dynamic friction, assuming that the *get* task has been already executed, the $\mu^E_{\widehat{p_x}^{C_2}}$ coefficient will be

$$\mu^E_{\widehat{p_1}^{C_2}} = \mu^E_{\widehat{p_2}^{C_2}} = \mu^E_{\widehat{p_3}^{C_2}} = \mu^E_{\widehat{p_4}^{C_2}} = \frac{2.33 \text{ s} + 0.36 \text{ s} + 0.49 \text{ s}}{11.6 \text{ s}} = \frac{3.18 \text{ s}}{11.6 \text{ s}} = 0.027$$

so that, based on the efforts reported in Table 3, the fourth and fifth configurations are preferable:

$$E^E_{\widehat{p_1}C_2} = 0.027 \cdot (3.18 \text{ s} + 3.9 \text{ s} + 0.02 \text{ s}) = 0.191 \text{ s}$$

$$E^E_{\widehat{p_2}C_2} = 0.027 \cdot (3.18 \text{ s} + 0.101 \text{ s} + 0.01 \text{ s}) = 0.088 \text{ s}$$

$$E^E_{\widehat{p_3}C_2} = E^E_{\widehat{p_4}C_2} = 0.027 \cdot (3.18 \text{ s}) = 0.085 \text{ s}$$

It is important to underline that the values expressed by $E^I$ and $E^E$ are only intended to provide an estimate of the effort necessary to share data that can be relevant in the design phase and not of accurate performance values. For example, the approach is useful when you are looking for a service that provides data in a federation, or you are thinking about offering a data service to a potential set of customers.

## 6 Conclusions

Despite the growing importance of inter-organizational data sharing, several factors obstacle the flow of data. To measure these obstacles, this paper has proposed a physics-inspired data friction model. The model distinguishes between static and dynamic friction, which influence different stages of the sharing process. The proposed model can provide a useful framework, especially in a data mesh-driven ecosystem to evaluate the effort required to expose or to consume a data product.

While this initial work focuses on linear pipelines, future research should explore the cost and other friction factors like carbon and address the contractual complexities of reusing pipelines. We also plan further validation in a broader European context and incorporate larger datasets from diverse inter-organizational data-sharing scenarios. Preliminary validation indicates the model's ability to guide data providers in evaluating different data sharing deployments, considering both data transport and trustworthy observability. Further, a more systematic validation in terms of acceptance of the method will be conducted to also improve the significance of the approach. In this direction, we plan to build a supporting tool able to guide the data provider in the execution of the steps required by the method.

## Acknowledgements

Funded by the European Union (TEADAL, 101070186). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

## References


1. Bates, J.: The politics of data friction. Journal of Documentation **74**(2), 412–429 (2018)
2. Dalmolen, S., Bastiaansen, H., Kollenstart, M., Punter, M.: Infrastructural sovereignty over agreement and transaction data ('metadata') in an open network-model for multilateral sharing of sensitive data. In: 40th International Conference on Information Systems, ICIS 2019. Association for Information Systems (2020)
3. Dehghani, Z.: Data Mesh. O'Reilly Media, Inc. (2022)
4. Dehghani, Z., Fowler, M.: How to move beyond a monolithic data lake to a distributed data mesh (2019), https://martinfowler.com/articles/data-monolith-to-mesh.html
5. D'Amore, J.D., et al.: Clinical data sharing improves quality measurement and patient safety. Journal of the American Medical Informatics Association **28**(7), 1534–1542 (03 2021). https://doi.org/10.1093/jamia/ocab039
6. Edwards, P.N.: A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming. MIT Press (2010)
7. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall Professional Technical Reference, Upper Saddle River, NJ (2005)
8. Falconi, M., Plebani, P.: Adopting data mesh principles to boost data sharing for clinical trials. In: ICDH 2023. pp. 298–306 (2023)
9. Gelhaar, J., Gürpinar, T., Henke, M., Otto, B.: Towards a taxonomy of incentive mechanisms for data sharing in data ecosystems. In: PACIS. p. 121 (2021)
10. Goedegebuure, A., et al.: Data mesh: a systematic gray literature review (2023)
11. Jussen, I., Schweihoff, J., Dahms, V., Möller, F., Otto, B.: Data sharing fundamentals: characteristics and definition. In: Proceedings of the 56th Hawaii International Conference on System Sciences (HICSS). Maui, Hawaii, USA (2023)
12. Leonelli, S., et al.: Making open data work for plant scientists. Journal of Experimental Botany **64**(14), 4109–4117 (2013)
13. Leung, H., Fan, Z.: Software cost estimation. In: Handbook of Software Engineering and Knowledge Engineering: Volume II: Emerging Technologies, pp. 307–324. World Scientific (2002)
14. Murray-Rust, P.: Open data in science. Nature Precedings pp. 1–1 (2008)
15. Otto, B.: A federated infrastructure for european data spaces. Communications of the ACM **65**(4), 44–45 (2022)