INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




# Transferências Assíncronas de Dados Não-Finitos através de Data Spaces

**AFONSO MIGUEL FERNANDES CORREIA**
Setembro de 2025

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

# Asynchronous Non-Finite Data Transfers through Data Spaces

**Afonso Correia**

**A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering**

**Advisor: Dr. Jorge Coelho**
**Supervisor: Dr. Jorge Coelho**

Porto, September 27, 2025

# Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, September 27, 2025

# Dedicatory

To the DSF team for their incredible support, proving that, with the right people, no data flow is too complex.

# Abstract

Data has become an increasingly vital asset for organizations, as the volume of information being generated reaches an all-time high. This growth affects not only individual enterprises but also entire supply chains, with data sharing becoming key to organizational success. Yet, data exchange remains a complex endeavor, as businesses must comply with strict security, sovereignty and privacy requirements.

Data Spaces were introduced to address these challenges, fostering ecosystems of trust where organizations can share information while preserving data sovereignty. At the core of this concept lies the Connector, a key component that acts as the gateway for data to flow between Data Space participants. Connectors support both synchronous and asynchronous transfers, handling either finite or non-finite data.

A prominent initiative in this field is the Eclipse Dataspace Components (EDC) project, an open-source framework for building Data Space components. Although this project aims to establish data sharing environments where any type of transfer is possible, it lacked support for asynchronous non-finite data transfers. The goal of this dissertation is to develop a new functionality for the EDC project to enable this type of data transfers, addressing this gap.

As part of this work, interviews were conducted with members of an organization participating in the Catena-X Data Space. These revealed that asynchronous non-finite data transfers were already taking place through workarounds that were either complex, costly, or incompatible with the core principles of Data Spaces. The interviewees also shared their expectations on how these transfers should occur instead, allowing for the definition of technical requirements.

Building on the insights gathered from the interviews, a design for handling asynchronous non-finite data transfers was proposed to the EDC project. This proposal was discussed within the community, being adjusted until a consensus was reached. As a result, the final approach introduces three contributions to the project: a method to identify non-finite data, a service to perform asynchronous non-finite data transfers, and a mechanism to trigger transfers on demand.

Following this design, a solution was developed to fulfill the defined requirements and tackle the identified problem. Its outcomes were presented to the participants of the interview process, receiving positive feedback. On a technical level, the EDC maintainers approved the contributions, including the feature for the upcoming release.

In the end, the defined objectives were achieved. Besides contributing the developed feature, this dissertation also provides a detailed overview of Data Spaces and the initiatives in this field. In conclusion, continued development on projects such as the EDC are essential to potentiate the benefits of Data Spaces, creating value for organizations seeking to thrive in this data-driven ecosystem.

**Keywords:** Data Spaces, Eclipse Dataspace Components, Data Transfers, Non-Finite Data

# Resumo

A engenharia e análise de dados tornaram-se essenciais para as organizações, levando a que o volume de informação gerada tenha atingido níveis históricos. Este crescimento não afetou apenas os processos internos das empresas, mas também cadeias de abastecimento inteiras, de forma que a partilha de dados se tornou um fator determinante para o sucesso organizacional. No entanto, a troca de informação continua a ser um obstáculo para as empresas, já que estas precisam de cumprir requisitos de segurança, soberania e privacidade.

Os Data Spaces foram criados para enfrentar estes desafios, estabelecendo ecossistemas de confiança nos quais organizações podem trocar dados entre si. No centro deste conceito está o Connector, um componente fundamental que atua como porta de entrada e saída de dados entre os participantes dos Data Spaces. Os Connectors suportam transferências síncronas e assíncronas, e permitem a troca tanto de dados finitos como não finitos.

Uma iniciativa de destaque nesta área é o projeto EDC, uma framework open-source para desenvolver componentes de Data Spaces. Embora este projeto vise estabelecer espaços de partilha de dados onde qualquer tipo de transferência seja possivel, o suporte para transferências assíncronas de dados não finitos encontrava-se em falta. Esta dissertação tem como objetivo desenvolver uma nova funcionalidade para o projeto EDC, colmatando esta lacuna de modo a possibilitar este tipo de transferências.

Para tal, foram realizadas entrevistas com membros de uma organização participante do Data Space Catena-X. As entrevistas revelaram que transferências assíncronas de dados não finitos já ocorriam por meio de soluções alternativas, mas estas eram complexas, dispendiosas ou incompatíveis com os princípios fundamentais dos Data Spaces. Os entrevistados também partilharam as suas expectativas sobre como estas transferências deveriam funcionar, permitindo a definição de requisitos técnicos.

Com base nos resultados das entrevistas, foi elaborada uma proposta para o projeto EDC sobre como lidar com transferências assíncronas de dados não finitos. A proposta foi discutida pela comunidade e ajustada até se alcançar um consenso. Como resultado, a abordagem final definiu três contribuições para o projeto: um método para identificar dados não finitos, um serviço para realizar transferências assíncronas de dados não finitos e um mecanismo para acionar transferências sob demanda.

Seguindo esta abordagem, foi desenvolvida uma solução que cumpriu todos os requisitos definidos e resolveu o problema identificado. De um ponto de vista técnico, os responsáveis pela manutenção do projeto EDC aprovaram as contribuições, incluíndo a funcionalidade para lançamento na próxima versão. Os resultados foram também apresentados aos participantes das entrevistas, que deram um parecer positivo.

Por fim, os objetivos definidos foram alcançados. Além de contribuir com a funcionalidade desenvolvida, esta dissertação também proporcionou uma visão detalhada sobre os Data Spaces e as iniciativas desta área. Conclui-se que continuar a desenvolver projetos como

o EDC será essencial para potenciar os benefícios dos Data Spaces, criando valor para organizações que procuram prosperar nestes ecossistemas.

# Acknowledgement

Throughout this past year I've dedicated my time and energy into this dissertation, with the goal of completing my master's degree in software engineering.

The journey was not easy, and hardly enjoyable at times, but fortunately there were people along the way that helped me in this process. I would like to express my gratitude to everyone that, in some way or another, contributed to the success of this dissertation.

To my family for the unconditional support, for always being there, and for helping me be consistent when all I wanted was to avoid writing.

To my girlfriend for keeping me company, for listening to me, for taking good care of me, and for cheering me up in the most stressful moments.

To the DSF team for giving me feedback on my work, for letting me engage in many interesting discussions, and for the confidence deposited in me.

To my supervisor for quickly reviewing my work, for guiding me, and for helping me achieve the best possible results.

To everyone that I did not mention, but positively impacted my development as a software engineer.

In the end, this was only possible because of you. **To everyone, a big thank you.**

# Contents

# List of Figures

# List of Tables

# List of Source Code

# List of Acronyms

| | |
|---|---|
| ALB | Application Load Balancer. |
| API | Application Programming Interface. |
| AWS | Amazon Web Services. |
| CD | Continuous Delivery. |
| CI | Continuous Integration. |
| DPF | Data Plane Framework. |
| DSC | Dataspace Connector. |
| DSP | Dataspace Protocol. |
| ECS | Elastic Container Service. |
| EDC | Eclipse Dataspace Components. |
| EKS | Elastic Kubernetes Service. |
| IDS | International Data Spaces. |
| IDSA | International Data Spaces Association. |
| ODRL | Open Digital Rights Language. |
| OSS | Open Source Software. |
| PRISMA | Preferred Reporting Items for Systematic reviews and Meta-Analyses. |
| RAM | Reference Architecture Model. |
| VPC | Virtual Private Cloud. |

# Chapter 1

# Introduction

This chapter introduces the problem addressed in this dissertation, providing context and outlining the research objectives and approach. Additionally, it reflects on the contributions made and considers ethical implications. Finally, it offers a detailed overview of the document's structure.

## 1.1 Context

In today's digital era, the volume of data generated across the globe has reached unprecedented levels, driven by advancements in technology and the continuous growth of online activity [1].

This abundance of data has become a critical asset for organizations, serving as the foundation for decision-making, innovation, and strategic planning. Properly harnessed, data empowers businesses to gain valuable insights into consumer behavior, optimize operations, and predict future trends [2].

The growing reliance on data is not only transforming individual organizations but entire supply chains, as collaboration between companies is becoming more and more prominent. The ability to share data securely and efficiently has become a cornerstone of organizational success [3].

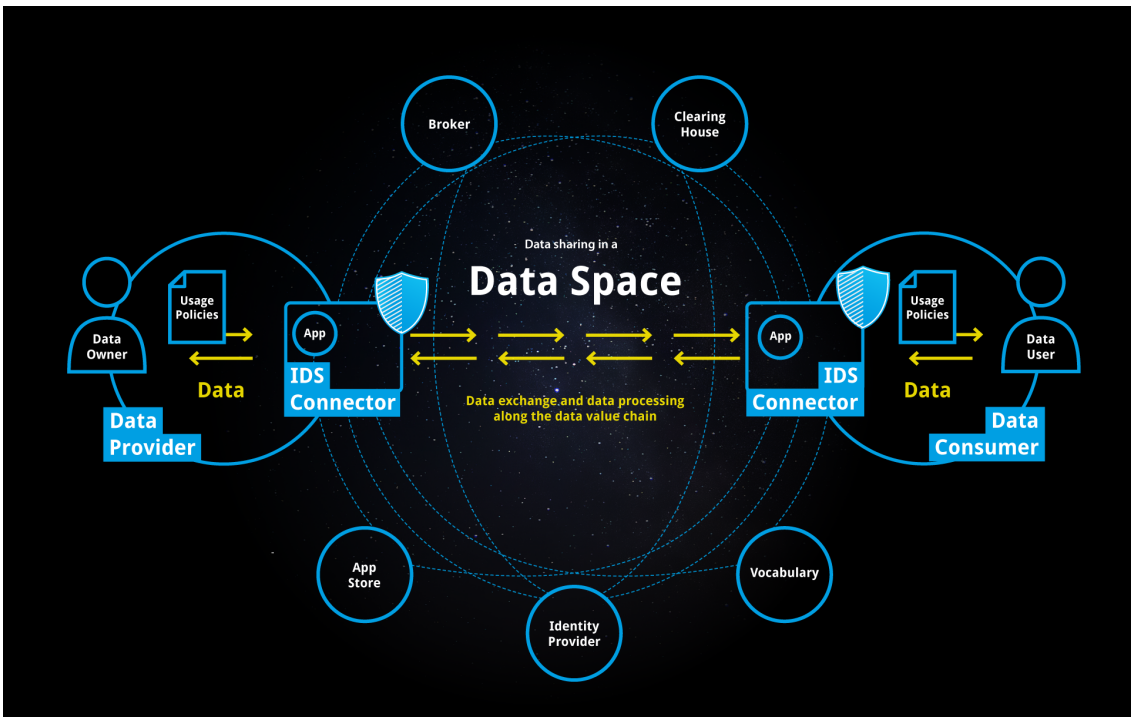
Nevertheless, data exchange remains a significant challenge. Organizations must address the complexities of ensuring data sovereignty, protecting sensitive information, and complying with a growing array of regulations. Striking the right balance between leveraging shared data and maintaining trust is essential for organizations seeking to thrive in this interconnected digital age [4, 5].

The concept of Data Spaces was created to overcome the challenges of enabling data exchange. Their primary goal is to foster an ecosystem of trust, where organizations can collaborate by sharing data securely while maintaining sovereignty and ownership over their assets [6].

Data Spaces create a distributed network of data suppliers and customers, where participants are responsible for deploying the necessary components to connect with each others. The most crucial component in this setup is the Connector, which serves as the gateway for both providing and consuming data [7].

Although the main function of Connectors is to allow data to be transferred between participants, they also play a crucial role in enabling participants to publish offers for their data,

Figure 1.1: Data sharing in a Data Space [9]




overlooking contract negotiations under usage policies, defining agreements, among other essential features [8].

Participants of a Data Space can use their Connectors to exchange data either synchronously, retrieving the data on request, or asynchronously, receiving the data at some point in the future [8]. Figure 1.1 illustrates how Data Spaces enable data sharing.

One of the leading initiatives in Data Space technology is the Eclipse Dataspace Components (EDC) project [10], an Open Source Software (OSS) effort supported by the Eclipse Foundation, which aims to establish a framework for implementing the core components of a Data Space.

This framework provides the tools to build EDC Connectors [11], which are implementations of Connectors designed to operate independently of any specific Data Space. While already in use across various Data Spaces, their most notable application is within Catena-X [10].

The Catena-X association is a collaborative network dedicated to driving innovation and standardization in the automotive industry. It supports the Catena-X Data Space, creating a specialized ecosystem for data sharing across the automotive supply chain [12].

## 1.2 Problem

Data sources can provide either finite or non-finite data. Finite data is data that is defined by a finite set, having a clear start and end, such as files or images. On the other hand, non-finite data is data that is defined by an infinite set or has no specified end, such as streams or data from Application Programming Interfaces (APIs) [13].

The current implementation of the EDC Connector allows finite data to be transferred both synchronously and asynchronously. However, non-finite data can only be transferred synchronously. This limitation impacts potential business use cases within the Data Space, as providers of non-finite data cannot distribute it asynchronously.

Let's look at an analogy for a possible use case within a Data Space for mobility and automotive data. Consider a car brand with numerous dealerships worldwide. Each month, the dealerships generate sales reports and send them to the brand's headquarters. Within the Data Space context, the dealerships act as data suppliers, and the car brand serves as the data customer. The brand wishes to receive these reports automatically at the end of each month without needing to request them repeatedly — an asynchronous transfer of non-finite data, as the reports continue indefinitely with each passing month.

Although workarounds exist, they may not be practical for all scenarios and can lead to inefficiencies, resulting in unnecessary costs and wasted time.

## 1.3 Objectives

This dissertation focuses on developing a new functionality for EDC Connectors, aiming to address the identified problem. To achieve this, the following objectives have been defined:

- Understand what are Data Spaces, their architecture, and the value they bring to businesses;
- Understand how the EDC Connector works as a tool to exchange data;
- Develop support for asynchronous data transfers of non-finite data.

Based on these objectives, a research question was formulated. Research questions are the fundamental queries that a study aims to answer, helping to shape its scope and keep the investigation aligned with its goals [14]. For this dissertation, the following research question was identified:

**How can the EDC Connector be extended to support asynchronous transfers of non-finite data while aligning with the principles of Data Spaces?**

This question reflects the objectives of the dissertation, focusing on the technical aspects required to develop a feature that enables asynchronous transfers of non-finite data through the EDC Connector. It also considers the context needed to understand the importance of this feature and its impact in businesses across Data Spaces.

## 1.4 Approach

To address the research question and meet the defined objectives, this dissertation adopts an approach grounded on the **Design and Creation** research strategy. This strategy focuses on developing artifacts as a means of generating knowledge, rather than relying solely on passive observation or theoretical abstraction [15].

Within the field of information systems and computing, Design and Creation is particularly suitable for projects where a technical solution is developed to address a real-world problem [15].

To support the development process, an initial investigation was conducted, gathering essential information through qualitative data generation methods [15].

The first data generation method employed was documental analysis. A state-of-the-art review was performed, examining existing documentation on Data Spaces, the Connector, and the role of non-finite data in these contexts. The analysis considered both technical and business perspectives.

Interviews were used as the second data generation method. Key stakeholders were consulted to understand the business impact of the proposed feature, while also gathering technical requirements to guide the development process.

The information collected through these methods was carefully analyzed, serving as the foundation for the design of a technical solution that addresses the research question. Establishing this design marked the transition into the creation phase.

During this phase, the insights obtained were applied to implement a solution that enables asynchronous transfers of non-finite data. This solution was thoroughly tested to ensure its quality.

To conclude the dissertation, the outcomes of the creation phase were evaluated, determining if they met the design requirements and effectively addressed the research question.

## 1.5 Contributions

The main contribution of this dissertation results from the development of a new feature for the EDC Connector.

As of November 2024, the Catena-X Data Space alone includes 193 partners [16]. Given the open-source nature of the EDC project, its adoption across multiple Data Spaces, and the growing community of organizations embracing this technology, the functionality developed in this work has the potential to support multiple use cases and benefit numerous enterprises.

The success of this contribution is determined by its acceptance by the EDC committers, a group of maintainers of the EDC code base. By reviewing every feature proposal, they have the power to ultimately decide which features are integrated.

Moreover, this document offers a comprehensive state-of-the-art analysis on Data Spaces, which could serve as a valuable resource for future researchers in the field.

## 1.6 Ethical Considerations

When working with data, it is crucial to address ethical aspects. Data Spaces inherently account for these considerations, with data privacy, data sovereignty and security serving as some of the core pillars of this concept [6].

Throughout this dissertation, the student took deliberate steps to ensure that no data or information can be traced back to the originating organizations. All information used in this dissertation is either publicly accessible or discovered during the development process.

For the interviews, information that could identify the interviewees was kept to a minimum, disclosing only what is essential for the purposes of this dissertation. Additionally, sensitive information regarding their activities was omitted from this dissertation. The applied measures were agreed upon with both the interviews participants and their organizations.

Finally, within the context of an OSS project, the code is inherently publicly visible, ensuring that no confidentiality is breached.

## 1.7 Document Structure

This document is structured into 6 chapters: Introduction, State of the Art, Problem Analysis & Requirements, Development, Evaluation and Conclusions.

The **Introduction** chapter presents the dissertation topic, along with other essential aspects. It begins by providing context for a better understanding of the subject matter, followed by a detailed description of the problem, the objectives, and the approach taken to achieve them. Additionally, it highlights the contributions made and discusses relevant ethical considerations. The chapter concludes with an overview of the document's structure.

The **State of the Art** chapter provides a comprehensive analysis of the systematic review conducted to identify the current state of the topics addressed in this dissertation. It includes the research methodology used, the results obtained and a thorough analysis and discussion of these findings.

The **Problem Analysis & Requirements** chapter describes the elicitation process conducted to uncover user needs, pain points and expectations. It starts by outlining the methodology used to direct and analyze the process, followed by a presentation of the results obtained, and finalizing with a discussion of the outcomes.

The **Development** chapter describes the work undertaken to design, implement, test and deploy the proposed feature. It details the applied methodology, covering the EDC contribution process and the planning conducted for this phase. It then outlines the developed contributions, highlighting important design decisions and implementation details. The chapter concludes by presenting the quality assurance mechanisms adopted and explaining the deployment process for hosting an EDC Connector with the new feature.

The **Evaluation** chapter provides a critical analysis of the developed contributions. It examines how they align with the defined functional and non-functional requirements and assesses whether the applied processes proved effective in delivering meaningful outcomes.

The **Conclusions** chapter reflects on the work carried out across all phases of the dissertation. It starts by reviewing the achievements obtained for each defined objective, followed by a discussion of the limitations encountered. The chapter concludes with some closing remarks that provide a forward-looking perspective on the topics explored in this dissertation.

# Chapter 2

# State of the Art

This chapter presents a state-of-the-art analysis of the Data Spaces ecosystem, providing greater context on the key components and processes involved.

It begins by detailing the research methodology, moves on to present the results and concludes with a summary and analysis of the findings.

## 2.1 Research Methodology

The research process followed the principles of Evidence-Based Software Engineering, ensuring that all findings are grounded in empirical research. This approach aims to identify the strongest available evidence to inform and support decision making [17].

To achieve this, the literature analysis was conducted using a systematic mapping review, a structured methodology for categorizing existing research. Through the definition of a mapping study, this approach identifies, classifies and analyzes relevant publications, providing a broad overview of the research landscape [18].

The research followed the Preferred Reporting Items for Systematic reviews and Meta-Analyses (PRISMA) statement, a comprehensive checklist designed to guide the reporting of systematic reviews. This approach seeks to identify a set of candidate papers for analysis, followed by steps to remove duplicates, screen for relevance, classify records through coding, and ultimately analyze the final set of studies [19].

### 2.1.1 Data Sources

Before starting the research, it is essential to identify suitable data sources for articles. It is recommended to select a small number of databases, prioritizing those with strong reputations and comprehensive coverage [20]. Table 2.1 presents the data sources chosen for this review.

Table 2.1: Data sources

| **Database** | **URL** |
|---|---|
| ACM | `https://dl.acm.org` |
| IEEE Xplore | `https://ieeexplore.ieee.org/` |
| ScienceDirect | `https://www.sciencedirect.com/` |

ACM offers a "Full-Text" collection, containing articles from a wide range of publishers, and a "Guide to Computing Literature" collection, which extends it by including citations and links to all major computing publishers. The latter is more suitable for this field of research and was therefore selected for article searches.

Given their size, duplicate articles are expected and will be promptly removed, as anticipated by PRISMA.

### 2.1.2 Search Strings

To ensure that only the most relevant studies were analyzed, search strings representing the research question domain were formulated. This was achieved by identifying relevant keywords on the research question, and deriving search terms with possible synonyms. Table 2.2 outlines the keywords and their corresponding search terms.

Table 2.2: Search terms

| RQ Keyword | Search Terms |
|---|---|
| Connector | EDC / Eclipse Dataspace Components / Dataspace Connector |
| Data Spaces | Data Space / IDS |
| Transfers | Transfer process / Data transfer / Data flow |

Note that keywords like "Connector" and "Transfers" are too broad for effective searching, therefore context-specific alternatives like "Dataspace Connector" and "Data transfer" were preferred. While this approach may filter out some relevant papers, it also helps excluding a large number of unrelated results.

Additional keywords like "asynchronous" and "non-finite data" could also be derived from the research question. Although crucial to the research, they were excluded to maintain a broader dataset. This allows the inclusion of papers that, while not directly focused on these aspects, may still offer relevant insights into Data Spaces.

Query strings were constructed by combining search terms related to the same keywords using the OR operator and search terms related to different keywords using the AND operator. The resulting search string was the following:

**(EDC OR "Eclipse Dataspace Components" OR "Dataspace Connector") AND ("Data Space" OR IDS) AND ("Transfer process" OR "Data transfer" OR "Data flow")**

Different data sources require tailored search strings to account for variations in how their search engines process queries. Factors such as plural forms and spelling variants were considered.

For example, ACM and IEEE Xplore require all linguistic alternatives to be manually included, but they also support wildcards to help broaden the search scope.

On the other hand, ScienceDirect does not support wildcards but uses stemming to automatically account for similar variations of a search term. However, it also restricts queries to a maximum of eight boolean operators, which limited the search scope.

As a result, the final dataset may vary depending on how the search string is adjusted to each data source. Table 2.3 presents the number of results obtained.

Table 2.3: Search results

| Data Source | Number of Results |
|---|---|
| ACM | 143 |
| IEEE Xplore | 2 |
| ScienceDirect | 293 |
| Total | 438 |

The limited number of studies retrieved from IEEE Xplore was notable compared to other data sources. This suggests that this database is underrepresented in the research topic, possibly due to its primary focus being electrical engineering.

### 2.1.3 Screening Criteria

It is common to retrieve a substantial number of articles from the initial database search, even with well-constructed query strings. Therefore, additional criteria should be established to determine which studies are relevant for review and which can be discarded, refining the search results.

The screening process applies both inclusion and exclusion criteria, ranging from basic filters, such as the language of the article, to more specific conditions aligned with the dissertation's objectives.

To ensure the quality of the information, only studies written in English and published from 2022 onward were included. In addition, the articles had to be peer reviewed and directly focused on the research question topic. Tables 2.4 and 2.5 present the defined inclusion and exclusion criteria, respectively.

Table 2.4: Inclusion criteria

| ID | Criteria |
|---|---|
| IC1 | The source describes the Data Space architecture and/or the Data Space components |
| IC2 | The source describes use cases of the Data Spaces and their business impact |
| IC3 | The source describes transfer processes and/or data transfer mechanisms within Data Spaces |
| IC4 | The source describes how non-finite data is handled by Data Spaces |

Table 2.5: Exclusion criteria

| ID | Criteria |
|---|---|
| EC1 | The source was published prior to 2022 |
| EC2 | The source is not written in English |
| EC3 | The source is not peer-reviewed |

The selected databases offer filters that aid in applying the exclusion criteria. However, to determine whether an article matched the inclusion criteria, the title, abstract, and relevant paragraphs containing the selected search terms were carefully reviewed.

Applying these criteria to the dataset obtained from the database searches reduced the total number of entries to 27 articles.

### 2.1.4 Data Extraction

After refining the dataset to a manageable size, the coding phase of the PRISMA framework begins, during which each screened article is thoroughly reviewed in full. The goal of this stage is to extract and record the resulting data.

During this process, notes were taken for each article, detailing the questions it addresses and the overall topics covered. After reading each paper, its contribution was ranked to determine its relevance to the research question using a classification system.

Occasionally, studies that passed the initial screening process may, upon closer examination, be deemed irrelevant and subsequently discarded. Of the 27 screened papers, 9 were excluded for lacking relevant information on any of the items in the classification system, leaving a total of 18 articles included.

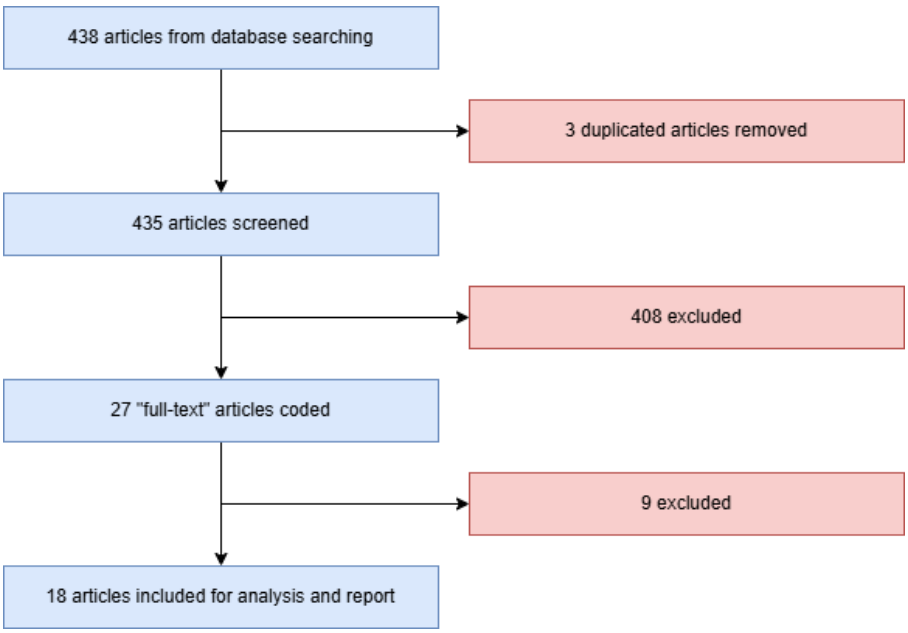
Table 2.6 presents the classification system and the number of articles associated with each item. The items are ordered by increasing relevance to the research question. Note that some articles relate to multiple classification items, expanding their contribution.

Table 2.6: Classification system

| ID | Description | No. Articles |
|---|---|---|
| CLS1 | The article provides insights on business use cases using Data Spaces | 9 |
| CLS2 | The article provides insights on Data Space architecture | 8 |
| CLS3 | The article provides insights on how the Connector leverages data transfers | 8 |
| CLS4 | The article provides insights into the management of non-finite data within Data Spaces | 0 |

None of the reviewed articles addressed how non-finite data is handled by Connectors, highlighting a gap in the existing literature and showing the novelty of this study in this area.

Figure 2.1: PRISMA flowchart




With the conclusion of the the coding phase, the research is complete. Figure 2.1 summarizes the research process, following the PRISMA flowchart.

As an important final note, during the analysis of the screened articles, it became clear that many papers heavily referenced both the International Data Spaces (IDS) Reference Architecture Model (RAM) [8] and the Dataspace Protocol (DSP) [13]. A closer review of these documents confirmed their substantial relevance to the dissertation's objectives.

Additionally, the official EDC documentation [21] also provides crucial information for the development of this component.

Although PRISMA does not formally account for such cases, the significance of these documents warranted a thorough examination, leading to their inclusion in the bibliography.

## 2.2 Results

This section presents the results of the research process, providing insights gathered from the selected articles.

### 2.2.1 Data Spaces

Data Spaces are digital, federated platforms formed out of the need for organizations to exchange data among themselves in a secure, controlled, and standardized manner. By bringing together participants interested in sharing data, they create decentralized ecosystems built on mutual trust and collaboration [22, 23].

In essence, Data Spaces offer tools to provide and consume data. Alberto Montero Fernández et al. [24] identifies the key features of the Data Spaces to be:

- **Interoperability:** Data Spaces seamlessly integrate data from diverse data sources and formats.

- **Access Control:** Data Spaces implement robust mechanisms to manage and restrict data access.
- **Security and Privacy:** Data Spaces ensure data protection through advanced security measures.
- **Data Sovereignty:** Data Spaces empower data providers to retain ownership and control over how their data is shared.
- **Value Creation:** Data Spaces promote the creation of innovative products and services.

This concept has recently gained traction among European organizations, with the Catena-X project serving as a prominent example. This project unites various German organizations, establishing a Data Space specifically designed for the automotive industry [23, 25].

The European Commission also joined the trend, proposing the creation of 10 common Data Spaces across strategic industries [23, 24]. Some notable examples are the European Health Data Space [26, 27] and the Common European Agricultural Data Space [28].

Developed by the International Data Spaces Association (IDSA), the IDS is a leading initiative created to establish a reference architecture for Data Spaces based in their core principles. By defining essential concepts, standards, and processes, it enables participants to securely and reliably exchange data [22, 29, 30].

Although the concept of Data Spaces is innovative and promising, implementing IDS is complex and not yet economically viable for many, particularly small and medium enterprises. Adoption is hindered by privacy concerns, legal barriers like GDPR, and fears of losing competitive advantages, prompting calls for business cases to demonstrate its feasibility [31, 32].

### 2.2.2 International Data Spaces Architecture

The IDS RAM, defined and maintained by IDSA, is a comprehensive document describing how to construct Data Spaces compliant with the IDS standard. Ilka Jussen et al. [33] describes the five layers that form this architecture:

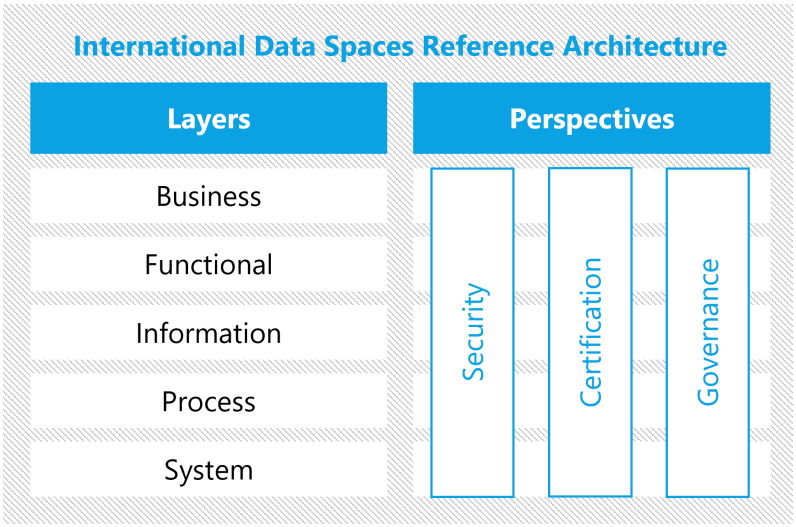
- **Business layer:** Specifies the participants of the Data Space.
- **Functional layer:** Identifies the functional requirements and features.
- **Information layer:** Defines the information model and vocabulary.
- **Process layer:** Describes the interactions between the different components.
- **System layer:** Provides insights regarding the implementation of the components.

Additionally, the RAM comprises three perspectives that need to be implemented across all five layers: **Security**, **Certification**, and **Governance** [33]. Figure 2.2 illustrates this structure.

#### Business Layer

Participants in a Data Space assume well defined roles, categorizing them into core participants, intermediaries, software developers or governance bodies [33, 34]. For the purpose of this analysis, we will focus exclusively on the core participants and intermediaries.

Figure 2.2: IDS RAM layers and perspectives [8]




Each data exchange within a Data Space involves two core participants: one acting as the Data Supplier and the other as the Data Customer [27, 31, 34].

The Data Supplier is responsible for introducing data into the Data Space. Depending on the use case and operations model, this role is typically decomposed into two specialized roles:

- **Data Owner:** Has ownership over the data and defines access control to it.
- **Data Provider:** Ensures the data is technically accessible within the Data Space.

The Data Customer remains on the other end of the exchange, being responsible for consuming data from the Data Space. Depending on the use case and operations model, this role is also usually broken down into two specialized roles:

- **Data Consumer:** Responsible for technically receiving the data.
- **Data User:** Has usage rights over the data and benefits from its consumption.

Intermediaries are trusted entities that assume central roles in the Data Space [34]. They include the:

- **Broker Service Provider:** Maintains a repository of metadata published by Data Providers and retrieved by Data Consumers.
- **Clearing House:** Logs data transactions in the Data Space for billing and validation purposes.
- **Identity Provider:** Validates identities within the Data Space using digital certificates or verifiable credentials.
- **Vocabulary Provider:** Oversees vocabularies for proper description of data in the Data Space.
- **App Store Provider:** Distributes Data Apps, providing tools for data processing.

### System Layer

Although the system layer is the last to be described in the IDS RAM, presenting the Data Space components and their interactions at this stage will provide valuable context for the sections that follow.

The IDS Connector, or simply Connector, is the main technical component of the Data Space, acting as the gateway for data to flow to and from a participant of the Data Space. Data Providers and Data Consumers operate Connectors to enable the data exchange [8, 35, 36].

The Identity Provider component, managed by the participant assuming the Identity Provider role, is responsible for providing identity and access management for the other components within the Data Space [8].

The App Store, managed by the App Store Provider, is responsible for distributing Data Apps - independent, functional and reusable software components that can be deployed, executed, and managed on an IDS Connector. Their goal is to process information either before or after the data transfer [8].

The Metadata Broker, managed by the Broker Service Provider, is the component responsible for managing the metadata that describes the data published by Data Providers. It enables Data Consumers to query and locate the data they need [8].

The Clearing House component, managed by the participant assuming the Clearing House role, is responsible for logging and persisting all information relevant for clearing and billing [8].

The Vocabulary Hub, managed by the Vocabulary Provider, is responsible for managing the vocabularies used within the Data Space. These machine-readable standards are essential for describing the data published by Data Providers [8].

Figure 2.3 shows how these components interact with each other, omitting the Identity Provider for simplicity as it engages with every other component.

### Functional Layer

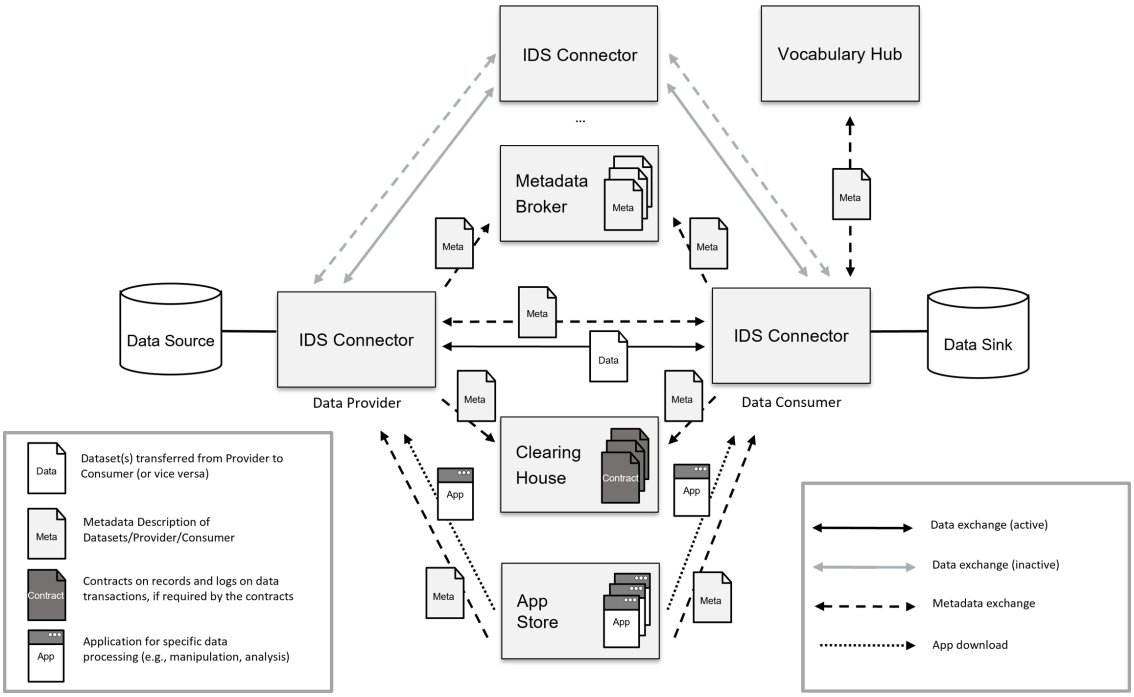
The IDS RAM specifies six high-level functional requirement groups that every IDS implementation must fulfill. These requirements align with the core features and objectives of Data Spaces [8, 33].

**Trust**, although usually associated with non-functional requirements, is described as a functional requirement in IDS. It comprises features associated with identity management and user certification [8, 33].

**Security and Data Sovereignty** are also commonly considered non-functional requirements, but classified as a functional requirements for IDS implementations. They encompass features associated with authentication and authorization, enforcement of usage policies, secure communication and design, and technical certification of participant components within the Data Space [8, 33, 37].

The **Ecosystem of Data** represents the functional requirements aimed at standardizing information within the Data Space. It focuses on enabling data source descriptions, metadata brokering, and the adoption of common vocabularies [8, 33].

Figure 2.3: IDS components and interactions [8]




**Standardized Interoperability** represents the functional requirements that enable data exchange. This includes functionalities for operating the Connector, such as publishing data, negotiating contracts, and transferring data [8, 33].

**Value-Adding Applications** encompass the functional requirements associated with the provisioning of Data Apps. These requirements include features for data processing and transformation, the implementation and publication of Data Apps in the App Store, and their subsequent installation and support [8, 33].

**Data Markets** address the monetization of data by incorporating features for clearing and billing, enforcing data governance, and ensuring compliance with legal requirements [8, 33].

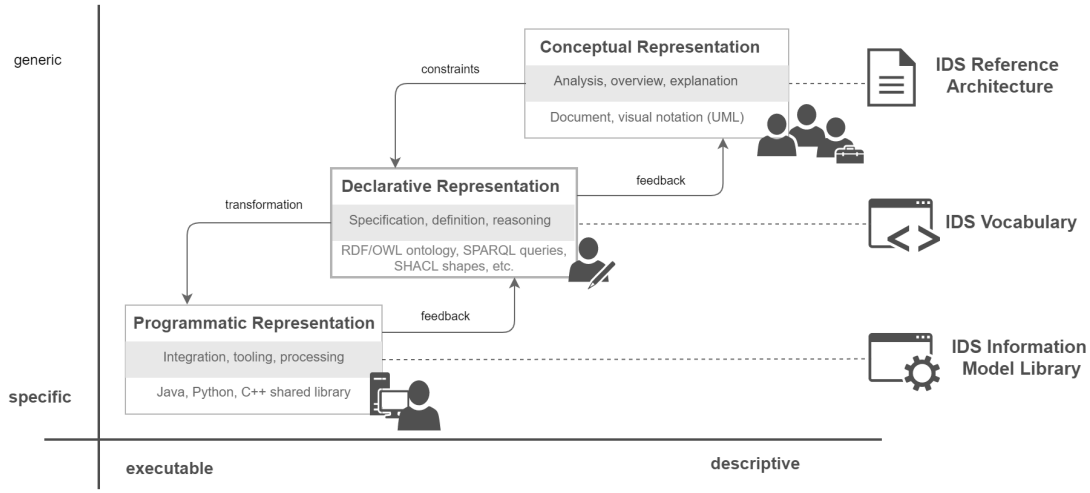
### Information Layer

Domain models and meta-models are not enforced by IDS. Instead, this responsibility is delegated to the vocabularies employed in each implementation. However, the IDS RAM specifies an Information Model with three levels of formalization: Conceptual Representation, Declarative Representation, and Programmatic Representation [8, 33]. These levels are illustrated in Figure 2.4.

The Conceptual Representation provides a high-level overview of the core, largely invariant concepts of the Information Model without binding them to specific technologies or domains [33].

The Declarative Representation offers a formal, machine-readable version of the Information Model, based on constraints from the Conceptual Representation. It uses existing standards and vocabularies, supporting various domains and applications [33].

Figure 2.4: IDS information model [8]




The Programmatic Representation is designed for software developers, aimed at integrating the Information Model with programming languages. It maps the IDS vocabulary onto code structures, promoting type safety and quality assurance [33].

### Process Layer

The Process Layer specifies the interactions taking place between the different components of the Data Space, thereby providing a dynamic view of the IDS RAM [8].

The **Onboarding** process defines the necessary steps an organization must follow to gain access to a Data Space. It begins with a certification phase, during which the organization's trustworthiness is assessed [8].

Once certified, the organization receives a digital identity and credentials from the Identity Provider. The organization must then obtain an IDS Connector, either by requesting a fully-fledged instance or by developing one internally [8].

The Connector itself must also undergo a certification process to ensure adequate levels of security and interoperability. Once certified, it must be properly configured with the new participant's identity and credentials, allowing other partners to verify this information [8].

Finally, the Connector must be made accessible within the Data Space, so that other participants can interact with it [8].

The **Publishing and Using Data Apps** process describes how to interact with an IDS App Store and how to use IDS Data Apps. Considering the dissertation's objectives, this process falls outside the scope of this analysis and will not be covered in detail.

This layer also includes three processes directly related to the Connector functionalities:

- **Data Offering:** How to publish data offers.
- **Contract Negotiation:** How to accept offers by negotiating usage policies.
- **Data Exchange:** How data is transferred between IDS participants.

These processes will be described in more detail in a following section, illustrating them in the context of a real Connector implementation.

### 2.2.3 Connector

The Connector is the core technical component of IDS, serving as the foundation for a standardized and interoperable data ecosystem [8, 35].

The collection of Connectors is what composes and shapes the Data Space, forming a distributed network of participants. This decentralized architecture eliminates the need for a central platform to handle data storage or routing [8, 35, 36].

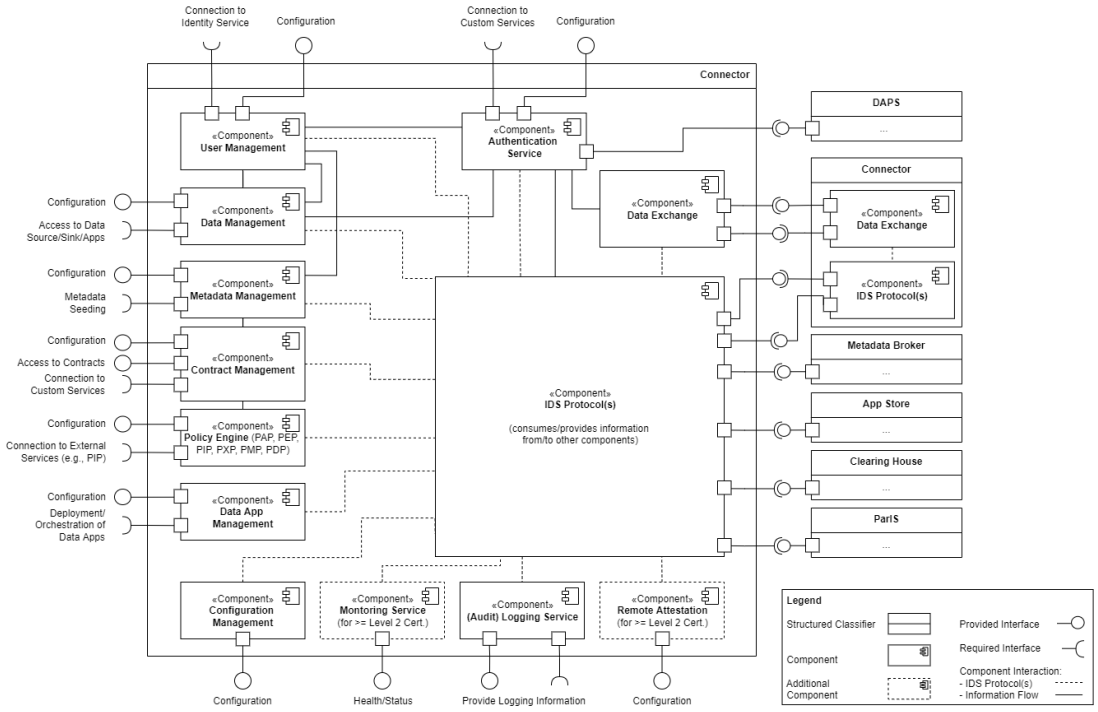
However, for Connectors of a Data Space to communicate with each other in this decentralized architecture, a well-defined communication interface is essential. This role is fulfilled by the DSP, a shared communications protocol that defines the standards, terminology, and specifications for IDS processes, ensuring consistent interactions between Connectors. By relying on the DSP, each organization can independently develop and operate its own Connector while remaining fully compatible with the Data Space [13].

The DSP does not explicitly mandate a specific transport protocol, allowing for implementations across a range of different web-based technologies. However, the specification does provide bindings for HTTPS, detailing how communication should be conducted over this protocol [13].

The concepts of finite and non-finite data are introduced by the DSP in the context of Data Spaces. Finite data refers to data that is defined by a finite set, such as files or images. In contrast, non-finite data is data that is defined by an infinite set or has no specified end, such as streams or data from APIs. Connectors must support both finite and non-finite data transfers [13].

While the critical role of Connectors in IDS is evident, setting up this component is a complex task. Implementations must fulfill all functionalities outlined in Figure 2.5 [8].

Figure 2.5: Connector functionalities [8]



The Fraunhofer Institute for Software and Systems Engineering was one of the first organizations to contribute to the development of Data Space components, pioneering the creation of the Dataspace Connector (DSC), a Java IDS Connector implementation based on REST APIs. This project was later adopted by the IDSA, aiming to facilitating the adoption of IDS [36, 38].

### 2.2.4 EDC

In 2020, eight organizations launched the EDC project under the Eclipse Foundation, a Java open-source framework for the development of IDS components in accordance with DSP standards [35].

As a framework, EDC is not a prepackaged system or application. Instead, it provides the building blocks for downstream projects to build shippable, installable distributions of their components [21].

Since different Data Spaces and different participants have distinct requirements, developing single, universal implementations of each of the IDS components is a challenging task. To address the diverse needs of the community, the EDC project is designed as a modular platform with an extension system. This architecture allows downstream projects to plug in only the extensions they require, thereby tailoring their distributions accordingly [21].

Components (the "C" in EDC) are logical software units that perform specific tasks. At a minimum, a component is composed by a runtime which is able to spin up, but does nothing more. Functionality is introduced by combining various extensions [21].

The modular design of the EDC supports many deployment topologies for these components, allowing them to be deployed as separate services or bundled together within a single process [21].

At the core of the EDC project is the EDC Connector, a pair of components that collectively form an IDS Connector implementation. The EDC Connector was developed as the successor to the DSC. Designed for interoperability across multiple Data Spaces, it has become one of the leading Connector initiatives to date. A notable example is the Catena-X Data Space, where EDC Connectors are prominently being used [25, 34, 35, 38].

The Control Plane is the first component that composes an EDC Connector. This component is responsible for assembling catalogs, creating contract agreements that grant access to data, managing data transfers, and monitor usage policy compliance [21].

The Data Plane is the second component of an EDC Connector. It is responsible for transmitting data via a wire protocol, operating under the direction of the Control Plane. Implementations can vary widely, ranging from lightweight serverless functions to fully-fledged data streaming platforms [21].

A single Control Plane may manage multiple Data Planes, each specialized for specific data types or wire protocols. While it is possible to bundle the Control Plane and Data Planes into a single deployed process, typical production environments deploy them as separate instances so that they can be operated and scaled independently [21].

Control Planes and Data Planes are dynamically associated. At startup, a Data Plane registers itself in a Control Plane, providing metadata about its capabilities such as supported protocols and transfer types [21].

### Interfaces and Communication

The EDC provides an HTTP-based implementation of the DSP, following the protocol bindings defined in its specification. With this implementation, the Control Plane exposes the DSP API, a public-facing interface through which external Connectors interact with a participant's Connector. This API makes the Connector available within the Data Space [21].

The Control Plane also exposes the Management API, a RESTful interface that enables client applications to interact with the EDC Connector. Through this interface, a Data Space participant can perform all management operations. Due to its scope, this API should only be accessible by the operating participant [21].

The EDC Connector exposes two additional APIs to establish communication between the Control Plane and the Data Plane. These interfaces are not intended to be accessed by either participant, and should therefore be restricted to the cluster hosting the EDC Connector [21].

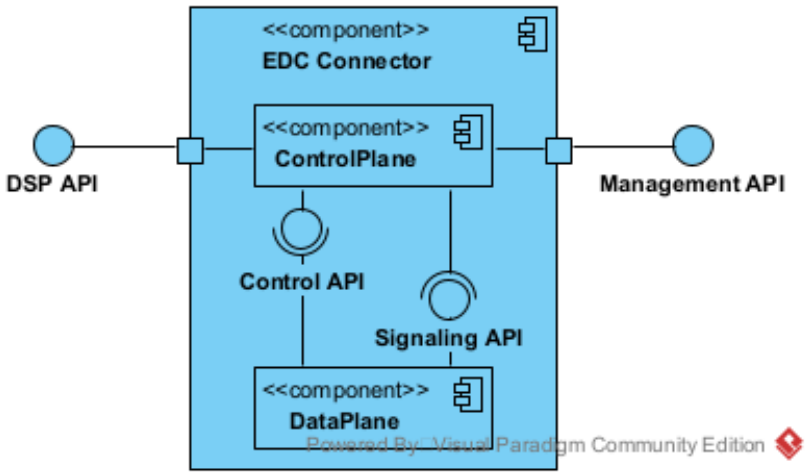
On the Control Plane side there is the Control API, a RESTful interface that enables the Data Plane to notify the Control Plane of specific events, such as the completion of a data transfer. It is also used for Data Planes to register themselves on a Control Plane [21].

On the Data Plane side there is the Data Plane Signaling API, another RESTful interface that allows the Control Plane to instruct the Data Plane of certain operations it must perform, such as initiating a data transfer.

Additionally, each Data Plane implementation may define its own APIs to enable the data transfer over the wire protocols it supports. Due the wide variety of possible Data Plane implementations, no mandatory protocol is imposed [21].

Figure 2.6 illustrates the EDC components and their APIs. Given the extensibility of the EDC, downstream projects can easily implement extensions that bring new interfaces to support custom features [21].

Figure 2.6: EDC components and interfaces, adapted from [21]



### Data Offering

As the EDC Connector follows the IDS RAM, it must implement the processes specified in the process layer. Understanding how these processes are handled requires examining the EDC entities in detail.

Everything starts with an **Asset**, the primary building block for data sharing. An Asset is a generic representation of data that a Data Provider may exchange, and is not bound to any specific format or data source type. Assets are not the data itself but rather descriptors containing metadata about it. One of these descriptors is the Data Address, a reference for the physical location where the data resides. To ensure interoperability and describe the data effectively, Assets should leverage Data Space vocabularies [8, 21, 39].

After registering Assets, Data Suppliers must specify **Policies**. A Policy defines a set of duties, rights and obligations, and can be used both to control access to data and to specify permitted data usage. In the EDC, Policies are expressed with Open Digital Rights Language (ODRL). Data Providers manage and persist Policies through **Policy Definitions**, which assign an identifier to a Policy. This separation offers great flexibility, decoupling the enforcement of Policies from their management [21].

A **Contract Definition** declares which Policies apply to a set of Assets, thereby linking these entities. It is the basis for a Provider to expose data to Consumers. Each Contract Definition must specify an Access Policy, a Contract Policy, and an Asset Selector. The Access Policy determines who can access an Asset by evaluating the conditions that Data Consumers must meet. The Contract Policy dictates the requirements that a Data Consumer must fulfill and what rights it obtains over an Asset. The Asset Selector is a query-like statement that selects a set of Assets by their attributes, such as filtering by the Asset ID or a specific property [21].

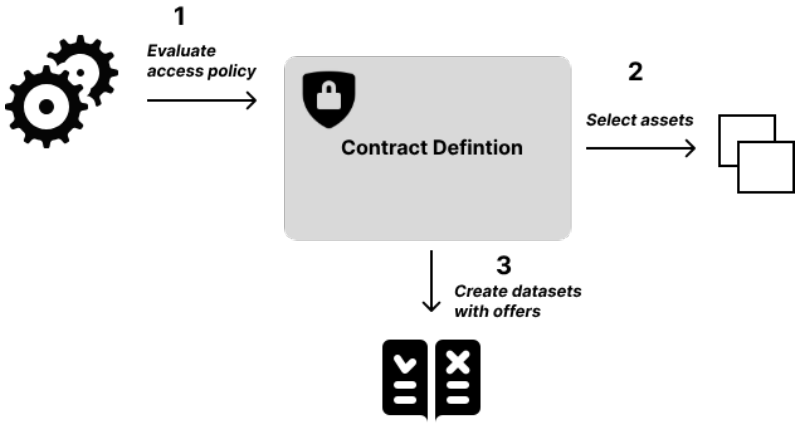
Executing the Asset Selector query yields a set of Assets. Each Asset, when combined with the Contract Policy, creates a **Contract Offer** (or simply Offer), which represents a dataset that can be negotiated by a Consumer and the conditions to do so. If two Contract Definitions include the same Asset, two distinct Offers are created, each referencing its respective Contract Policy. Conversely, if no Assets match the selector, no Offers are generated [21].

Unlike other EDC entities, Offers are not persisted in the Connector. They are dynamically generated at runtime when a Consumer requests a Provider's **Catalog**. A Catalog is a DSP concept that contains a set of Offers available to the requesting Consumer. To build it, the Connector first evaluates the Access Policy of each registered Contract Definition against the Consumer, filtering out those for which access is not granted. Then, it generates the Offers from the remaining Contract Definitions. Figure 2.7 displays the catalog generation process [21].

### Contract Negotiation

Once a Consumer receives a Catalog, it may solicit access to an Asset by sending a negotiation request for a specific Offer. This starts a **Contract Negotiation**, an asynchronous, stateful, (semi-)automated process that represents the request to access a dataset. Throughout its life cycle, a Contract Negotiation progresses through a state machine, where either participant is able to advance its state [8, 21, 27, 39].

Figure 2.7: Catalog generation [21]




Both the Consumer and the Provider maintain a correlated Contract Negotiation instance within their Connectors. Consistency between the two is ensured through an asynchronous messaging system. The EDC provides an implementation of this system that emphasizes reliability, guaranteeing that no messages are lost and that state transitions are acknowledge by both participant Connectors [21].

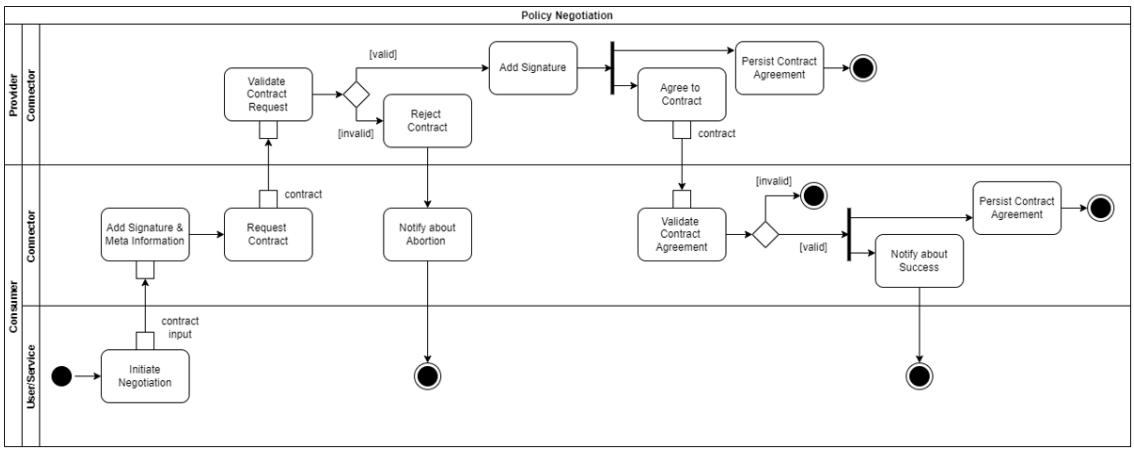
If a Contract Negotiation is successfully completed, a **Contract Agreement** is created and sent to the Consumer. A Contract Agreement is a binding digital contract that grants the Consumer the right to access an Asset under the conditions specified by the Contract Policy. In case of contract infringement, legal action may apply. Both the Provider and the Consumer retain a copy of the Agreement within their respective Connectors [8, 21, 27, 39].

Figure 2.8 presents a simplified representation of this process. Variations may occur depending on the use case, having the possibility to log the operation to the Clearing House, reverse the sequence to be initiated by the Data Provider, or allow counter offers [8].

### Data Exchange

After a Contract Agreement is established between two participants, the Data Consumer can initiate a **Transfer Process**, an asynchronous, stateful, (semi-)automated procedure

Figure 2.8: "Contract Negotiation" process [8]



that oversees a data transfer. Similarly to the Contract Negotiation, Transfer Processes also progress through a state machine over their life cycle, maintain correlated instances in both the Provider and Consumer Connectors, and rely on an asynchronous messaging system to ensure consistency [21].

A Transfer Process consists of two phases: the control phase and the transfer phase. The control phase runs first, during which the transfer request is initiated and necessary resources are provisioned. This phase is executed between the Provider and Consumer Control Planes. The transfer phase takes place afterwards, and leverages the Provider Data Plane to exchange data between the participants [8, 39].

Transfer Processes can be finite or non-finite according to the finiteness of the data being exchanged. Finite Transfer Processes complete after all data has been transferred. Non-finite Transfer Processes do not complete because non-finite data has no predefined completion point, and therefore will continue until manually terminated by either the Consumer or the Provider [13, 21].

Additionally, Transfer Processes support two data transfer flows, determined by which participant initiates the data send operation: **Consumer Pull** or **Provider Push**. The steps performed during the control and transfer phases depend on the chosen transfer flow [8, 21].

Consumer Pull (or simply Pull) data transfers allow the Data Consumer to request the agreed data. After initiating the Transfer Process, the Consumer receives information on how to retrieve data from the Provider as an endpoint. This endpoint may route directly to the exact location where the data is stored or act as a proxy to avoid exposing internal resources. Pull transfers are synchronous, ensuring immediate data delivery [8, 21].

A common example of a Consumer Pull transfer is when an application produces data and exposes it through an HTTP API. During the Transfer Process, the corresponding API endpoint is made available to the Consumer, allowing it to request the data directly [21].

Provider Push (or simply Push) data transfers place the responsibility of sending the agreed data on the Data Provider. In this flow, the Provider Data Plane retrieves the data from its source and transmits it to a destination specified by the Consumer. The process is asynchronous, meaning that the data is not delivered immediately after the transfer is started, but will eventually arrive at the designated destination [8, 21].

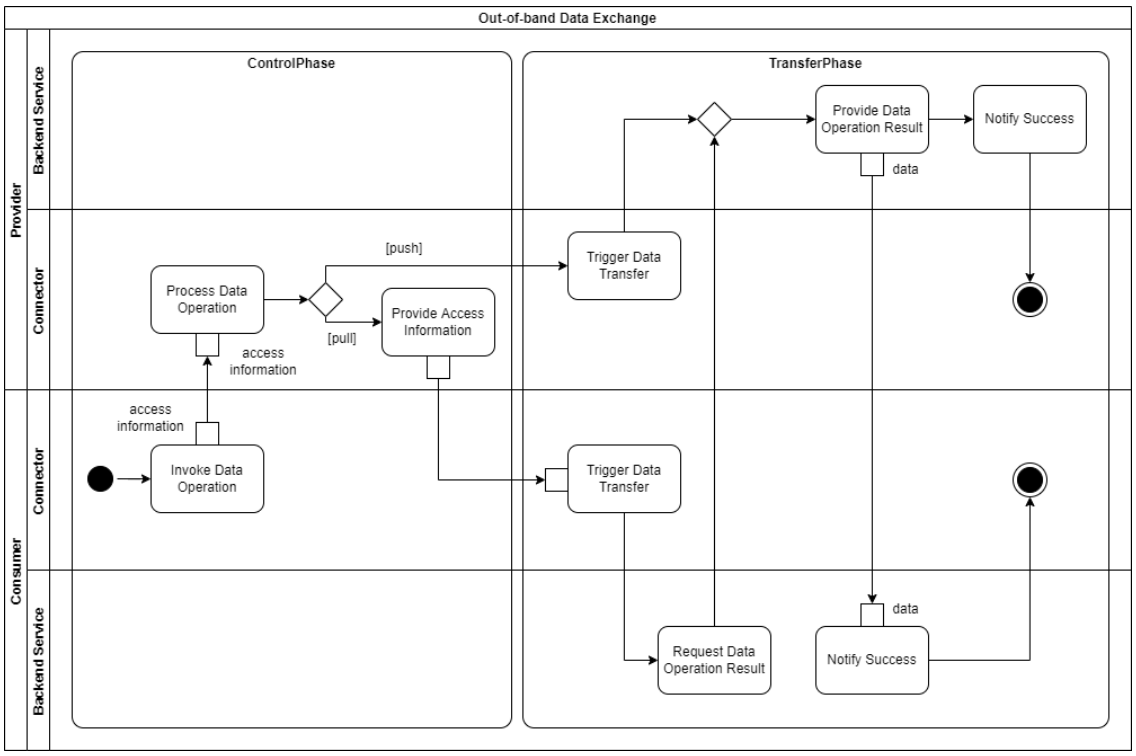
An example of this flow is when the Provider has a file at a file storage system and the Consumer wants to receive this file in its own file storage system. After initiating the Transfer Process, the file is transferred to the Consumer's data destination without any further interaction on their part [21].

Figure 2.9 illustrates this process. Note that the Data Consumer is always the participant that initiates the Transfer Process, regardless of the chosen transfer flow [8, 25].

### Data Plane Framework

The EDC includes a framework for building custom Data Planes called the Data Plane Framework (DPF). This framework supports end-to-end streaming of data, Pull and Push transfers, and various extensibility points, making it a good starting point for both general-purpose and specialized Data Planes [21].

Figure 2.9: "Data Exchange" process [8]




The DPF models data transfers as **Data Flows**, the Data Plane counterpart of a Transfer Process. A Data Flow contains all information required to process the exchange [11].

Data Planes built with the DPF fulfill Data Flow requests through a **Transfer Service**, an extension point that defines how transfers are executed. Multiple implementations can coexist, enabling a single Data Plane to support different types of transfers. The **Pipeline Service** is an additional extension point that expands the capabilities of the Transfer Service by enabling transfers across different technologies [21].

In the DPF, the location where data resides is called the **Data Source**. Since different technologies require different mechanisms to retrieve data, each technology has its own Data Source implementation. For example, the DPF provides an HTTP Data Source for pulling data exposed through an HTTP API. Custom Data Source implementations can be added as extensions [21].

Conversely, the **Data Sink** represents the location where data is delivered. As with sources, different technologies require different mechanisms to send data, and therefore each technology has its own Data Sink implementation. Custom Data Sink implementations can likewise be introduced as extensions [21].

The Pipeline Service allows the registration of Data Sources and Data Sinks, making use of their implementations to perform the data transfer. This effectively integrates any Data Source technology to any Data Sink technology, supporting the interoperability aspect of Data Spaces. For example, a dataset available via an HTTP Data Source can be transformed into a file and delivered to a file storage Data Sink. The EDC provides a default Pipeline Service implementation that streams data directly from the Data Source to the Data Sink [21].

## 2.3 Analysis

Data Spaces are already being used across a wide range of industries, supported by federated entities such as the European Commission and enterprise consortia like Catena-X.

Although smaller organizations face challenges in adopting Data Spaces, the concept has been steadily gaining traction, particularly in Europe.

Moreover, initiatives like the IDS have introduced an additional layer of standardization, facilitating the adoption of these technologies.

The IDS RAM establishes a set of constraints and guidelines that IDS implementations must comply with. It provides detailed specifications encompassing roles, components, functional requirements, information models and processes, ensuring the core features of Data Spaces are consistently upheld.

The Connector serves as the beating heart of Data Spaces, playing a pivotal role in their operations. Each participant manages its own Connector, creating a decentralized architecture that shapes the Data Space.

The DSP is a communication protocol that standardizes interactions between Connectors. It defines terminology, processes, and message formats to ensure consistent and reliable operation between Connectors. Additionally, the DSP introduces the notions of finite and non-finite data, concepts that are crucial to the scope of this dissertation.

OSS projects like the EDC are leading the development of IDS Connectors, promoting collaboration and further encouraging adoption by organizations within IDS.

EDC provides the tools to build EDC Connectors, Connector implementations built on top of a modular extension system. These Connectors are already being used across various IDS, with the most notable contribution being in the Catena-X Data Space.

An EDC Connector consists of two components: the Control Plane, responsible for creating data offers, negotiating contracts and managing data transfers, and the Data Plane, responsible for sending data over a wire protocol.

Communication with an EDC Connector occurs through the APIs it exposes. Table 2.7 summarizes the minimum APIs that an EDC Connector should provide, along with their expected access level.

Table 2.7: EDC APIs summary

| API | Purpose | Access Level |
|---|---|---|
| DSP | Allow interactions from external Connectors over DSP | Public |
| Management | Allow a participant to operate its own Connector | Internal |
| Control | Communication from a Data Plane to its Control Plane | Private |
| Signaling | Communication from the Control Plane to a Data Plane | Private |

Public = external access; Internal = organizational access; Private = cluster access

The Data Offering process in the EDC requires a Data Provider to register Assets, Policy Definitions and Contract Definitions to securely expose Contract Offers. Then, a Data Consumer can request a Catalog from the Provider, obtaining the list of offers available for negotiation.

The Contract Negotiation process occurs when a Consumer negotiates a Provider's Offer. This is an asynchronous process that takes place between their Control Planes. Once complete, a Contract Agreement is generated, granting the Consumer rights to access and use a dataset under certain conditions.

The Data Exchange process takes place after a Contract Agreement has been established, when the Consumer initiates a Transfer Process. This is an asynchronous process that involves both the Provider and Consumer Control Planes, and the Provider Data Plane.

Transfer Processes can be finite or non-finite depending on the finiteness of the data being exchanged. Finite Transfer Processes complete once all data has been transferred, and non-finite Transfer Processes remain active until manually terminated.

There are two modes to transfer data: Consumer Pull and Provider Push. In Pull transfers, the Consumer receives information on how to obtain the agreed data, and then retrieves it synchronously. In Push transfers, the Consumer defines a destination where data should be sent to, and the Provider is responsible to asynchronously deliver it.

Through the analysis of these documents, the student gained a deeper understanding of the context surrounding the research question. Considering the objectives of this dissertation, the main takeaway from this investigation is that the EDC Connector must support non-finite data transfers via the Provider Push flow by keeping the corresponding Transfer Process in an active state.

# Chapter 3

# Problem Analysis & Requirements

This chapter presents the elicitation process conducted to analyze the problem and design a solution. To do so, a series of interviews were conducted with individuals directly involved in the use and development of the EDC Connector.

These conversations were crucial in shaping the direction of the work, offering valuable insights into the real-world context in which the Connector operates. By uncovering user needs, pain points, and expectations, the interviews helped define the feature requirements.

A brief discussion of the results is also included, establishing the main guidelines for the design of the solution.

## 3.1 Methodology

The conducted interviews followed predefined guides designed to be concise, direct, and goal-oriented. Each guide includes a clear goal, a list of prepared questions and potential follow-ups.

Interview guides were structured into logical sections, each focused on addressing a specific goal and containing a subset of questions. All guides included an introductory section for interviewee profiling and a closing section for final remarks. The content of the intermediate sections varied based on the specific goals of each guide.

To ensure that each question could be easily identified and referenced, a structured ID system was implemented. Numeric identifiers were assigned to guides, sections, and questions, resulting in fully qualified question IDs in the format <GuideID>-<SectionID>-<QuestionID>. For example, question 2 in section 1 of guide 3 was labeled 3-1-2. Planned follow-up questions extended the parent question's ID using decimal notation (e.g., 2.1, 2.2).

Interviews were conducted remotely through online meetings and recorded for future reference, with appropriate consent. Although in-person meetings were preferable, location constraints made remote sessions the more practical choice, while still allowing for natural discussion to flow.

To protect the privacy of interviewees, identifiable information was kept to a minimum. For example, interviewees were labeled using generic capital letters (e.g., A, B), and only role-related questions were included in the profiling sections. These measures were agreed upon with every participant.

Additionally, any sensitive information discussed during the interviews, such as business strategy, was omitted from this dissertation. All presented information is either publicly

available or modeled into generic scenarios. While these restrictions limit the depth of insight that can be shared, the conclusions drawn nonetheless accurately reflect the real findings.

## 3.2 Interviewees

Interviewees were selected from a single organization participating in the Catena-X Data Space. Although the study focused on one organization, participants were carefully chosen to represent a range of roles and domains within the company. This approach ensured coverage of diverse business use cases and different ways of using the EDC Connector.

All selected interviewees had over a year of hands-on experience with Data Spaces and relied on them regularly in their daily work. Three interviewees participated in the study, labeled as Interviewee A, B, and C.

Interviewee A is an IT use case lead specialized in sustainability and circular economy. They have been involved in the Catena-X Data Space for four years, although initially as a systems architect before taking on their current role. Their main responsibilities are to promote awareness of Data Spaces and their benefits within business departments, as well as gathering business requirements and translating them into technical requirements.

Interviewee B is also an IT use case lead, primarily focused on traceability and quality. They have been working with Data Spaces for almost five years. Similarly to interviewee A, their main responsibilities include contacting business departments to gather requirements, translating them into technical specifications. In addition, they participate in meetings with the Catena-X consortium.

Interviewee C is the sub-product owner of the organization's feature team responsible for developing Data Space components, and has been working in this domain for four years. They oversee the team's work, which includes maintaining the organization's Connector and contributing to the EDC project. They also take part in open-source discussions, influencing features and architectural decisions.

## 3.3 Use Case Analysis

To explore the use cases managed by interviewees A and B, an interview guide tailored to IT use case leads was created.

This guide has two primary objectives: to understand how the proposed feature impacts existing business use cases and to gather expectations on how non-finite data should be handled by the EDC Connector. Note that the specific details of each use case are not the focus here, but more so their interaction with the Connector.

The guide contains two intermediate sections. The first section explores the current experience with the EDC Connector, and the second section focuses on collecting expectations regarding the proposed feature. Table 3.1 presents the structure of this interview guide.

### 3.3.1 Results

To ensure relevance, the analyzed use cases needed to revolve around non-finite data, as this is a crucial part of the problem that the proposed feature aims to address. Interestingly,

Table 3.1: IT use case lead interview guide

| ID | Question |
|---|---|
| 1-1-1 | What is your role within the organization? |
| 1-1-2 | How long have you been working with Data Spaces? |
| 1-1-3 | What are your main responsibilities related to Data Spaces? |
| 1-2-1 | Does your use case primarily act as a Data Provider, a Data Consumer, or both? |
| 1-2-2 | Does your use case mainly perform Provider Push or Consumer Pull transfers? |
| 1-2-3 | Does your use case involve non-finite data? |
| 1-2-3.1 | If yes, what is its relation to the Connector? |
| 1-3-1 | Would a mechanism to transfer non-finite data asynchronously be useful in your use case? |
| 1-3-1.1 | If yes, what behavior would you expect from such a mechanism? |
| 1-3-1.2 | If yes, how could this feature improve your use case (e.g., cost, time, simplicity)? |
| 1-4-1 | Is there anything else you would like to add? |

all analyzed use cases involve non-finite data, which appears to be more common than finite data in real-world inter-organizational scenarios. In total, five use cases were identified.

Additionally, all use cases work with an EDC Connector composed by a general-purpose Data Plane built with the default services of the DPF.

Some use cases use the Consumer Pull flow to transfer non-finite data. After receiving information on how to access the Data Source, the Consumer can repeatedly request its endpoint, polling for new data as it becomes available.

Providers can enforce authentication at the Data Source endpoint to increase security. If so, Consumers are granted temporary access to this endpoint. To maintain valid access after the expiry date, refresh mechanisms are in place.

The existing Provider Push mechanism does not allow non-finite data transfers. Once the currently available data is transmitted, the Transfer Process completes. This behavior does not reflect that the transfer is ongoing and that new data may become available.

By examining how these use cases interact with the EDC Connector, the student identified "workarounds" that allow Data Providers to push non-finite data to Data Consumers.

#### Transfer Data Chunks Individually

To create the illusion that non-finite data is actually finite, each data chunk may be treated individually and transferred using the existing Provider Push mechanism.

The issue with this approach is that the Consumer must request each chunk every time new data is available, creating a new Transfer Process. This increases the overall complexity, as there is a recurrent step which must always be performed to ensure the correct data exchange, and also adds computing costs, as each new Transfer Process must process multiple states throughout its life cycle.

Moreover, requiring the Consumer to request each data chunk resembles the Consumer Pull, thereby losing the essence and benefits of the Provider Push.

#### Invert The Participants' Roles

Non-finite data transfers are also possible by reversing the participants' roles and leveraging the Consumer Pull capabilities. To illustrate this, consider the following example: Company A wants to receive non-finite data from Company B, making Company A the Data Customer and Company B the Data Supplier.

From the Connector's perspective, lets treat Company A as the Data Provider and Company B as the Data Consumer, despite Company B holding the desired data. This effectively flips the roles expected in a typical data transfer scenario. Here's how the process works:

1. Company A creates an Asset pointing to where the data will be stored and offers it.
2. Company B negotiates Company A's Offer, reaching a Contract Agreement.
3. Company B starts a Pull transfer, receiving information of an endpoint to company A's Data Plane.
4. Company B requests this endpoint, embedding the desired data in the request payload.
5. Company A's Data Plane receives the data and forwards it to Data address specified in the Asset, allowing it to be processed and stored.

While this approach technically works, it is considered an anti-pattern and should be avoided. Inverting the participants roles results in the Contract Policy being enforced on Company B instead of Company A, which is illogical because Company B holds the data.

Moreover, credential validation is also inverted, meaning that the Data Supplier (Company B) no longer verifies the Data Customer's (Company A's) credentials when the Transfer Process is initiated.

In summary, this setup undermines proper Policy enforcement and violates the Data Space's trust principle.

## 3.4 Technical Requirements

A second interview guide was created, now targeting product owners. This guide was used to interview participant C, and it has a single objective: to gather technical requirements that will shape the development of the proposed functionality.

The guide contains three intermediate sections. The first gathers technical requirements for initiating a Transfer Process that supports non-finite data. The second focuses on how the data transfer for non-finite data should be handled. The third covers requirements for terminating the Transfer Process. Table 3.2 presents the structure of this interview guide.

Table 3.2: Product owner interview guide

| ID | Question |
|---|---|
| 2-1-1 | What is your role within the organization? |
| 2-1-2 | How long have you been working with Data Spaces? |
| 2-1-3 | What are your main responsibilities related to Data Spaces? |
| 2-2-1 | What is your vision for the creation of Transfer Processes that support non-finite data? |
| 2-2-2 | Can the current Transfer Process creation mechanism be adapted for this purpose? |
| 2-2-2.1 | If no, why? |
| 2-2-2.2 | If no, should a completely separate mechanism be developed? |
| 2-2-3 | How should Transfer Processes for non-finite data be differentiated from those handling finite data? |
| 2-3-1 | What is your vision for how new data in a non-finite data source should be transferred to Consumers? |
| 2-3-2 | Should this process be automated? |
| 2-3-2.1 | If yes, is there any benefit in starting with a manual mechanism first? |
| 2-3-3 | Should data source updates be reflected to a single participant, or to multiple participants? |
| 2-4-1 | What is your vision for terminating Transfer Processes that handle non-finite data? |
| 2-4-2 | Is the current termination mechanism suitable for non-finite data scenarios? |
| 2-4-3 | Who should be able to terminate such Transfer Processes? |
| 2-5-1 | Is there anything else you would like to add? |

### 3.4.1 Results

Functional requirements specify the basic facilities that the system must satisfy to successfully implement the feature. Table 3.3 lists the identified functional requirements.

Table 3.3: Functional requirements

| ID | Requirement |
|---|---|
| FR1 | Consumers must be able to create Transfer Processes that allow multiple data transfers. |
| FR2 | Providers must be able to manually trigger the transfer of data to a single partner. |
| FR3 | Providers must be able to manually trigger the transfer of data by dispatching an event. |
| FR4 | Participants should be able to terminate Transfer Processes on demand, closing the communications channel. |

The development process should be guided by these requirements to ensure that the implemented solution fulfills the needs of the participants.

Non-Functional requirements are quality constraints that the system must comply to function correctly. Table 3.4 presents the identified non-functional requirements.

Table 3.4: Non-functional requirements

| ID | Requirement |
|---|---|
| NFR1 | The system should integrate and reuse existing Provider Push concepts where applicable. |
| NFR2 | The system should integrate and reuse Consumer Pull concepts where applicable. |

These non-functional requirements emphasize maintaining consistency with existing processes to promote maintainability and modularity, two core design principles of the EDC.

## 3.5 Discussion

The interviews successfully met the objectives defined in each guide, yielding many valuable insights.

Interviewees A and B had their business use cases analyzed, helping to understand their specific needs and how these map to EDC Connector capabilities.

Interviewee C contributed from a technical perspective, helping identifying both functional and non-functional requirements to guide the development of the proposed feature.

In the state-of-the-art analysis it was concluded that the EDC Connector must support non-finite data transfers via the Provider Push flow. However, these interviews revealed that such transfers are currently only possible through workarounds that are problematic and should be avoided.

Relying on Provider Push transfers avoids the use of the Consumer Pull flow, therefore preventing the workaround of inverting the participants' roles. Furthermore, as the problem is specific to non-finite data, the solution requires the use of non-finite Transfer Processes, which prevents the workaround of transferring individual chunks.

To create non-finite Transfer Processes it is essential to distinguish between finite and non-finite data. Currently, this is not supported, thereby the proposed solution must address this gap.

According to the requirements, Providers must be able to trigger data transfers on demand to a single partner by dispatching an event. The proposed solution should also include a triggering mechanism to support this.

Finally, it must be possible to terminate non-finite Transfer Processes, allowing the communication channel to be closed. The EDC already provides a mechanism to do so via the Management API, which is capable of terminating any type of Transfer Process. Therefore, no additional developments are needed to meet this requirement.

# Chapter 4

# Development

This chapter describes the development process of the proposed feature, covering the design derived from the business analysis and its subsequent implementation. It also outlines the decisions made and the impediments encountered, providing a clear view of this phase.

The chapter begins by presenting the methodology followed during this phase. Then, it moves on to details the contributions made, followed by a description of the quality assurance mechanisms set in place. This chapter ends by outlining the work carried out to deploy an EDC Connector.

## 4.1 Methodology

The development process of the proposed feature followed the guidelines and processes established by the EDC project, while also allowing space for personal planning and adaptation. This sections describes the methodologies applied during each phase.

### 4.1.1 Contribution Process

As is common in OSS projects, the EDC project enforces strict contribution guidelines. Contributors must follow the established process from the moment they propose a feature until its eventual implementation.

Specifically for the EDC, contributors must also adhere to the Eclipse Community Code of Conduct, ensuring an open and welcoming environment to the community, and sign the Eclipse Contributor Agreement, which defines the legal terms of their contributions.

For technical topics, the first step is to open a discussion in the appropriate repository. This discussion should present the new idea to the community, along with a preliminary design.

Community members interested in the feature can participate in the discussion to help shape the requirements and overall design. Committers may also provide feedback on the proposal, assessing its feasibility or indicating if the feature is not intended for the project.

Once the idea gains some traction, and consensus on an approach is achieved within the community, a decision record can be created. A decision record is a document that persists a decision made in the project, detailing its rationale and a comprehensive plan on how to tackle the implementation of the new feature.

The decision record must be reviewed by the committers to ensure it accurately reflects the outcomes of the discussion. This is done by opening a pull request in the appropriate repository. Once the pull request is approved and merged, implementation can begin.

The implementation process includes writing code and automated tests following the approach described in the decision record. Once ready for review, another pull request must be opened with the proposed changes. Committers may then review the code to ensure it aligns with the approach defined in the decision record. Once approved and merged, the contribution is finally complete.

### 4.1.2 Project Planning

The EDC follows a predictive, Waterfall-like planning methodology [40], with release cycles of three months. Each release contains well defined phases for planning, implementation, testing, review, etc. Contributions developed during a release cycle will be available when the release is complete.

As an aspiring contributor, the student adhered to the planned timeline for the current release, discussing the issue during the planning phase and implementing the contribution during the implementation phase.

Throughout the implementation phase, the student adopted an adaptive methodology using the Scrum framework [41]. The feature was divided into smaller units of work, tackled in two-week development iterations named sprints. Each sprint focused on achieving a specific goal, leading to the creation, review, and merging of at least one pull request per iteration. Only two development cycles were required.

By combining the predictive methodology imposed by the EDC project with the adaptive methodology adopted by the student, a hybrid methodology [42] was used to plan and develop the feature.

The student also applied DevOps practices [43] throughout development, employing techniques such as automating build and test processes, provisioning infrastructure with Infrastructure as Code tools, and deploying through Continuous Integration (CI)/Continuous Delivery (CD) pipelines. Since these practices are generically applicable and do not directly contribute to the unique aspects of this work, their detailed implementation falls outside the scope of the dissertation.

## 4.2 Contributions

According with the contribution guidelines, a discussion was opened in the EDC Connector repository, presenting a feature proposal. This proposal explained the problem it aimed to address and presented an initial design.

However, the proposal was not well received by the EDC committers due to its high complexity and uncertainty. In addition, the existence of workarounds and the possibility to develop internal extensions to enable Provider Push non-finite data transfers ultimately led to the committers rejecting the idea.

Since the dissertation's objectives include contributing the feature to an OSS project, allowing other Data Space participants to benefit from it, the student explored alternative projects where the contributions could be made. This analysis led to the selection of Tractus-X.

Tractus-X is an open-source downstream project that provides runnable distributions of EDC Connectors to participants of the Catena-X Data Space. In practice, it distributes

Control Plane and Data Plane executable artifacts that bundle EDC core services, upstream extensions and custom extensions developed at Tractus-X.

Being also supported by the Eclipse Foundation, the Tractus-X project follows similar contribution guidelines and processes. This enabled a smooth transition to the project without requiring major changes to the dissertation's plan.

However, there is one notable difference that affected development. Being a downstream project, upstream classes are unchangeable, meaning that higher level logic cannot be altered and that implementation is restricted to the expected extensibility model.

The proposal was positively received in the Tractus-X project, generating an extensive discussion on how to approach the issue. After months of discussion, a final design was agreed upon and a decision record created.

### 4.2.1 Identify Non-finite Data

The first step to enable Provider-Push non-finite data transfers is to determine how non-finite data should be identified. It is important to note that non-finite data is not bound to any specific technology. For example:

- A message may represent a finite fragment of a potentially non-finite dataset available in a messaging queue;
- A JSON payload may represent a finite fragment of a potentially non-finite dataset exposed through an API endpoint;
- A file may represent a finite fragment of a potentially non-finite dataset provided by a file storage system;

This means that an identifier for non-finite data must be placed in a generic, technology-agnostic entity. In the EDC Connector, both the Transfer Process and the Asset were analyzed to decide which of these entities should hold this identifier.

Adding the identifier when initiating a Transfer Process is a plausible option because the finiteness of the data is only relevant during the transfer itself. In this context, the identifier would represent the finiteness of the Transfer Process, thereby reflecting the finiteness of the data.

However, this approach brings some issues. Since the Consumer is responsible for initiating Transfer Processes, they would also be the participant setting this identifier. As the data and its properties are responsibility of the Data Owner, it would be incorrect for the Consumer to assert the finiteness of the data being transferred.

As a result, the Asset was selected as the entity that represents the finiteness of the data. Since the Asset describes the data itself, adding the identifier to this entity is conceptually consistent. This way, the Provider can specify whether the data is finite or non-finite at the time of Asset registration.

To achieve this, a new extension point named `FinitenessEvaluator` was created for determining if a data transfer involves finite or non-finite data. As an extension point, it allows custom implementations on how this evaluation should be processed.

A default implementation was also provided as a new extension. This implementation verifies the value of a property named `"isNonFinite"` in the Asset's Data Address. Providers

can identify non-finite data by setting this property to "true", while any other values will consider the data to be finite. Listing 4.1 shows an example JSON payload that can be used to register an Asset that represents non-finite data.

```
{
  "@context": {},
  "@type": "Asset",
  "@id": "asset-id",
  "properties": {
    "description": "EDC Demo Asset"
  },
  "dataAddress": {
    "@type": "DataAddress",
    "isNonFinite": "true",
    "type": "HttpData",
    "baseUrl": "https://my.api.com"
  }
}
```

Listing 4.1: JSON payload to register an Asset

### 4.2.2 Non-Finite Provider Push Transfers

The next step is to determine how to prevent the Data Plane from completing the Data Flow, and consequentially the Transfer Process, after performing a Push transfer.

As seen in the state-of-the-art analysis, data transfers in the DPF are handled by a Transfer Service. More specifically, since this feature is intended to be compatible with any Data Source and Data Sink implementation, data transfers should be managed by a Pipeline Service. As an extension point, custom implementations can be provided to override the default one, making the Pipeline Service the most suitable place to adapt the transfer behavior and enable support for non-finite data.

The Pipeline Service exposes a `transfer` method that starts a data transfer. Implementations should execute the transfer asynchronously, returning a `CompletableFuture` that will eventually hold the transfer result.

In the default Pipeline Service implementation, the `transfer` method first creates a Data Source and a Data Sink instance synchronously, and then dispatches a task that pulls data from the Data Source and pushes it to the Data Sink asynchronously.

This task is actually handled by the Data Sink, which performs the transfer via streaming, returning a `CompletableFuture` with the transfer result, which in turn is also returned by the `transfer` method. As part of the defined non-functional requirements, the transfer details should continue to be delegated to the Data Sink.

When the `CompletableFuture` returned by the `transfer` method completes, whether due to a successful or failed transfer, an event handler executes, notifying the Control Plane that the transfer has finished and consequentially updating the state of the Transfer Process. This logic is implemented as core functionality, not as an extension point, and therefore cannot be overridden in a downstream project.

To avoid this behavior, the community agreed that the `CompletableFuture` returned by the `transfer` method should never complete if the transfer succeeds, preventing the event handler from executing and keeping the Transfer Process active.

On the other hand, if an error occurs during the data transfer, the Data Flow should terminate immediately with an error message, consequentially terminating the Transfer Process as well. This fail-fast approach ensures that the error is visible to participants. If the Data Plane recovered from a transfer error, no indication of this failure would be available other than the data not reaching the destination. Listing 4.2 illustrates a snippet of the `transfer` method from the new Pipeline Service implementation.

```
public CompletableFuture<StreamResult<Object>> transfer(DataFlow request
    ){
    var sink = sinkFactory.createSink(request);
    var source = sourceFactory.createSource(request);

    CompletableFuture futureResult = sink.transfer(source);

    if (!finitenessEvaluator.isNonFinite(request)) {
        return futureResult;
    }

    // Complete transfer only if transfer result failed for non-finite
    data
    var returnedFuture = new CompletableFuture<StreamResult<Object>>();
    futureResult.thenAccept(result -> {
        if (result.failed()) {
            returnedFuture.complete(result);
        }
    });
    return returnedFuture;
}
```

Listing 4.2: Snippet of the Pipeline Service

Note that this Pipeline Service implementation is capable of transferring both finite and non-finite data. While it would be possible to separate this logic, using the default implementation to transfer finite data and the new implementation for non-finite data, that would require participants to deploy two different Data Planes, as only one Pipeline Service implementation can be present in a runtime.

This decision simplifies the deployment process, allowing for a single Data Plane to be deployed that can handle both finite and non-finite data. Figure 4.1 represents the data transfer process using the new Pipeline Service, where preexisting behavior is shown in black and new functionality introduced by the new implementation is highlighted in red.

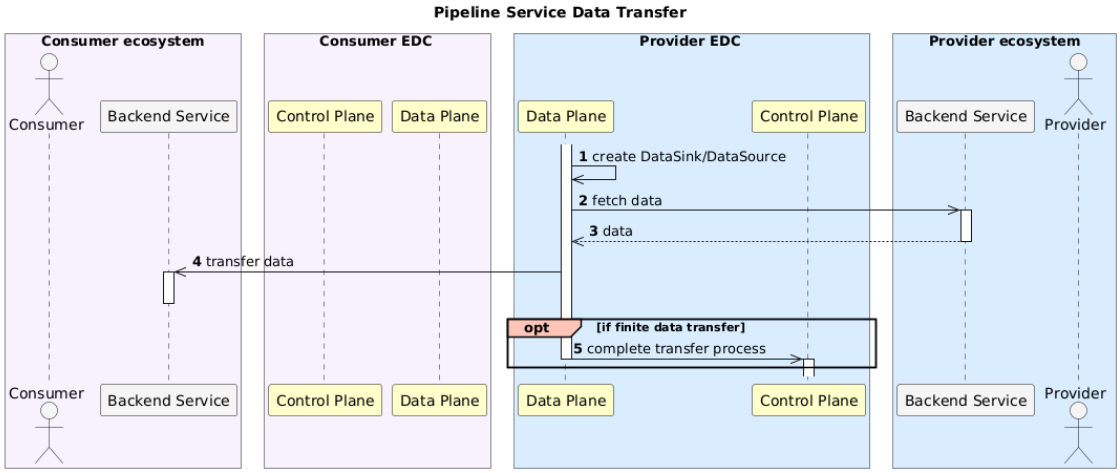
A new extension was developed to provide this new Pipeline Service implementation. This extension was introduced into the base Tractus-X runtime, ensuring that newly built EDC Connector images automatically ship with this feature.

### 4.2.3 Trigger Data Transfers

The previous extensions enable Provider Push non-finite data transfers by keeping the Transfer Process active. However, on their own, these extensions only allow one data chunk to be transferred, therefore not providing much use.

According to the functional requirements, Providers must be able to trigger new transfers on demand to continue delivering non-finite data. To address this, a triggering mechanism was introduced, allowing to reuse the active Transfer Processes and Data Flow. The triggering

Figure 4.1: Pipeline Service data transfer




mechanism consists in exposing an API endpoint on the Provider side, capable of receiving transfer requests and dispatching their execution.

There was a discussion in the community on where to host this new API endpoint. One option was to add it to the Control Plane, extending the Management API. Alternatively, it could be placed in the Data Plane, but since the Data Plane does not currently expose a Provider-side API, a new one would need to be introduced.

While adding this endpoint in the Control Plane would be simpler, as it would integrate naturally in the Transfer Process API context, the responsibility of receiving and executing transfer requests belongs to the Data Plane. In addition, it would also add unnecessary load on a component that should not be directly involved in the transfer itself.

The Data Plane was therefore chosen as the most appropriate place to add this new endpoint. The introduction of a new API was done by extending the scope of the Management API. In this setup, the Control Plane continues to expose its Management API for managing EDC Connector entities, and the Data Plane provides its own Management API for managing Data Flows.

Implementation-wise, an API controller was added to handle requests sent to the new endpoint. A new application service named `DataFlowService` was also introduced as an extension point, responsible for managing Data Flow operations, which in this case includes the logic for triggering a transfer. A default implementation was also provided, which validates that the transfer uses the Push flow type, the data is non-finite, and the Data Flow is still active. If all validations pass, the data transfer is restarted.

The actual transfer continues to be performed by the Pipeline Service. Its new implementation ensures that the Transfer Process and Data Flow remain active, even after sequential data transfers. As a result, the trigger endpoint can be called multiple times, with each call starting a new data transfer. Listing 4.3 shows a snippet of the `DataFlowService` implementation, representing the logic to trigger a data transfer.

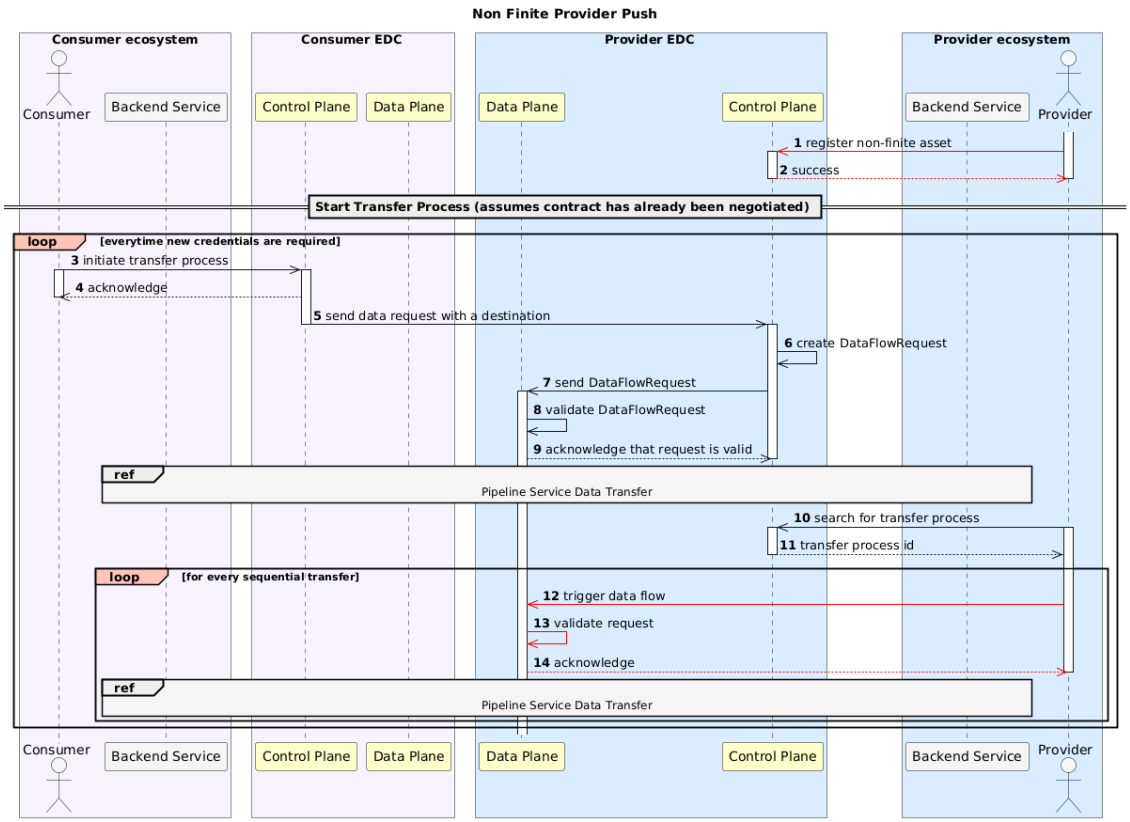
```java
public ServiceResult<Void> trigger(String dataflowId) {
    return ServiceResult.from(
            dataPlaneStore.findByIdAndLease(dataflowId))
            .compose(this::isPushFlowType)
            .compose(this::isNonFinite)
            .compose(this::isInStartedState)
            .onSuccess(this::startDataTransfer)
            .mapEmpty();
}
```

Listing 4.3: Snippet of the Data Flow Service

Triggered transfers do not calculate the difference between previously sent data and new data. All data available in the Data Source at the time of the transfer will be pushed to the Consumer. Since each Data Source has its own peculiarities, providing a generic way to calculate this delta is neither feasible nor always desirable. Instead, it is the Data Supplier's responsibility to manage the contents of the Data Source, adding and clearing information as needed, so that each triggered transfer sends the intended data.

When a new Transfer Process is created, an initial data transfer always occurs before any subsequent transfers triggered by the mechanism. This behavior aligns with finite Provider Push transfers, which also start the transfer immediately. Providers must ensure that their Data Source is ready for transfer as soon as the Contract Agreement is generated, so as to guarantee that all transfers succeed. Figure 4.2 represents the complete process for transferring non-finite data via the Provider Push flow. Once again, preexisting behavior is shown in black and new functionality introduced by this feature is highlighted in red.

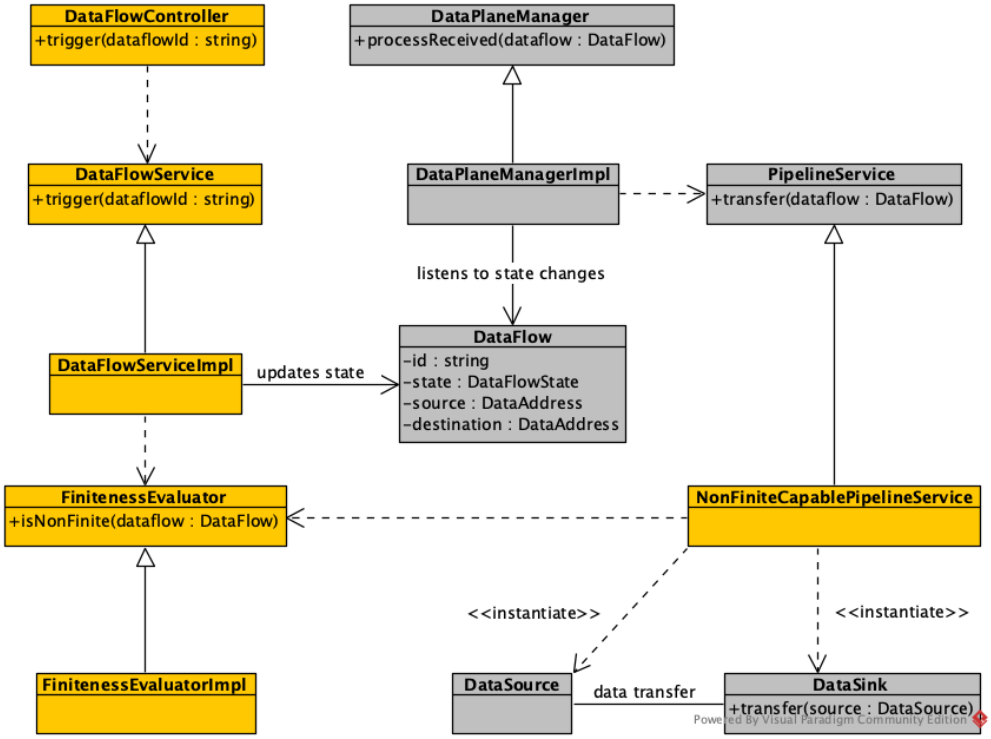
Figure 4.2: Non-Finite Provider Push sequence diagram



### 4.2.4 Architectural Overview

The developed feature introduced several new components and concepts, creating new interactions within the system. Figure 4.3 illustrates these interactions, where gray classes represent existing EDC entities and services, and orange classes represent the newly added components.

Figure 4.3: Non-finite Provider-Push class diagram




Classes were deliberately simplified to display only the most relevant attributes and operations, avoiding unnecessary clutter in the diagram. Moreover, some interactions were also abstracted for clarity, such as the persistence of the `DataFlow`.

The `FinitenessEvaluator` and `FinitenessEvaluatorImpl` classes represent the contributions added to identify non-finite data.

The `NonFiniteCapablePipelineService` represents the new `PipelineService` implementation, that supports both finite and non-finite data.

The triggering mechanism involves the `DataFlowController`, which is the API controller that exposes the trigger endpoint, along with the `DataFlowService` extension point and the `DataFlowServiceImpl` as its default implementation.

Note that the `DataFlowServiceImpl` does not depend directly on the `PipelineService`, and therefore does not invoke the `transfer` method itself. Instead, upon receiving a valid trigger request, it updates the `DataFlow` state. Consequently, the `DataPlaneManager`, which listens for state changes in the `DataFlow`, detects this update and invokes the `transfer` method.

This event-based approach ensures that the `DataFlow` state correctly reflects the initiation of a new transfer, and also enables other listeners to react to the event, such as for audit logging purposes.

### 4.2.5 Access Management

Since non-finite data transfers remain active during long time periods, access management becomes an important task, ensuring that Providers can continue pushing data while maintaining security.

To control access to the Data Sink, Consumers can enforce authentication. While the exact authentication process varies across technologies, it typically involves either obtaining access tokens from a third-party identity provider or using a credentials set. The Transfer Process should carry the access information, passing it to the Provider, and enabling it to push data to the destination.

#### Access Tokens

Access tokens have an expiration date, after which the Provider is no longer allowed to push data. Consumers may optionally provide a refresh mechanism that allows tokens to be renewed, extending the access granted to the Provider. When the Transfer Process is terminated, Consumers should revoke the active token and prevent further refreshes. This process can be done manually, but preferably it should be automated.

Consider an example data exchange where the Data Sink is an HTTP API that enforces authentication through OAuth tokens. While the exact steps may vary, the following sequence outlines the general flow.

1. The Consumer generates an access token and provides it to the Provider.
2. The Provider uses this token to push data to the Data Sink.
3. As the token approaches expiration, the Provider requests a refresh from the Consumer.
4. The Consumer issues a new access token and returns it to the Provider.
5. The Provider continues sending data to the Data Sink.
6. When the Consumer decides to close the connection, it terminates the Transfer Process.
7. The Consumer revokes the active token and blocks further refresh requests.
8. The Provider can no longer push data.

While this is the intended general flow, the EDC Connector currently does not support some of these functionalities, namely token refresh and revocation. These actions must therefore be performed either manually or automated through external applications. The developed feature intentionally did not address these gaps in order to avoid broadening the scope of the changes.

#### Credentials

Credentials are typically key–secret pairs that, when presented at the authentication point, grant access to their holder. If given such credentials, the Provider can reach the data destination either by presenting them directly or by using them to obtain access tokens from an authorization server. Credentials are typically long-lived, which offers great flexibility

since the Provider does not need to request refreshed access from the Consumer, but also reduces security as the impact of losing credentials is bigger.

To mitigate risks, Consumers should periodically rotate credentials. The validity period depends on the specific use case and should be a good compromise between security and allowing an adequate number of data transfers. When credentials are rotated, Consumers should also terminate the existing Transfer Process and initiate a new one with the updated credentials, ensuring the Provider can continue sending data to the Data Sink.

Consider an example data exchange where the Data Sink is an Amazon Web Services (AWS) S3 bucket secured by enforcing AWS security credentials to be presented. While the exact steps may vary, the following sequence illustrates the general flow.

1. The Consumer generates AWS security credentials and provides them to the Provider.
2. The Provider uses these credentials to push data to the Data Sink.
3. When credentials need to be rotated, the Consumer generates new AWS security credentials.
4. The Consumer initiates a new Transfer Process with the updated credentials.
5. The Consumer revokes the old credentials and terminates the previous Transfer Process.
6. The Provider continues sending data to the Data Sink using the new credentials.
7. When the Consumer decides to close the connection, it terminates the active Transfer Process.
8. The Consumer revokes the active credentials.
9. The Provider can no longer push data.

As with access tokens, the EDC Connector does not currently support some of these functionalities such as credential generation and rotation. These actions must be carried out manually or automated through external applications. Once again, the proposed feature does not address these gaps in order to keep its scope focused only on the mechanisms to push non-finite data.

## 4.3 Quality Assurance

Quality assurance mechanisms are critical in OSS projects. Unlike proprietary software, contributions can originate from a wide range of developers, which increases the risk of inconsistencies and errors. If left unchecked, these issues can compromise the maintainability, reliability, and security of the code-base. To mitigate these risks, OSS projects typically employ a combination of automated and manual quality assurance processes.

In the Tractus-X project, Committers conduct code reviews for every pull request, ensuring correct implementations. This process involves checking if coding best practices were followed and verifying that the implementation meets the acceptance criteria and requirements associated with the feature.

Additionally, a suite of automated quality checks runs whenever new code is pushed to the repository. These checks assist committers by handling routine verifications, allowing them

to focus their reviews on more complex topics. They are also beneficial to contributors, as they provide fast feedback on their code, helping them identify and correct issues early.

Some checks perform static code analysis, a quality assurance mechanism that inspects the source code for errors, vulnerabilities, code smells, or other deviations from established coding standards. Although no new static code analysis checks were added or updated as part of the proposed feature, the existing checks were applied to the new code and are therefore important to describe.

Checkstyle [44] was used to enforce coding standards and style conventions, TruffleHog [45] scanned the code-base for secrets such as API keys or passwords, KICS [46] analyzed Infrastructure as Code configurations for potential security issues, and SonarCloud [47] verified Java source code for bugs, vulnerabilities, and code smells.

Other quality checks ran the various test suites available in the repository, which include unit, integration, API, smoke and end-to-end tests. During development, tests were added to these suites to ensure coverage of the newly implemented functionalities.

Unit tests are small, fast, and isolated checks that validate individual units of the code-base. They should be abundant and provide broad coverage of most functionalities. Unit tests were added for all new code, ensuring that each method was thoroughly tested. As an example, two tests were added for the provided `FinitenessEvaluator` implementation, one to verify that a finite Data Flow is correctly identified as finite, and another ensuring that a non-finite Data Flow is correctly identified as non-finite.

Integration tests verify that two or more components work together as expected. For this feature, a new integration test was added for the new implementation of the Pipeline Service. The test sets up a dummy Data Source representing an in-memory input stream and a dummy Data Sink representing an in-memory output stream. The goal of the test is to validate that the integration between these two components during a transfer functioned correctly when orchestrated by the new Pipeline Service implementation.

API tests are a special type of integration tests that validate the integration between an API endpoint and its controller. They run against a test runtime containing a mock HTTP server to assert specific system behaviors. New API tests were added for the introduced trigger endpoint, ensuring that requests were correctly deserialized and routed, and verifying that the expected responses were returned in different scenarios.

Smoke tests are fast, high-level checks that verify the most essential functionalities of an application. They run after a deployment to confirm that the system is up and stable before executing more detailed tests. For this feature, a new smoke test was added to ensure that the trigger endpoint was reachable by sending an HTTP request to it.

End-to-end tests are also a special type of integration tests that cover a functionality in its entirety, verifying the integration of all its underlying components. An end-to-end test was created to cover the complete process of transferring non-finite data, representing the steps participants would follow in practice.

The test begins by registering an Asset that represents non-finite data, followed by the usual process of publishing it as an offer and negotiating it. Once an Agreement is established, a non-finite Provider Push Transfer Process is initiated. The test verifies that the transfer

executes correctly and the Transfer Process remains active. Then, a new transfer is triggered, and the same assertions are confirmed. Finally, the Transfer Process is terminated, completing the test.

## 4.4 Deployment

Throughout the implementation phase, deployments were carried out locally within a controlled Docker Compose environment. While this setup simulates the execution of an EDC Connector in a clustered environment, being very helpful during development, it does not reflect the complexity and unpredictability of a production-grade system.

However, considering the EDC roadmap, deploying a real user-facing environment was not feasible, as the feature will only be officially released at the end of the current release cycle. Nevertheless, a production-like environment was established, where each code increment was rolled out. This environment was used for testing under realistic conditions and for demonstration purposes. An AWS cloud room was used to deploy all the necessary infrastructure.

Regarding compute options, both containers and virtual machines were considered. To guide this decision, the EDC architecture was analyzed. The EDC Connector's physical model consists of at least a Control Plane and a Data Plane, with the latter often requiring multiple instances to handle data transfer loads. This indicates that isolation between components and scalability are important operational requirements to consider.

After evaluating the costs and effort of deploying small, specialized containers versus scaling large, generic virtual machines, containerizing the EDC Connector was chosen as the most suitable approach. As a bonus, Tractus-X already provides the necessary means to create the container images.

Considering the scalability requirements of deploying an EDC Connector, its containers needed to be orchestrated in a clustered environment. This raised the choice between creating an Elastic Container Service (ECS) cluster or an Elastic Kubernetes Service (EKS) cluster. Table 4.1 presents a comparison between these two technologies.

Table 4.1: ECS versus EKS comparison

| **Aspect** | **ECS** | **EKS** |
|---|---|---|
| Engine | AWS internal systems | Kubernetes |
| Ease of use | Simple, minimal setup required | Requires Kubernetes expertise |
| Portability | Limited to AWS | Vendor-neutral |
| Scaling | AWS auto-scaling service | Kubernetes-native scaling |
| Price | Pay for compute | Pay for compute |

Overall, both options are suitable and would serve the deployment needs. However, since this setup is mainly intended for testing and demonstrations, portability is not required. On the other hand, simplifying the deployment process helped accelerate development. For this reason, the EDC Connector was ultimately deployed in an ECS cluster.

Auto-scaling was enabled in the cluster to automatically adjust the number of instances according to the current load. This service monitors the containers resource consumption to spin up new instances when demand increases and remove them when demand decreases.

An AWS Application Load Balancer (ALB) was also deployed to distribute traffic across the instances. The ALB automatically updates its targets as instances are added or removed, and was configured with routing rules to direct traffic to either the Control Plane or the Data Plane. Consequently, it was set up as the cluster's entry point.

Regarding the networking, deploying EDC Connectors can be particularly challenging. Many organizations operate with intranets, as in private networks not accessible from the internet, where internal resources reside. Connectors must be strategically deployed at the boundary between the intranet and the internet.

The DSP API must be exposed to the internet so that other Data Space components can communicate with the Connector. At the same time, the Management API, both on the Control Plane and the Data Plane, must be exposed to the intranet, allowing the participant managing the Connector to interact with it.

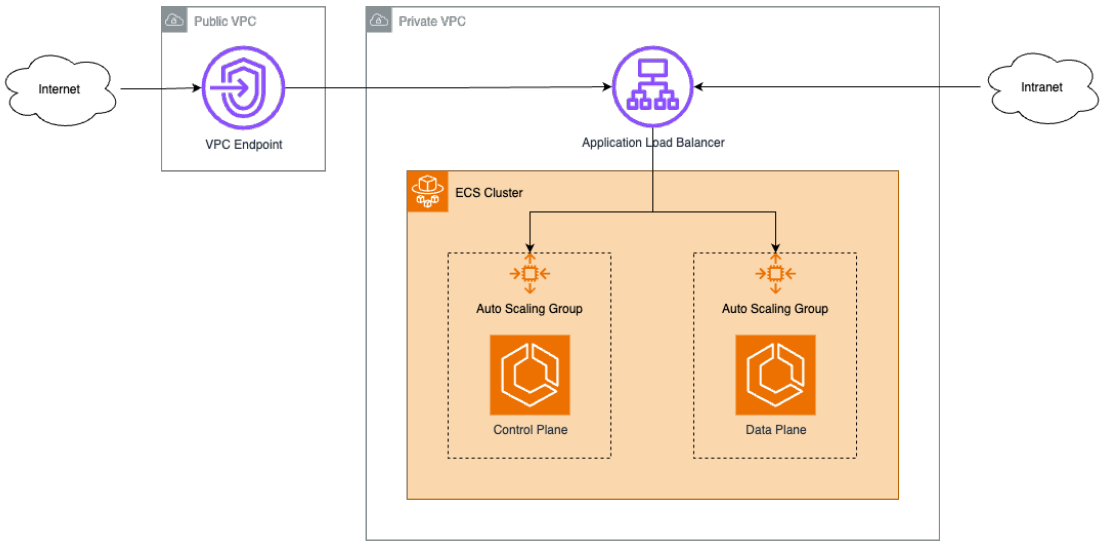
Furthermore, EDC Connectors require outbound internet access, as they also need to communicate with other Data Space components, as well as intranet access to retrieve data from internal Data Sources.

Two networks were created in the AWS cloud room as Virtual Private Cloud (VPC)s: one public, with internet access, and another private, with intranet access. The chosen setup was to deploy the EDC Connector in the private VPC, and enable connectivity to the public VPC through supporting infrastructure.

A reversed setup could also be achieved, deploying the EDC Connector in the public VPC and configuring routing to the private VPC. Both setups would be equivalent and viable.

Figure 4.4 illustrates the physical view of this environment. Additional ancillary infrastructure was deployed for security, configuration and persistence, but for simplicity purposes, not all resources are shown.

Figure 4.4: Physical view



# Chapter 5

# Evaluation

Following the development process, the outcomes of the created work were analyzed. This chapter evaluates the results achieved with the developed feature, outlining what has been accomplished and identifying what remains to be done.

It begins by describing the evaluation methodology used to collect data on the outcomes of this feature. Next, it reviews the developed work, assessing its alignment with the initial requirements. Furthermore, it outlines the feature's adoption process and identifies potential future work deriving from this initial implementation.

At the end of this chapter, a brief discussion of the assessment is provided, presenting the final evaluation results.

## 5.1 Methodology

The outcomes of the developed feature were analyzed from a technical and a business perspective, ensuring that both dimensions were properly addressed.

The technical evaluation was performed by the Tractus-X committers in each pull request review. Besides verifying that good coding practices and project standards were used, they also compared the proposed changes against the design stipulated in the decision record, ensuring coherence with the defined approach. Feedback was promptly provided throughout the process, requesting changes whenever necessary.

On a business level, the feature was demonstrated to the participants of the interview process, showcasing the new functionality. This demonstration took place in the production-like deployed environment to simulate real conditions. From this presentation, stakeholders were able to understand when to apply the developed feature, as well as the necessary steps to use it.

The adoption process was also discussed, with migration plans being created for use cases currently relying on workarounds to push non-finite data. Each plan details the steps required for a successful transition. In the end, the demonstration participants provided feedback on this functionality, offering a qualitative perspective on the developments.

## 5.2 Review

The developed feature proved to be quite comprehensive, introducing several new extensions and generating large discussions within the community. In the end, all three contributions

were approved and integrated into the code-base, enabling the transfer of non-finite data through the Provider Push flow.

The first contribution added an extension point that evaluates the finiteness of a Data Flow, allowing participants to identify their Assets that represent non-finite data. The second contribution provided a new Pipeline Service implementation, capable of handling non-finite data transfers. Finally, the third contribution introduced a new Data Plane endpoint to trigger new transfers, enabling Data Providers to continuously push their non-finite data.

To evaluate the outcomes of the development, it is important to revisit the initially defined functional and non-functional requirements, and analyze how the final implementation fulfills them.

**FR1 - Consumers must be able to create Transfer Processes that allow multiple data transfers**

This requirement was fulfilled through the combination of all contributions. By enabling the identification of non-finite data, Transfer Processes can remain active, allowing multiple transfers to be initiated via the trigger endpoint. Additionally, the security aspect of this requirement was also considered, as there were designed mechanism to manage access to the data destination.

**FR2 - Providers must be able to manually trigger the transfer of data to a single partner**

This requirement was achieved by the introduction of the trigger mechanism. The new trigger endpoint initiates a data transfer for a specific Data Flow, which represents a Transfer Process backed up by a Contract Agreement between two participants. Consequently, each triggered transfer exchanges data with the counter party identified in the underlying Contract Agreement.

**FR3 - Providers must be able to manually trigger the transfer of data by dispatching an event**

This requirement was also addressed by the introduction of the trigger mechanism. Instead of directly executing the transfer, the trigger endpoint updates the state of the Data Flow to reflect a received transfer request. This operation dispatches an event, allowing associated event handlers to execute asynchronously. One of these handlers invokes the Pipeline Service, starting the transfer.

**FR4 - Participants should be able to terminate Transfer Processes on demand, closing the communications channel**

This requirement was already supported prior to the work developed in this dissertation, as the Management API includes an endpoint for terminating Transfer Processes. Nevertheless, this is an essential aspect of the new feature, and was taken into account during the creation of the end-to-end test.

**NFR1 - The system should integrate and reuse existing Provider Push concepts where applicable**

This requirement guided every decision taken throughout development. Although some new concepts had to be introduced, such as the trigger mechanism, the overall process remains quite similar to the existing Push flow for finite data transfers. For example, this feature

reuses the Data Sources and Data Sinks of the Pipeline Service, avoiding the need to create new ones for non-finite data transfers.

**NFR2 - The system should integrate and reuse Consumer Pull concepts where applicable**

This requirement was relevant for designing access management mechanisms for non-finite transfers, since Consumer Pull similarly relies on access tokens and credentials.

## 5.3 Feature Adoption

New business use cases revolving around pushing non-finite data can be established with relative ease, as the process is largely similar to how other use cases are setup. Providers must identify their Assets that point to non-finite data and register them in their EDC Connector. From there, the standard process of publishing offers and negotiating them applies. However, the data exchange is somewhat distinct from finite transfers, and both Consumers and Providers must understand it. Consumers need to correctly configure their access management mechanisms to ensure security, and Providers are responsible for triggering the data transfer when required.

Furthermore, recalling back to the business analysis, there are already business use cases in production that handle non-finite data through workarounds. For compliance, and to avoid the issues identified with each workaround, these use cases should migrate to the new setup.

The workaround of transferring data chunks individually treats non-finite data transfers as an endless sequence of finite transfers, with each transfer sending a fragment of the data. The issue with this approach is that the Consumer must explicitly request each chunk, resembling the Consumer Pull flow.

Use cases relying on this workaround can migrate to the new setup with minimal adaptations. The Provider needs to update their Asset so that it identifies the data as being non-finite, after which the new transfer model can be used.

The workaround of inverting the participants' roles switches the usual responsibilities, so that, from the perspective of the EDC Connector, the Data Supplier is treated as the Consumer, and the Data Customer is treated as the Provider. This approach exploits the capabilities of the Consumer Pull flow, so that when a request reaches the endpoint where the data should be pulled from, the payload is actually forwarded to the Asset Data Address instead. This is an anti-pattern as it defies the policy enforcement mechanism, and bypasses proper credential validation.

Moving away from this approach is not as straightforward as the previous workaround. Since the participant roles are reversed, new Assets, Policy Definitions, and Contract Definitions must be created in the appropriate EDC Connector. Afterwards, a new Contract Agreement must be established before transfers can begin. In addition, the DPF already supports a token refresh mechanism for Consumer Pull transfers, but one is not present for Provider Push transfers. Given that these changes are significant, resistance from the Data Space participants is expected. Nevertheless, use cases should still migrate to avoid relying on a dangerous anti-pattern.

## 5.4 Future Work

Considering that the developed feature represents an initial step towards enabling non-finite data in Push transfers, there are still improvements to be done. A natural next step would be to address the identified gaps in access management. Introducing extensions for token refresh and revocation, as well as mechanisms for automatic credential generation and rotation, would reduce the number of manual tasks required from participants and, in turn, lower the overall complexity of the process.

Another improvement to the current approach would be to introduce distinct Transfer Services for finite and non-finite data transfers. Managing both in the same Pipeline Service implementation creates unnecessary coupling, which affects the maintainability of Tractus-X. Currently, only a single Pipeline Service implementation can exist in a runtime. If this behavior were changed to allow multiple implementations to be registered, the default implementation could continue handling finite data transfers, and the new implementation would manage only non-finite data transfers.

There is also work on Push non-finite data transfers that can be done beyond the scope of this feature. As an example, enabling the trigger of a batch of Data Flows could be beneficial. In a scenario where an Asset is consumed by multiple participants, being able to trigger transfers for all associated Data Flows with a single request would be more efficient than issuing multiple requests, one for each partner.

Another possible feature would be the introduction of listeners that actively detect changes in the Data Source, and automatically trigger a transfer without requiring Provider intervention. Such a feature would be particularly useful for streaming technologies that support data push.

Overall, there are endless possibilities to move forward from this initial implementation. With the adoption of this feature, new requirements are bound to appear, promoting the development of new ways to handle non-finite data transfers.

## 5.5 Discussion

Since every contribution was integrated into the Tractus-X project, from a technical standpoint, the development was a success. Although changes were requested along the way, eventually all pull requests were approved.

During the demonstration, the feedback given by stakeholders was very positive. They understood how the new approach works and its benefits, which enabled the elaboration of adoption plans for use cases that currently rely on workarounds. Overall, the introduction of this feature is bound to solve the issues identified in each workaround, whether by reducing complexity and costs or by providing a Data Space compliant way of pushing non-finite data.

Even though these contributions were considered a success, the work on non-finite Provider-Push transfers is not yet complete. There is still work to be done on this topic to fully unlock its potential.

The outcomes of this feature could only be evaluated through qualitative feedback. Although the contributions were merged into the code-base, EDC only publishes official releases every three months. As a result, Data Space participants will only be able to use this feature once the current cycle concludes.

Moreover, most participants do not update their Connectors immediately after a new release. There is usually a testing and stabilization phase in a staging environment before bumping their production distributions. Finally, participants also require time to adapt their use cases to the new model, following the provided migration guides.

This means that the feature will take a long time to reach its intended users, making it impossible to obtain quantitative data on its benefits as this stage.

# Chapter 6

# Conclusions

This final chapter brings together the outcomes of every phase of the dissertation and reflects on the work carried out.

It starts by revisiting the initially defined objectives, analyzing what was achieved for each one, and assessing the extent to which they were fulfilled. Next, it offers a critical retrospective on the methodologies applied throughout the dissertation, highlighting valuable processes and identifying practices that should be reconsidered.

The chapter then outlines the limitations encountered, examining their impact on the outcomes and the ways in which these challenges were addressed. Finally, it concludes this dissertation with some closing remarks, providing a perspective on the future of non-finite data transfers in organizations.

## 6.1 Achievements

Looking back at the introductory chapter, this dissertation was grounded in a clear problem: EDC Connectors lack support for asynchronous non-finite data transfers. To address this challenge, three objectives were defined, which aimed to gather the necessary understanding of the problem and to develop a technical solution to resolve it.

Based on these objectives, a research question was formulated to represent the fundamental goal of the dissertation in the form of a query. Both the objectives and the research question served as guiding principles throughout the work carried out.

Now, as this dissertation comes to an end, it is clear that all defined objectives were successfully met, and the research question answered. This section outlines each achievement in detail, describing the work undertaken to accomplish them.

**Understand what are Data Spaces, their architecture, and the value they bring to businesses**

The concept of Data Spaces was extensively examined in the state-of-the-art analysis, forming the knowledge foundation that supported this dissertation.

Data Spaces are digital, federated platforms formed out of the need for organizations to exchange data among themselves in a secure, controlled, and standardized manner. They offer tools for participants to provide and consume data.

The key features of Data Spaces are the interoperability across diverse sources and formats, the possibility to define access control mechanisms to data, the assurance of security and privacy measures, the guarantee of data sovereignty allowing participants to retain ownership

of their data, and the promotion of value creation by fostering innovative products and services.

The state-of-the-art analysis highlights the growing relevance of Data Spaces, driven by the increasing demand for data between organizations. Notable examples include the Catena-X project and the European Commission, which demonstrate how Data Spaces are being established and applied in businesses across Europe.

On their own, Data Spaces are mere abstract concepts without a clearly defined architecture, allowing participants to freely organize their setup according to their specific needs. The IDS is a leading initiative in Data Space design, providing a reference architecture for building Data Spaces. This architecture is described in the IDS RAM.

The IDS RAM defines five layers: the business layer, which specifies the roles of Data Space participants; the functional layer, which outlines the Data Space features; the information layer, which defines the information model; the process layer, which describes the interactions and workflows within a Data Space; and the system layer, which details the technical components that support it.

By thoroughly reviewing the available literature, the student gained the necessary understanding to achieve this objective.

**Understand how the EDC Connector works as a tool to exchange data**

EDC Connectors were also described in detail in the state-of-the-art analysis. This research provided an in-depth understanding of the processes attached to this technology, with special regards to the data exchange.

The EDC project is an OSS framework for implementing IDS components. It was designed as a modular platform with an extension system, allowing downstream projects to plug-in the extensions they require.

At the core of this project is the EDC Connector, a pair of components that collectively form an IDS Connector implementation. These components are the Control Plane and the Data Plane.

The Control Plane is responsible for managing EDC entities and DSP processes, allowing participants to operate their EDC Connectors. It is also the point of interaction with other Connectors, handling both inbound and outbound communication.

The Data Plane is responsible for executing the data transfers over a wire protocol, operating under the direction of the Control Plane. Multiple Data Planes can coexist in an EDC Connector, and their implementations can vary widely.

Transfer Processes support two transfer flows: Consumer Pull transfers enable the Data Consumer to retrieve the desired data from the Data Source by requesting an endpoint that synchronously returns the required information. Provider Push transfers place the responsibility to send data on the Data Provider, so that when a request to initiate a Transfer Process is received, the data is asynchronously pushed to the Data Consumer.

The finiteness of data directly affects how it is exchanged. Finite data transfers conclude once all data has been delivered, at which point the Transfer Process completes. On the other hand, non-finite data transfers have no predefined completion point because new data continuously becomes available. In this case, the Transfer Process remains active until it is manually terminated by either the Consumer or the Provider.

This investigation on the EDC and its data exchange process provided the student with the knowledge required to achieve this objective.

**Develop support for asynchronous data transfers of non-finite data**

From a practical standpoint, the student created a solution to enable asynchronous data transfers of non-finite data during the development phase. This solution was based on the knowledge acquired from the previous research on Data Spaces and the EDC project.

The first step towards developing this feature was to analyze business use cases that rely on Data Spaces, in order to understand how participants are using their EDC Connectors. To achieve this, interviews were conducted with key stakeholders. Moreover, functional and non-functional requirements were also obtained from the interview process. The insights gained from these conversations were crucial for designing the solution.

Following this process, an initial approach was outlined and presented to the community, where it was discussed and refined. Once all interested parties reached a consensus on how to tackle this issue, a decision record was formalized. This marked the beginning of the implementation phase, where the agreed approach was implemented.

The developed solution enables the exchange of non-finite data through the Provider Push flow. When the Consumer initiates a Transfer Process, data starts to be sent to the data destination. However, unlike finite data transfers, the Transfer Process remains active after the first data chunk is transferred, allowing additional data to be transmitted until either participant manually closes the communication channel.

This feature was integrated into the current EDC release cycle. Once officially released, interested participants will be able to include it into their EDC Connector distributions. While new use cases will be able to simply plug-in the corresponding extensions and start using the new feature, existing use cases that already handle Push non-finite data transfers through workarounds will need to migrate their solutions.

The outcomes of the development phase directly address the underlying problem described in this dissertation. At the same time, the process enriched the student's knowledge on the behavior of the EDC Connector by applying theoretical concepts to real-world scenarios.

**How can the EDC Connector be extended to support asynchronous transfers of non-finite data while aligning with the principles of Data Spaces?**

Three steps were required to support this: enable Providers to identify their non-finite data, adapt the Push transfer mechanism to handle non-finite data, and allow Providers to trigger additional data transfers. Each step was implemented as a distinct contribution.

For Providers to identify their non-finite data, a new Data Address property was added. This property is not tied to any specific technology, and can be included in any type of Data Address. Providers can set it to indicate the finiteness of the data represented by the Asset.

Non-finite Provider Push transfers were enabled by extending the behavior of the Pipeline Service. In practice, the event signaling the completion of a data transfer is not dispatched if the transfer involves non-finite data. As a result, the Transfer Process remains open, allowing additional data to be transmitted over time.

Subsequent data transfers were made possible through the introduction of a trigger mechanism, which empowers Data Providers to dispatch transfers on demand. This mechanism

accepts transfer requests and, upon successful validation, emits an event to asynchronously perform a non-finite data transfer.

Although no specific developments were made to the access control mechanisms provided by the EDC, authentication was carefully considered for non-finite Provider Push transfers. Whether using access tokens or credentials, these should be refreshed or rotated periodically to prevent long-lived access to the data destination.

## 6.2 Retrospective

Throughout the dissertation, many processes and methodologies were applied across the different phases. While some of these proved to be crucial for the development of the feature, others did not perform as planned.

The research methodology used in the state-of-the-art analysis followed the PRISMA statement, conducting a systematic mapping review of the literature on Data Spaces, Connectors, and their data transfer mechanisms. This approach resulted in a valuable set of papers that provided key insights for the work developed in this dissertation. However, the process was very time-consuming, and some documents had to be manually added to the final articles set. Overall, despite these drawbacks, the use of this methodology proved to be beneficial.

The elicitation process was conducted through interviews with stakeholders. These interviews proved to be a very useful practice, helping to identify real use cases involving Push non-finite data transfers and to define technical requirements. The insights obtained guided the decisions taken throughout the design and implementation phases, and were undoubtedly crucial to the success of the contributed work.

The contribution process defined by the EDC project turned out to be a significant setback of this dissertation. Given the large scope of the feature, there were many divergent opinions on how to approach it, causing community discussions to take much longer than anticipated and delaying all subsequent work. While following the contribution guidelines is mandatory, future contributions could benefit from employing additional practices. For example, consulting directly with a committer and aligning on an initial design privately before presenting it to the community could help accelerate the process.

Regarding planning methodologies, the predictive approach enforced by the EDC project did work well. Despite the delays caused by the community discussions, the Waterfall-like structure used during development allowed to easily adjust the project timelines. Subsequent phases were postponed until the conclusion of the discussions.

The use of Scrum during the implementation yielded mixed results. On the one hand, it encouraged breaking down the feature into smaller contributions, which were easier to implement, test, review, and demonstrate. This allowed to get continuous feedback during development, as progress was presented to interested participants at the end of each sprint. On the other hand, Scrum's short development cycles are not ideal for the open-source contribution process. Since contributors have little control over how long a contribution remains in review, as this depends on the availability of committers, pull requests can remain open for extended periods. This uncertainty complicates iteration planning and reduces the effectiveness of Scrum in this context.

The adoption of DevOps practices was, as expected, very useful in delivering increments fast and with consistent quality. Their application was crucial in ensuring steady progress.

The final demonstration to stakeholders during the evaluation phase was effective in gathering qualitative feedback on the outcomes of the development process. This approach allowed to assess whether the requirements were met and helped defining next steps. Nonetheless, obtaining quantitative data from production use cases would have provided stronger evidence of the benefits introduced by the developed feature.

On a general note, the processes applied throughout this dissertation proved to be valuable. While there is room for improvement, these practices contributed positively to the outcomes of this work.

## 6.3 Limitations

Although the dissertation successfully met all defined objectives, the process was not without its limitations. Several challenges appeared along the way, constraining the work carried out and, in some cases, potentially influencing the outcomes.

The first limitation emerged during the interview process. Given the decentralized nature of Data Spaces, the ideal approach would have been to interview a diverse set of participants. While the interviews conducted yielded valuable insights, including a broader set of organizations could have provided more comprehensive results. However, because EDC Connectors operate with sensitive data, discussing production use cases with organizations risked potential legal implications, making them reluctant to participate.

Another limitation of the interview process was the restricted amount of information that could be disclosed in this dissertation document due to the sensitivity of the topics discussed. While the outcomes themselves were not affected, as they derived directly from the conversations, this necessary discretion may create some gaps for the reader.

The initial feature proposal not being well received by the EDC committers also proved to be an unexpected challenge, as downstream projects have limited control over core functionality and must largely conform to the EDC's extensibility model. Although the design was able to be quickly adapted, this could have been a major obstacle if the proposed changes included functionality that could not be modified from a downstream project.

Given the open-source nature of this dissertation, a major limitation was the dependency on the community. During the design phase, the proposal had to be discussed and agreed upon by community members, a process complicated by conflicting opinions on how to proceed. During implementation, committers needed to review every contribution, a process heavily dependent on their availability and subject to differing viewpoints on implementation details. Overall, this reliance on external factors resulted in significant delays in the development of the dissertation.

Finally, a major limitation of this dissertation was the inability to evaluate the feature in production use cases at the time of writing. Being restricted to the EDC release cycle, which publishes contributions only every three months, means that the feature has not been officially released yet. Even after the release, participants will need to upgrade to the latest version, which is typically a slow process, and existing use cases handling non-finite data in Push scenarios will also require additional time to migrate to the new approach. Overall, these timing constraints made it impossible to collect data from production deployments, significantly affecting the evaluation phase.

## 6.4 Closing Remarks

The role of data in organizations is becoming more and more important in today's digital landscape, yet exchanging it securely and in a controlled manner remains a significant challenge. Data Spaces represent a modern approach that helps address this issue.

Non-finite data in particular is increasingly becoming the norm in many processes, especially with the rise of streaming technologies that inherently depend on it.

The conducted interviews reinforced this trend, as all analyzed use cases involved non-finite data. This dissertation contributes to this evolution by extending the ways such data can be transferred using EDC Connectors, taking an important step toward its standardization across Data Spaces.

Beyond the technical solution, the state-of-the-art review on Data Spaces and the EDC offers a comprehensive analysis of these technologies. These insights may prove highly valuable for researchers in this area.

While many challenges remain, the contributions provided by this dissertation lay the groundwork for future improvements. Businesses across Data Spaces can leverage this initial approach on non-finite data applied to asynchronous transfers as a basis to define new requirements.

Ultimately, this work reinforces the value of data sharing in an increasingly data-driven world. In the end, this dissertation has hopefully demonstrated how the work carried out will shape the way organizations transfer non-finite data in the years to come.

# Bibliography


[1] Kevin Bartley. *Big data statistics: How much data is there in the world?* Dec. 2024. url: https://rivery.io/blog/big-data-statistics-how-much-data-is-there-in-the-world.

[2] Christine Hoyt. *The importance of data and analytics in making informed and strategic decisions*. Jan. 2023. url: https://strategy.wsu.edu/the-importance-of-data-and-analytics-in-making-informed-and-strategic-decisions.

[3] SAP SE. *The Ultimate Guide to Supply Chain Collaboration*. Aug. 2024. url: https://www.sap.com/resources/supply-chain-collaboration.

[4] International Data Spaces Association. *Data sovereignty*. Sept. 2024. url: https://internationaldataspaces.org/why/data-sovereignty.

[5] Adhil Badat. *Data sovereignty: Keeping your bytes in the right place*. Dec. 2024. url: https://www.rackspace.com/blog/data-sovereignty-data-protection-strategy.

[6] International Data Spaces Association. *How to Build Data Spaces? | IDS Knowledgebase*. Oct. 2023. url: https://docs.internationaldataspaces.org/ids-knowledgebase/how-to-build-data-spaces.

[7] Fraunhofer ISST. *Dataspace Connector*. 2021. url: https://international-data-spaces-association.github.io/DataspaceConnector/.

[8] International Data Spaces Association. *IDS RAM 4*. Feb. 2023. url: https://docs.internationaldataspaces.org/ids-knowledgebase/ids-ram-4.

[9] International Data Spaces Association. *Data Spaces*. 2024. url: https://internationaldataspaces.org/why/data-spaces/.

[10] Maria Teresa Delgado. *Eclipse Dataspace Components*. June 2021. url: https://projects.eclipse.org/projects/technology.edc.

[11] Eclipse Foundation. *Connector*. Nov. 2024. url: https://github.com/eclipse-edc/Connector.

[12] Catena-X. *Catena-X*. 2024. url: https://catena-x.net/en/1/about-us.

[13] International Data Spaces Association. *Dataspace Protocol 2024-1*. Feb. 2024. url: https://docs.internationaldataspaces.org/ids-knowledgebase/dataspace-protocol.

[14] Mary Shaw. *What Makes Good Research in Software Engineering?* School of Computer Science, Carnegie Mellon University, 2002. url: https://www.cs.cmu.edu/~Compose/ftp/shaw-fin-etaps.pdf.

[15] Briony J Oates, Marie Griffiths, and Rachel McLean. *Researching Information Systems and Computing*. Los Angeles: Sage, 2006. isbn: 9781412902236.

[16] Catena-X. *Catena-X Automotive Network e.V. List of Members*. Nov. 2024. url: https://catena-x.net/fileadmin/user_upload/06_Ueber_uns/Catena-X_List_of_Members_02.pdf.

[17] Durham University. *Evidence-Based Software Engineering*. Feb. 2024. url: https://ebse.webspace.durham.ac.uk/ (visited on 11/11/2024).

[18] M. Salama, R. Bahsoon, and N. Bencomo. *Chapter 11 - Managing Trade-offs in Self-Adaptive Software Architectures: A Systematic Mapping Study*. Ed. by Ivan Mistrik et al. Boston: Morgan Kaufmann, 2017, pp. 249–297. isbn: 978-0-12-802855-1. doi: https://doi.org/10.1016/B978-0-12-802855-1.00011-3. url: https://www.sciencedirect.com/science/article/pii/B9780128028551000113.

[19] Matthew J Page et al. "PRISMA 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews". In: *BMJ* 372 (2021). doi: 10.1136/bmj.n160. eprint: https://www.bmj.com/content/372/bmj.n160.full.pdf. url: https://www.bmj.com/content/372/bmj.n160.

[20] Guy Paré et al. "Synthesizing information systems knowledge: A typology of literature reviews". In: *Information & Management* 52.2 (2015), pp. 183–199. issn: 0378-7206. doi: https://doi.org/10.1016/j.im.2014.08.008. url: https://www.sciencedirect.com/science/article/pii/S0378720614001116.

[21] Eclipse Foundation. *EDC Documentation*. Sept. 2024. url: https://eclipse-edc.github.io.

[22] Zeyd Boukhers, Christoph Lange, and Oya Beyan. "Enhancing Data Space Semantic Interoperability through Machine Learning: a Visionary Perspective". In: *Companion Proceedings of the ACM Web Conference 2023*. WWW '23 Companion. Austin, TX, USA: Association for Computing Machinery, 2023, pp. 1462–1467. isbn: 9781450394192. doi: 10.1145/3543873.3587658. url: https://doi.org/10.1145/3543873.3587658.

[23] Marcel Altendeitering, Tobias Moritz Guggenberger, and Frederik Möller. "A design theory for data quality tools in data ecosystems: Findings from three industry cases". In: *Data & Knowledge Engineering* 153 (2024), p. 102333. issn: 0169-023X. doi: https://doi.org/10.1016/j.datak.2024.102333. url: https://www.sciencedirect.com/science/article/pii/S0169023X24000570.

[24] Alberto Montero Fernández et al. "A catalyst for European cloud services in the era of data spaces, high-performance and edge computing: NOUS". In: *Proceedings of the 4th Eclipse Security, AI, Architecture and Modelling Conference on Data Space*. eSAAM '24. Mainz, Germany: Association for Computing Machinery, 2024, pp. 1–9. isbn: 9798400709845. doi: 10.1145/3685651.3686660. url: https://doi.org/10.1145/3685651.3686660.

[25] Simon Jungbluth et al. "Architecture for Shared Production Leveraging Asset Administration Shell and Gaia-X". In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. 2023, pp. 1–8. doi: 10.1109/INDIN51400.2023.10218150.

[26] Wenkai Li and Paul Quinn. "The European Health Data Space: An expanded right to data portability?" In: *Computer Law & Security Review* 52 (2024), p. 105913. issn: 0267-3649. doi: https://doi.org/10.1016/j.clsr.2023.105913. url: https://www.sciencedirect.com/science/article/pii/S0267364923001231.

[27] Amir Shayan Ahmadian et al. "Privacy-Friendly Sharing of Health Data Using a Reference Architecture for Health Data Spaces". In: *Proceedings of the 4th Eclipse Security, AI, Architecture and Modelling Conference on Data Space*. eSAAM '24. Mainz, Germany: Association for Computing Machinery, 2024, pp. 103–112. isbn: 9798400709845. doi: 10.1145/3685651.3685657. url: https://doi.org/10.1145/3685651.3685657.

[28] Can ATİK. "Horizontal intervention, sectoral challenges: Evaluating the data act's impact on agricultural data access puzzle in the emerging digital agriculture sector". In: *Computer Law & Security Review* 51 (2023), p. 105861. issn: 0267-3649. doi: https:

//doi.org/10.1016/j.clsr.2023.105861. url: https://www.sciencedirect.com/science/article/pii/S0267364923000717.

[29] Qusai Ramadan et al. "Data Trading and Monetization: Challenges and Open Research Directions". In: *Proceedings of the 7th International Conference on Future Networks and Distributed Systems*. ICFNDS '23. Dubai, United Arab Emirates: Association for Computing Machinery, 2024, pp. 344–351. isbn: 9798400709036. doi: 10.1145/3644713.3644758. url: https://doi.org/10.1145/3644713.3644758.

[30] Hendrik Meyer zum Felde et al. "Extending Actor Models in Data Spaces". In: *Companion Proceedings of the ACM Web Conference 2023*. WWW '23 Companion. Austin, TX, USA: Association for Computing Machinery, 2023, pp. 1447–1451. isbn: 9781450394192. doi: 10.1145/3543873.3587645. url: https://doi.org/10.1145/3543873.3587645.

[31] Danniar Reza Firdausy et al. "A Data Connector Store for International Data Spaces". In: *Cooperative Information Systems: 28th International Conference, CoopIS 2022, Bozen-Bolzano, Italy, October 4–7, 2022, Proceedings*. Bozen-Bolzano, Italy: Springer-Verlag, 2022, pp. 242–258. isbn: 978-3-031-17833-7. doi: 10.1007/978-3-031-17834-4_14. url: https://doi.org/10.1007/978-3-031-17834-4_14.

[32] Bahar Farahani and Amin Karimi Monsefi. "Smart and collaborative industrial IoT: A federated learning and data space approach". In: *Digital Communications and Networks* 9.2 (2023), pp. 436–447. issn: 2352-8648. doi: https://doi.org/10.1016/j.dcan.2023.01.022. url: https://www.sciencedirect.com/science/article/pii/S2352864823000354.

[33] Tímea Czvetkó and János Abonyi. "Data sharing in Industry 4.0 - AutomationML, B2MML and International Data Spaces-based solutions". In: *Journal of Industrial Information Integration* 33 (2023), p. 100438. issn: 2452-414X. doi: https://doi.org/10.1016/j.jii.2023.100438. url: https://www.sciencedirect.com/science/article/pii/S2452414X23000110.

[34] Anhelina Kovach et al. "Sovereign IIoT Data Exchange Using DAG-Based DLT and International Data Spaces Architecture". In: *Proceedings of the 4th Eclipse Security, AI, Architecture and Modelling Conference on Data Space*. eSAAM '24. Mainz, Germany: Association for Computing Machinery, 2024, pp. 76–85. isbn: 9798400709845. doi: 10.1145/3685651.3686658. url: https://doi.org/10.1145/3685651.3686658.

[35] Jacopo Marino et al. "Enabling Compute and Data Sovereignty with Infrastructure-Level Data Spaces". In: *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*. eSAAM '23. Ludwigsburg, Germany: Association for Computing Machinery, 2023, pp. 77–85. isbn: 9798400708350. doi: 10.1145/3624486.3624509. url: https://doi.org/10.1145/3624486.3624509.

[36] Julia Pampus, Brian-Frederik Jahnke, and Ronja Quensel. "Evolving Data Space Technologies: Lessons Learned from an IDS Connector Reference Implementation". In: *Leveraging Applications of Formal Methods, Verification and Validation. Practice: 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part IV*. Rhodes, Greece: Springer-Verlag, 2022, pp. 366–381. isbn: 978-3-031-19761-1. doi: 10.1007/978-3-031-19762-8_27. url: https://doi.org/10.1007/978-3-031-19762-8_27.

[37] Ilka Jussen et al. "Issues in inter-organizational data sharing: Findings from practice and research challenges". In: *Data & Knowledge Engineering* 150 (2024), p. 102280. issn: 0169-023X. doi: https://doi.org/10.1016/j.datak.2024.102280. url: https://www.sciencedirect.com/science/article/pii/S0169023X24000041.

[38] Simon Jungbluth et al. "Architecture for Shared Production Leveraging Asset Administration Shell and Gaia-X". In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. 2023, pp. 1–8. doi: 10.1109/INDIN51400.2023.10218150.

[39] Vasileios Karagiannis, Astrid Al-Akrawi, and Oliver Hödl. "Data Sovereignty at the Edge of the Network". In: *2023 IEEE 7th International Conference on Fog and Edge Computing (ICFEC)*. 2023, pp. 33–39. doi: 10.1109/ICFEC57925.2023.00013.

[40] Adobe Communications Team. *Waterfall Methodology: Project Management*. Mar. 2022. url: https://business.adobe.com/blog/basics/waterfall.

[41] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. Schwaber, Ken and Sutherland, Jeff, Nov. 2020. url: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf.

[42] Wilson College. *What Is Hybrid Project Management?* Aug. 2024. url: https://online.wilson.edu/resources/hybrid-project-management/.

[43] Ramtin Jabbari et al. "What is DevOps? A Systematic Mapping Study on Definitions and Practices". In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. XP '16 Workshops. Edinburgh, Scotland, UK: Association for Computing Machinery, 2016. isbn: 9781450341349. doi: 10.1145/2962695.2962707. url: https://doi.org/10.1145/2962695.2962707.

[44] 2013. url: https://checkstyle.sourceforge.io/.

[45] 2025. url: https://trufflesecurity.com/.

[46] 2025. url: https://kics.io/index.html.

[47] 2025. url: https://www.sonarsource.com/products/sonarcloud/.