ETSI GR PDL 009 V1.1.1 (2021-09)

Permissioned Distributed Ledger (PDL); Federated Data Management

Reference

DGR/PDL-009_Fed_Data_Mgmt

Keywords

authentication, data preservation, data sharing, privacy, security

Contents

Intellectual Property Rights

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Permissioned Distributed Ledger (PDL).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document will describe use case scenarios, functional architecture, key functional components mechanisms of leveraging PDL for federated data management (e.g. PDL for federated learning, the integration of PDL and the whole data pipeline).

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Industrial Internet Consortium: "The Industrial Internet of Things Volume G1: Reference Architecture (Version 1.9)", June 19, 2019.

NOTE: Available at https://www.iiconsortium.org/pdf/IIRA-v1.9.pdf.

[i.2] A. C. Yao: "Protocols for Secure Computations", 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 1982, Pages 160-164.

[i.3] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart and R. N. Wright: "From Keys to Databases - Real-World Applications of Secure Multi-Party Computation", The Computer Journal, Volume 61, Issue 12, December 2018, Pages 1749-1771.

[i.4] C. Zhao, S. Zhao, M. Zhao, Z. Chen, C.-Z. Gao, H. Li and Y. Tan: "Secure Multi-Party Computation: Theory, practice and applications", Information Sciences, Volume 476, 2019, Pages 357-372.

[i.5] World Economic Forum White Paper: "Federated Data Systems: Balancing Innovation and Trust in the Use of Sensitive Data", July 2019.

NOTE: Available at http://www3.weforum.org/docs/WEF_Federated_Data_Systems_2019.pdf.

[i.6] World Economic Forum, Insight Report: "Sharing Sensitive Health Data in a Federated Data Consortium Model - An Eight-Step Guide", July 2020.

NOTE: Available at http://www3.weforum.org/docs/WEF_Sharing_Sensitive_Health_Data_2020.pdf.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

federated data collection: data collection scenario where multiple data types are involved and/or multiple organizations jointly collect data of their interest, for instance, to improve data collection efficiency

federated data computing: data computing scenario where multiple organizations work together to solve a data computation task

NOTE: Examples of federated data computing include, but not limited to, federated learning, multi-party computation, and even decentralized Artificial Intelligence/Machine Learning (AI/ML).

federated data discovery and sharing: data discovery and sharing scenario where federated data is discovered by and shared among multiple organizations

federated data management: data management scenario where multiple organizations and/or multiple data types could get involved in each stage of the entire data pipeline or lifecycle and form data federation

NOTE: Examples of federated data management are federated data collection, federated data storing, federated data computing such as federated learning and multi-party computation, federated data sharing, etc.

federated data storing: data storing scenario where multiple organizations participate in storing data, likely, in distributed places

federated learning: distributed machine learning approach where multiple clients and a federated learning server jointly learn an AI model and provide data privacy protection

NOTE: A federated learning process generally works with a few steps:

1) training data are distributed and kept at federated learning clients;
2) a federated learning server coordinates all federated learning clients for them to perform local training and generate local and temporary model updates for each learning round;
3) the federated learning server receives model updates from federated learning clients and aggregate them together to generate a global model;
4) the global model will be sent to federated learning clients for them to perform next round of local training until the goal model converges to the one meeting the expected accuracy.

multi-party computation: secure computation protocol where multiple parties jointly compute a function and guarantees their data privacy

NOTE: In a multi-party computation:

1) multiple parties jointly compute a function over their individual data inputs to get a computation result;
2) each party knows the computation result; and
3) none of parties can learn other parties' data inputs but only knows the computation result.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AI	Artificial Intelligence
ETSI	European Telecommunications Standards Institute
FDDSS	Federated Data Discovery and Sharing Service
FDM	Federated Data Management
FDS	Federated Discovery Service
FL	Federated Learning
FPP	FDM-PDL Proxy
GDPR	General Data Protection Regulation
IIC	Industrial Internet Consortium
IIoT	Industrial Internet of Things
IoT	Internet of Things
IT	Information Technology
LMS	Ledger Messaging Service
LSS	Ledger Storage Service
ML	Machine Learning
MPC	Multi-Party Computation
MSG	Message
PDL	Permissioned Distributed Ledger
TXN	Transaction

4 Use Cases for Federated Data Management

4.1 Introduction of Use Cases

This clause describes some selected federated data management use cases or scenarios, which could be benefited from the use of Permissioned Distributed Ledger (PDL) technology and/or introduce new requirements to PDL technology. As illustrated in Figure 4.1-1, a general data pipeline in federated data management could consist of a set of relatively sequential stages such as data collection, data storing, data computing, data sharing, and data visualization. For each stage, multiple organizations could participate and work together. Each organization could have their own data, for example, generated from ubiquitous devices deployed for different applications such as connected vehicles. In general, a data pipeline (e.g. data pipeline A and data pipeline B) starts with data collection from devices, but it could complete in different places in the networking system. For example, data pipeline B in Figure 4.1-1 stops in edge networks leveraging edge servers for data storing, data computing and data visualization, while data pipeline A ends in the cloud. This clause will not cover the entire data pipeline but focus more on the stages and corresponding scenarios, which are more relevant to PDL technology.

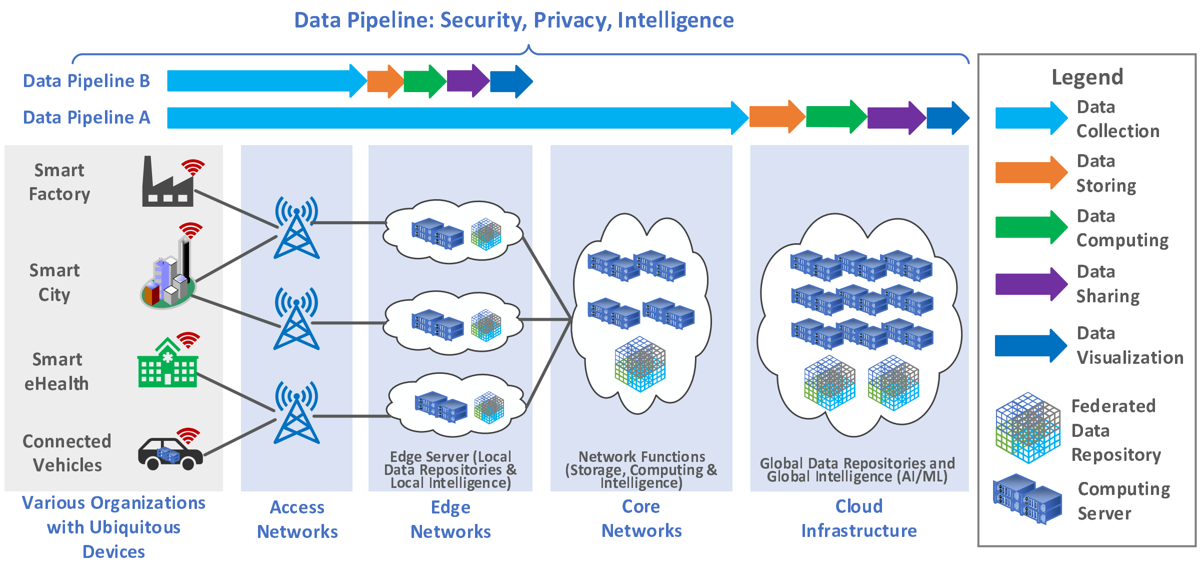
Data Pipeline: Security, Privacy, Intelligence
Data Pipeline B
Data Pipeline A
Smart Factory
Smart City
Smart eHealth
Connected Vehicles
Various Organizations with Ubiquitous Devices
Access Networks
Edge Server (Local Data Repositories & Local Intelligence)
Edge Networks
Network Functions (Storage, Computing & Intelligence)
Core Networks
Global Data Repositories and Global Intelligence (AI/ML)
Cloud Infrastructure
Legend
Data Collection
Data Storing
Data Computing
Data Sharing
Data Visualization
Federated Data Repository
Computing Server

Figure 4.1-1: General Data Pipeline in Federated Data Management

4.2 Federated Data Collection

Our daily lives are surrounded by a variety of sensors and devices. Internet of Things (IoT) technology enables us to leverage these sensors/devices to monitor and measure the physical world in a real-time manner. In many data-driven IoT applications, the first and most important stage is data collection. During data collection, the system can collect data from different devices such as consumer equipment, personal devices, cameras, and wearable health devices; data can also be collected from commercial equipment including security monitoring systems, traffic monitoring equipment, production lines, logistics and supply chain systems, etc. These devices generate different types of data and could belong to and be owned by multiple organizations; the resulted data collection that contains multiple data types and/or relies on multiple organizations is referred to as federated data collection.

Figure 4.2-1 shows an Industrial Internet of Things (IIoT) use case, which includes a few processes such as smart manufacturing, smart logistics, and customer experience monitoring. Multiple scenarios could be involved in each process. For example, the smart manufacturing process could cover product quality control, storage management, onsite energy management, equipment maintenance, etc. All those processes and scenarios need to be monitored in real-time to ensure overall product delivery and product quality. As a result, a large amount of production, logistics and customer experience data are generated at all times and need to be collected. However, in a real-world production environment, manufacturing equipment and Information Technology (IT) systems usually involve multiple manufacturers; in the meantime, a complete manufacturing process could involve different departments or even different companies/organizations. Similarly, during the smart logistics process, products will be transported from factories to customers through multiple intermediate transit places, where multiple organizations are involved as well. All of these facts demonstrate that data collection in IIoT is a complex system and needs multi-party collaboration, which is referred to as federated data collection. Please note that Industrial Internet Consortium (IIC) [i.1] defines more IIoT use cases, which are not limited to Figure 4.2-1. In these use cases being considered, data security could be needed; as a result, data at rest and/or data in transit could be encrypted when there is a risk of data leakage.

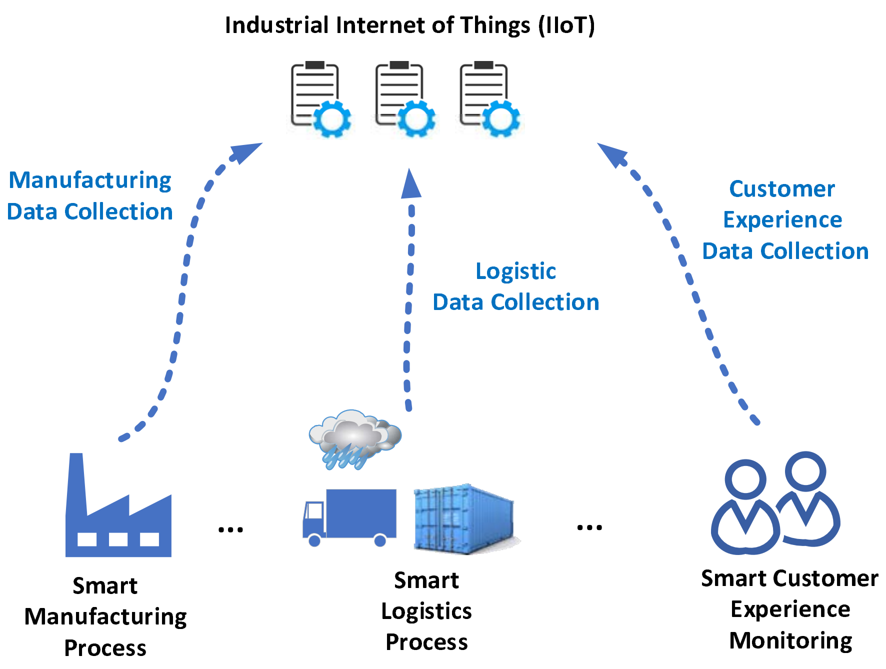
Industrial Internet of Things (IIoT)
Manufacturing Data Collection
Logistic Data Collection
Customer Experience Data Collection
Smart Manufacturing Process
...
Smart Logistics Process
...
Smart Customer Experience Monitoring

Figure 4.2-1: Federated Data Collection for Industrial Internet of Things (IIoT)

4.3 Federated Learning

Traditional Machine Learning (ML) technology is usually centralized, in the sense that:

1) training data is usually collected to be stored at a centralized location such as a centralized database; and

2) learning process is performed at a centralized location such as clouds as well. However, traditional ML could cause data leakage issues, since training data is maintained at a location, different than its original place and likely losing data privacy protection.

As a distributed ML technology and a type of federated data computing, Federated Learning (FL) was to implement a distributed ML model training process by multiple FL participants while still ensuring data privacy, security and legal compliance. Using FL-based mobile keyboard prediction as an example, FL usually consists the following steps:

- Step 1: Mobile phones as FL participants participating in an FL task first download initial training model (i.e. the initial global model) from an FL Server.

- Step 2: Each mobile phone conducts the local training over its local data to train the model and generate its local model (or model update).

- Step 3: After the local model is trained, the mobile phone uploads the encrypted local model update (i.e. gradients) to the FL sever.

- Step 4: The FL server aggregates all local model updates collected from multiple mobile phones to obtain a new/updated global model. The updated global model will be then further sent to each mobile phone for the next round of training (Similar to Step 1).

- Overall, steps 1-4 will be executed for multiple rounds to improve the global model with expected quality and/or other requirements.

From the above process, it can be seen that FL can make full use of the data and computing power of the FL participants. Multiple parties (i.e. participants) can collaborate to build a more robust ML model without sharing/moving their data. This is very important for ML tasks when a strict data law/supervision is enforced. For example, the General Data Protection Regulation (GDPR) in Europe puts forward strict requirements on the storage, use, and transfer of users' private data. Therefore, FL can be used to solve key issues such as data ownership, data privacy, and data access rights in this environment.

Consider a general use case of smart city and smart transportation as shown in Figure 4.3-1:

- In smart city applications, many cameras will be deployed on streets and generate continuous data or data streams. These urban camera data can be used to train an ML model for urban environmental monitoring and predicting. However, uploading all camera data to cloud could be cumbersome or unrealistic. Accordingly, FL is a more feasible and efficient method.
- Similarly, in smart transportation applications, there will be a large number of vehicles driving on the road, and each vehicle will generate massive real-time driving data. These data can be trained to generate many ML models (e.g. to predict which road sections or during which time periods vehicles are most likely to have poor driving behaviour/performance). However, these data are not only large in quantity, but also contain personal privacy information; as a result, it is unwise or inefficient to upload these data to a cloud for centralized processing/training as in traditional ML. FL can be applied in this use case such that a global ML model can be jointly trained by vehicles without uploading driving data from vehicles to cloud.

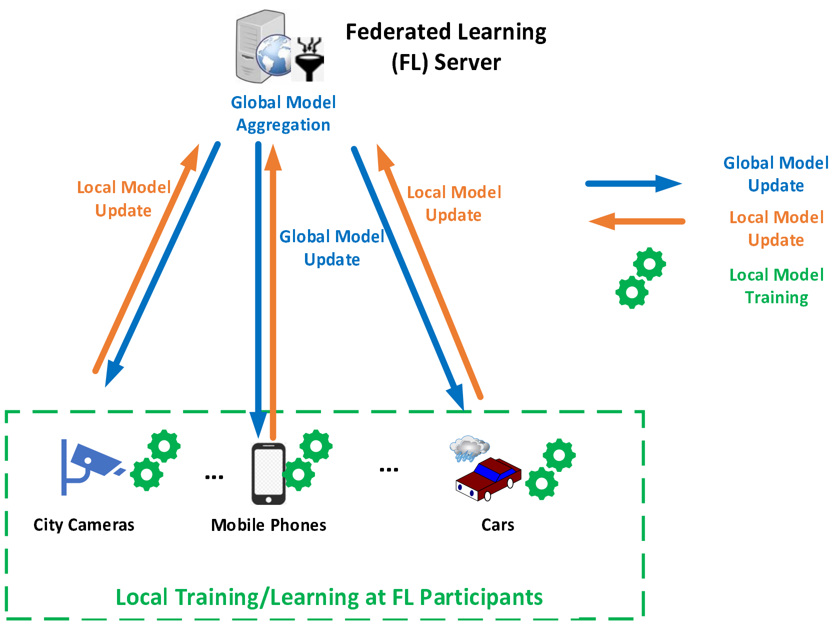

Figure 4.3-1: Federated Learning in Smart City and Smart Transportation

4.4 Multi-Party Computation Use Case

Secure Multi-Party Computation (MPC) was originally introduced in [i.2] in the form of "The Millionaire's Problem". Since then, many advances have been made in both MPC theories and practical MPC deployments [i.3], [i.4].

In a general setting of MPC, there are n parties. Each party *Pi* hosts its own input data *xi*. They want to jointly compute a function to get a result: *result=f(x1, x2, …, xn)* with the requirement that no party can know or deduce input data hosted by other parties. In other words, all parties only know the function and the computed result. Figure 4.4-1 shows such a general MPC structure as an example of MPC use cases, which consists of the following procedures:

- Step 1: Parties encrypt their input data.
- Step 2: Parties exchange their encrypted input data.
- Step 3: One (or multiple) party computes the function over received encrypted input data from other parties to generate a temporary result.
- Step 4: The temporary result is sent to other parties.
- Step 5: One (or multiple) party computes the function over the temporary result to generate the final result.
- Step 6: The final result is sent to other parties.

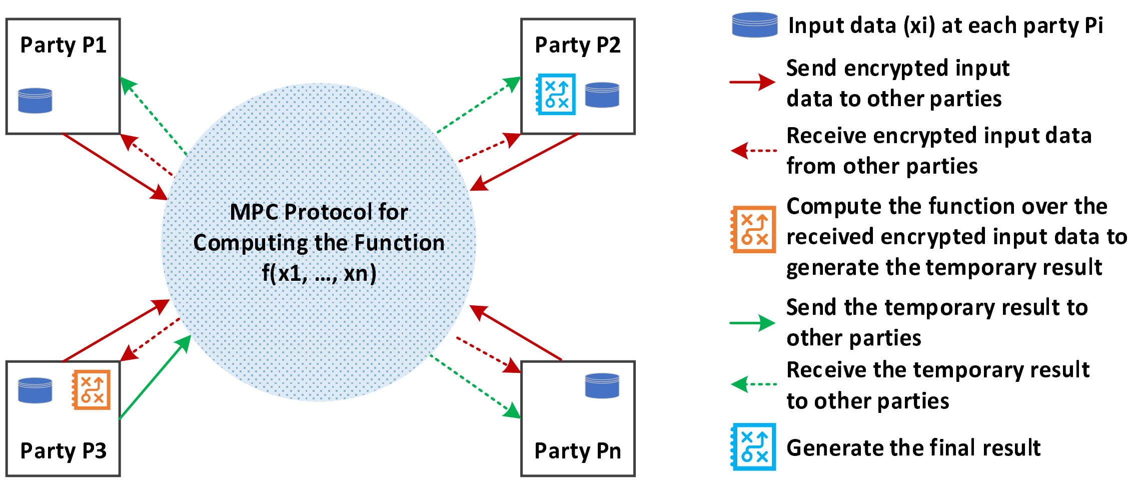
Party P1
Party P2
Party P3
Party Pn
MPC Protocol for Computing the Function f(x1, ..., xn)
Input data (xi) at each party Pi
Send encrypted input data to other parties
Receive encrypted input data from other parties
Compute the function over the received encrypted input data to generate the temporary result
Send the temporary result to other parties
Receive the temporary result to other parties
Generate the final result

Figure 4.4-1: General Multi-Party Computation

4.5 Federated Data Discovery and Sharing

As a critical stage of federated data management pipeline, federated data discovery and sharing refers to the process, where data discovery cannot be solely served by a single organization, but served by multiple organizations. In other words, a federated data discovery request will trigger data lookup operations on data maintained locally by different organizations, and discovery results from each organization will be combined or aggregated as the final result for the federated data discovery request.

Figure 4.5-1 illustrates a federated data discovery scenario, where a user (e.g. a researcher) can discover data (e.g. genomic data) from multiple organizations (e.g. hospitals). In other words, the user's data discovery request will not be served by a single organization, but served by multiple, unnecessarily trusted, organizations [i.5], [i.6]. This scenario consists of the following steps:

- Step 1: A user (e.g. a doctor or a researcher) issues an initial data discovery request to a Federated Data Discovery and Sharing Service ("FDDSS"), which is a logical function and has access to data maintained locally at different organizations. It is assumed that the user knows the address of FDDSS (e.g. through pre-configuration or provisioning).

- Step 2: FDDSS could simply forward the initial data discovery request to organizations (e.g. Organization-1, Organization-2 and Organization-3); alternatively, it could transform the initial data discovery request to multiple transformed data discovery requests and forward each transformed data discovery request to a different organization. Within this step, FDDSS could first authenticate and authorize if the user has the right to leverage the discovery service. Then, FDDSS could enforce certain access control rules limiting data discovery based on access criteria. As an example, access control rules could specify the list of data types or items that are not discoverable.

- Step 3: Each organization receives a separate data discovery request from FDDSS. The organization will authenticate and authorize the data discovery request, look up the data maintained locally against any discovery criteria contained in the data discovery request, and generate discovery result. If any data cannot be discovered (e.g. due to confidential or privacy considerations), the organization could reject the data discovery request and/or exclude such data from the discovery result.

- Step 4: FDDSS receives data discovery results from multiple organizations, aggregates these results, generates an aggregated result, and forwards the aggregated result to the user.

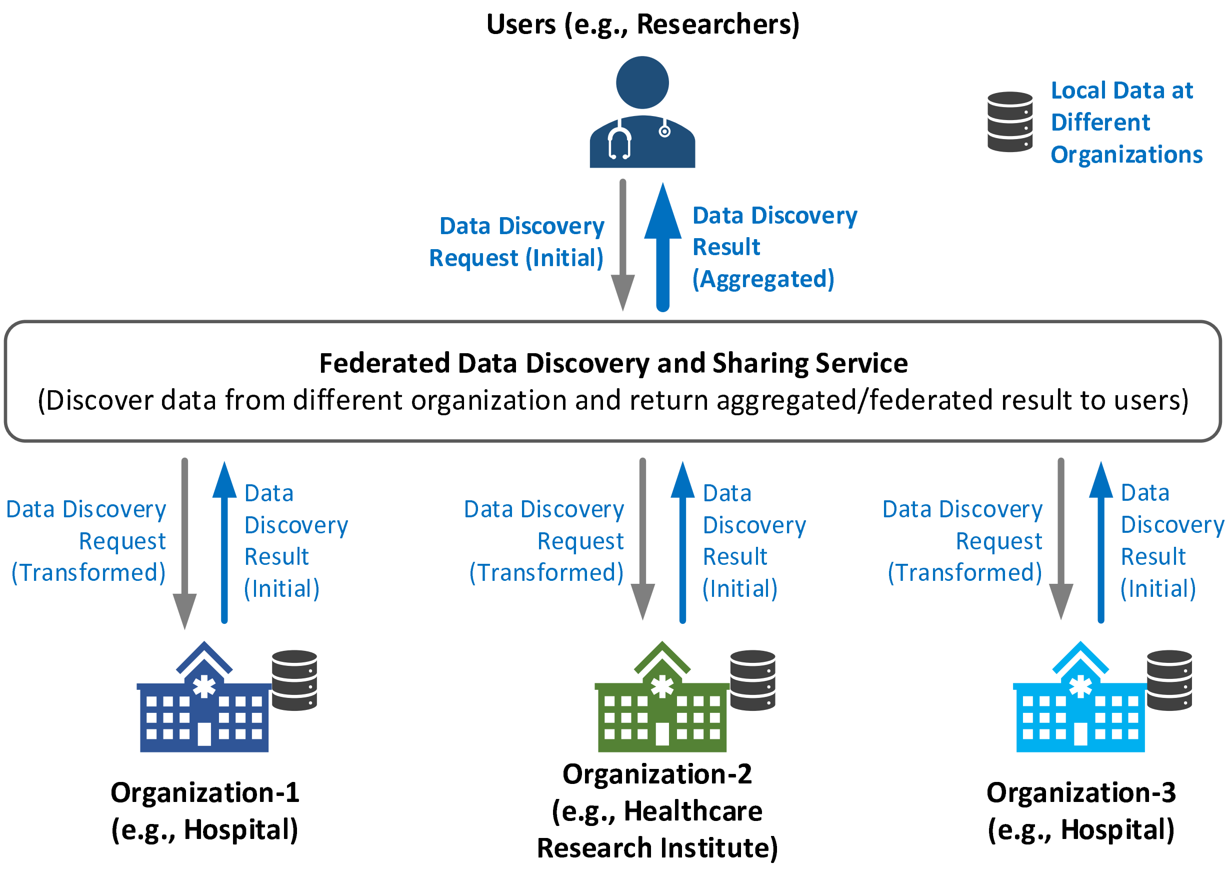
Users (e.g., Researchers)
Local Data at Different Organizations
Data Discovery Request (Initial)
Data Discovery Result (Aggregated)
Federated Data Discovery and Sharing Service
(Discover data from different organization and return aggregated/federated result to users)
Data Discovery Request (Transformed)
Data Discovery Result (Initial)
Data Discovery Request (Transformed)
Data Discovery Result (Initial)
Data Discovery Request (Transformed)
Data Discovery Result (Initial)
Organization-1 (e.g., Hospital)
Organization-2 (e.g., Healthcare Research Institute)
Organization-3 (e.g., Hospital)

Figure 4.5-1: Federated Data Discovery

4.6 Possible Actors in FDM Systems

Dependent on the support applications (e.g. federated data collection, federated learning, federated data discovery and sharing), FDM systems could have various types of data and different kinds of actors such as data source, data host, data consumer, etc.:

- Data Source: The entity such as end devices that generates original data.
- Data Host: The entity that stores and hosts the data. A Data Source could send the original data to a Data Host; if the Data Source maintains the original data locally, it acts as the Data Host as well.
- Data Consumer: The entity that requests to access the data. A Data Consumer requests data from a Data Host and/or a Data Source.

Table 4.6-1: Possible Actors in FDM Systems

Actors		Data Source	Data Host	Data Consumer
Federated Data Collection		Entities (e.g. devices) that generate data	Entities (e.g. servers) that collect data	Entities (e.g. users) that use data
Federated Learning	**Training Data**	FL participants (or other entities that send training data to FL participants)	FL Participants	FL Participants
	Local Model	FL Participants	FL Server and FL Participants	FL Server
	Global Model	FL Server	FL Server and FL Participants	FL Participants and Users
Multi-Party Computation (MPC)		MPC Parties	MPC Parties	MPC Parties
Federated Data Discovery and Sharing		Domain entities of an organization that generates data	Organizations that maintain data locally	Users issuing data discovery and sharing requests

5 Key Issues

5.1 Introduction

This clause describes some key issues, which are related to federated data management use cases as discussed in the clause 4 and can be potentially solved by leveraging PDL technologies.

5.2 Key Issues with Federated Data Collection

5.2.1 Overall Issues with Federated Data Collection

As illustrated in Figure 4.2-1, federated data collection is essentially a distributed system and involves multi-party collaboration, which leads to the following issues:

- The first issue is related to trust. These multiple parties or multiple organizations involved in federated data collection need to build a certain level of trust in order to jointly collect data in a trustworthy manner.
- The second issue is how to guarantee the integrity of the data collected by multiple organizations. For example, in the IIoT use case, logistics data as collected from a shipping truck will probably not be tampered.
- The third issue is how to incentivize multiple organizations to participate federated data collection. For example, customer experience data in the IIoT use case can be very useful for enhancing the manufacturing process. A proper incentive mechanism is needed to encourage customers to contribute their experience data.
- The fourth issue is related to network security. For example, the IIoT use case relies on distributed IoT networks to enable federated data collection. To secure distributed IoT network is crucial.

The above issues could be solved by leveraging PDL technologies. For example, PDL technologies can be used to form a unified ledger infrastructure, which allows various companies, various equipment manufacturers and various logistic companies to achieve more trustful collaboration relationships, which will ultimately ensure the credibility, accountability and transparency of federated data collection in the IIoT use case, and in turn improve the efficiency and reliability of next-generation smart manufacturing. More detailed solutions for PDL-enabled federated data collection will be developed in clauses 6 and 7. In clause 5.2.2, a few specific key issues are elaborated.

5.2.2 How to efficiently and concurrently collect data and store data collection records in PDL?

As described in federate data collection use cases, IIoT data will be transmitted and collected from various IIoT devices (e.g. factory devices, shipping vehicles) to data collection service in the cloud. This is usually done without using or interacting with PDL system. In other words, many applications currently use regular communications (i.e. off-chain communications without leveraging PDL) for normal data transmission and collection, and only leverage distributed ledgers for recording selected data collection histories (i.e. on-chain communications using PDL). This approach has two problems:

- it involves two separate processes (i.e. off-chain and on-chain communications) and is inefficient in terms of overall overhead. However, in some cases, it could be more desired to leverage PDL to support both on-chain and off-chain communication; and
- applications need to directly deal with PDL, which might not be affordable especially when applications are hosted on resource-constrained IIoT devices. As a result, new functionalities such as intermediary entities can be designed to help applications to interact with the designated PDL on behalf of applications; such intermediary entities are logical entities, which could be co-located with servers, gateways, and/or other type of network nodes.

When considering leveraging PDL systems to transmit/convey and record original application messages (or data) simultaneously, a few issues need to be considered to improve the efficiency of such concurrent data transmission and recording:

- For example, when a sender application sends an application message to a receiver application through a selected PDL chain, the selected PDL chain needs to be able to route the application message through appropriate PDL nodes and eventually arriving at the receiver application. Given the massive number of applications (e.g. hosted by IIoT devices), it is inefficient and impractical to let PDL nodes to identify application messages and their routing for these applications. In addition, an application could continuously send many application messages through a PDL chain, while another application could only send sporadic application messages through the PDL chain; different approaches will probably be designed for such applications, which have different message generation and transmission needs.
- Another consideration is how to enable that these applications can flexibly and efficiently use various PDL chains. An application could need to use different type of PDL chains but do not have adequate capability to discover any available PDL chains and maintain their information. Plus, an application could even not directly interact with any PDL nodes in order to reduce its complexity.
- In addition, how to efficiently transform and adapt application messages to the format of PDL transactions needs to be considered for several reasons:
    1) it could be required that the application message content cannot be seen by all other entities but entities involved in the same application;
    2) the message content also will probably be transparent to PDL nodes, but PDL nodes need to know some metadata (e.g. which messages are from which applications so that messages from different applications could be handled by PDL nodes differently based on their needs); and
    3) the size of a single application message could be too small and to contain it in a PDL transaction could cause high overhead.

5.3 Key Issues with Federated Learning

5.3.1 Overall Issues with Federated Learning

In smart city and smart transportation use case as illustrated in Figure 4.3-1, FL is used to learn AI models from distributed camera data, mobile phone data and vehicle data. Although the use of FL does not need to move local data away from FL participants (e.g. cameras, mobile phones, vehicles), traditional FL still introduces a few issues:

- First, many FL participants are not from the same organization and do not trust each other, which makes effective collaboration and coordination between them difficult especially in a fully distributed scenario.
- Second, a FL participant could have useless and even malicious local data, which cannot help training a good local model.
- Third, a FL participant could inject a bad local model, which will impact the aggregated global model. The integrity of local model and global mode also needs to be guaranteed and accountable.
- Fourth, a sufficient number of FL participants are required to guarantee the quality of the global model. The issue to how to incentivize FL participants with good local data to participate FL.
- Fifth, local models generated in each FL round could provide insights on the whole FL and enable explainable AI. But it relies on how the integrity and accountability of local models can be guaranteed.
- Last but not the least, the FL aggregation server is still a single-point-of-failure. If the FL aggregation server fails, the global model will never be appropriately generated.

PDL technologies help to solve and/or mitigate the above issues. For example, local models can be stored in the ledger for future traceability and explanation purposes. Also, smart contracts can be leveraged to encourage FL participants to actively cooperate and contribute their local data and learning capabilities. More detailed solutions for PDL-enabled federated learning will be developed in clauses 6 and 7. In clause 5.3.2, a few specific key issues are elaborated.

5.3.2 How to efficiently store FL-related data in PDL?

In reality, various FL tasks can be initiated and multiple FL participants in a given FL task could not be affiliated with the same organization. In other words, those FL participants could not trust or know each other, and they could join a specific FL task randomly. Those FL participants usually do not have formal business collaboration relationships with the FL task initiator and therefore the FL participants do not have obligations for contributing themselves to an FL training process. Given that, all the data related to FL training can be recorded using PDL since it can enable the traceability and accountability of the FL training process among those untrusted FL participants. For instance, when certain FL participants are malicious nodes and uploaded many bad local model updates, the FL records in the PDL can be used to identify those malicious behaviours.

In the meantime, massive data such as training data and FL models can be generated by the FL participants and the FL server during the whole FL training process:

- For example, during the FL training process, local model updates are produced by FL participants. Once the FL training process is completed, the final global model is generated by the FL server.
- Other types of data could include training progress and performance-related data/statistics (e.g. how long did an FL participant take for completing a local training during each training round? how much computing resources were allocated for the local training?).

All types of these data can be recorded in PDL chains in order to support accountability and traceability (e.g. to support rollback operations if an FL training process needs to be restarted from a certain point in order to eliminate a bad effect made by a malicious FL participant). As such, how to effectively store them in PDL remains a major design challenge, for example, based on the following design considerations:

- The type of FL-related data to be stored to PDL systems will probably be determined, for instance, based on the availability and capability of PDL systems. For a specific FL training process, it might be determined that only the final global model and/or the list of FL participants will be stored to a PDL chain to reduce overhead to PDL system.
- FL participants will probably be appropriately instructed or notified of the type of FL-related data that they need to store to PDL chains. Note that the entity that creates/initiates a specific FL training process could now know or trust involved FL participants. Note that the size of FL models (either local models or the global model) could be in tens of megabytes and even larger. An FL model could be stored with a full version or with smaller tailored versions. As such, a critical issue is how to prepare the FL model in an appropriate version based on PDL capabilities and/or constraints before storing it onto a designated PDL chain.
- An intermediary service or function can be designed to help the interaction between the FL entities and PDL systems. Otherwise, all the FL tasks/applications have to implement their own solutions for interacting with each PDL system, which increases additional development complexity and burden for FL application developers.

5.4 Key Issues with Federated Data Discovery and Sharing

As illustrated in Figure 4.5-1, multiple organizations and distributed data are involved in FDDSS, which leads to the following issues:

- The first issue is related to trust. A user from an organization X could issue a discovery request to a different organization Y. In the meantime, organization X could provide access to its data (including the data obtained from organization Y) to other organizations (e.g. organization Z), practically giving organization Z the access to organization Y's information. Therefore, it is critical to have a mechanism to enable and build mutual trust among these untrusted organizations.
- The second issue is how to incentivize multiple organizations to make their data discoverable and sharable to other organizations. An organization providing data could be rewarded or could collect credits, while other organizations discovering/utilizing the data could make contributions or pay credits for the data they consume.
- The third issue is guaranteeing the quality of the discovered data. Each organization could maintain and provide the same type of data or similar type of data with different quality (e.g. date can be presented in year only or in year-month-day format for higher accuracy). This can be resolved by using a tool to identify data with appropriate quality satisfying the user's discovery criteria.

- The fourth issue is related to privacy and access control. For example, the data maintained locally by an organization could only be discoverable by certain users/organizations. In another example, an organization could need to hide data source information although it is willing to make the data discoverable.

PDL technologies can be leveraged to solve or mitigate these issues. For example, any data discovery and sharing record could be recorded in PDL permanently; as such, trust relationship among all participating organizations can be automatically established. In addition, smart contracts can be used to enable incentivized interactions between organizations providing data and organizations discovering data. Furthermore, PDL governance could manage access control of federated data discovery and sharing. More detailed solutions for PDL-enabled federated data discovery and sharing will be developed in clauses 6 and 7.

6 Architecture for PDL-based Federated Data Management

6.1 Introduction

This clause describes PDL-based Federated Data Management architecture including primary functional components. According to key issues as described in clause 5, the following requirements could be considered for designing the architecture for PDL-based federated data management:

- PDL can be leveraged to build trust relationships among untrusted participants/parties/organizations involved in federated data management.
- Smart contracts can be leveraged as an effective mechanism to incentivize participants/parties/organizations to participate in federated data management and to enable autonomous interactions among them.
- PDL can be leveraged not only for recording data, but also a mechanism to propagate/transmit data among participants/parties/organizations involved in federated data management.

6.2 Architecture

In the context of PDL-based Federated Data Management (FDM), there are two separate systems, namely PDL system and FDM applications. To leverage PDL to solve key issues as described in previous clause and eventually enable PDL-based FDM, these two systems need to interact and interwork with each other.

Figure 6.2-1 illustrates a general proxy-based solution to interwork FDM system and PDL system, where FDM-PDL Proxy (FPP) is included as a logical entity to connect both systems. Via FPP, FDM applications (e.g. federated data collection, federated learning, federated data discovery and sharing), which could be a data source or a data consumer, can access PDL systems, for instance, to store FDM-related data (e.g. operation records) to a PDL chain. FPP can provide the following functions:

- find appropriate PDL chains from PDL system for an FDM application based on its requirements;
- interact with PDL system on behalf of an FDM application;
- buffer and send requests (e.g. to create a transaction) from an FDM application to PDL system;
- buffer and forward notifications and/or responses from PDL system to an FDM application; and
- knows how to talk to FDM applications and how to talk to PDL systems (e.g. ledgers); and
- FPP is a logical entity, which can be deployed as a service function or as a part of PDL system in a distributed manner. For example, if FPP needs to implement PDL-related governance and intelligence, it can be implemented as a distributed function within PDL systems.

FPP can provide the following benefits:

- alleviate overheads at both FDM and PDL; and

- provide data access control and security between FDM and PDL.

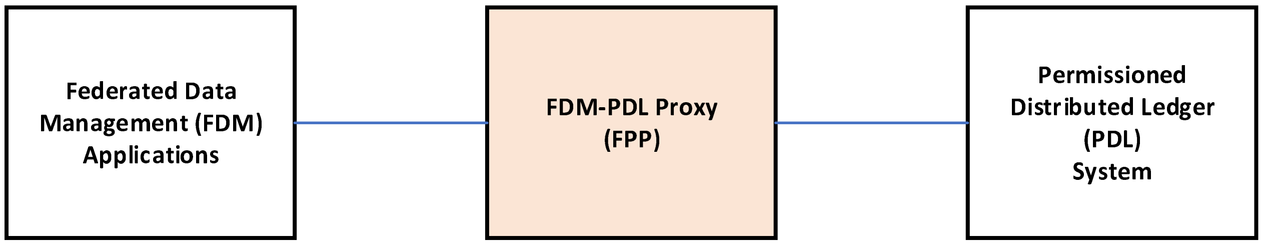
Federated Data Management (FDM) Applications
FDM-PDL Proxy (FPP)
Permissioned Distributed Ledger (PDL) System

Figure 6.2-1: PDL-based Federated Data Management via an FDM-PDL Proxy

There could be multiple FPPs between FDM system and PDL system. As an example, Figure 6.2-2 shows an extended solution, where FDM Entity-1 and FDM Entity-2 interact with PDL system through multiple and different FPP (i.e. via FPP-1 and FPP-2, respectively). The following scenarios and operations can be supported via FPP-1 and FPP-2.

Scenario 1: FDM Entity-1 (e.g. an FL participant) needs to record an FDM message (e.g. a local model update) to PDL system. After it is done, FDM-Entity-2 (e.g. the FL server) expects to receive a notification from PDL system:

- FDM Entity-1 creates an FDM message MSG1 and sends it to FPP-1.
- FPP-1 transforms the FDM message MSG1 to a PDL transaction TXN1. FPP-1 sends the PDL transaction TXN1 to PDL system.
- The PDL transaction TXN1 will be propagated through the PDL system, so that all PDL nodes will receive it and eventually the PDL transaction TXN1 will be included and stored in the ledger.
- After the PDL transaction TXN1 is stored in the ledger, PDL system could send a notification to FPP-2 to indicate the successful inclusion of TXN1; FPP-2 could forward the notification to FDM Entity-2.

Scenario 2: FDM Entity-1 (e.g. an IIoT device) leverages PDL system to transmit an FDM message (e.g. IIoT sensory reading) to FDM Entity-2 (e.g. IIoT data collection server), while storing this transmission record to ledgers:

- FDM Entity-1 creates an FDM message MSG1 and sends it to FPP-1.
- FPP-1 transforms the FDM message MSG1 to a PDL transaction TXN1. FPP-1 sends the PDL transaction TXN1 to PDL system.
- The PDL transaction TXN1 will be propagated through the PDL system, so that all PDL nodes will receive it.
- A PDL node forwards the PDL transaction TXN1 to FPP-2 and stores a record of this event to ledgers.
- FPP-2 receives the PDL transaction TXN1 and recovers the contained message MSG1.
- FPP-2 forwards the message MSG1 to FDM Entity-2.

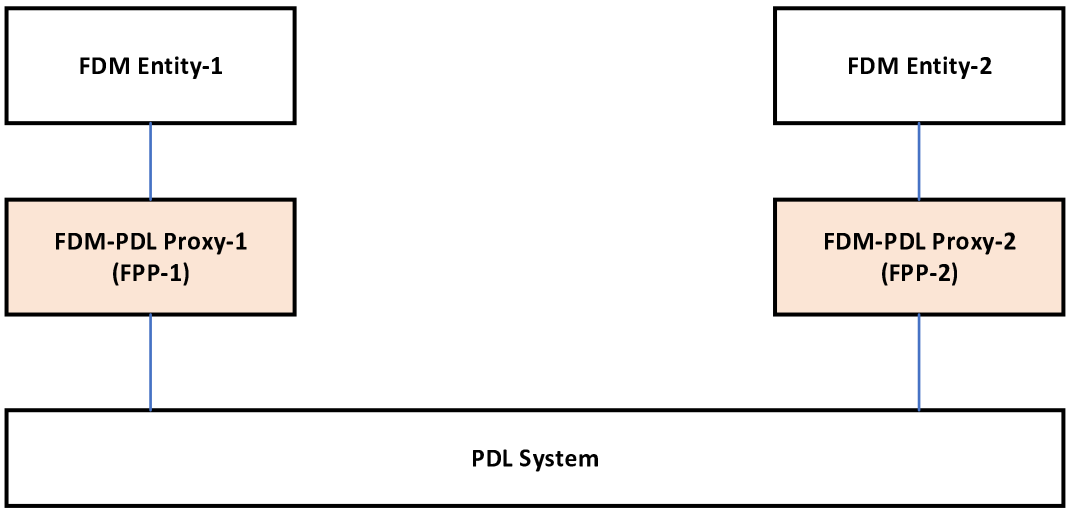

Figure 6.2-2: PDL-based Federated Data Management with Multiple FDM-PDL Proxies

7 Key Solutions

7.1 Solutions for PDL-based Federated Learning

To solve PDL-based federated learning issues as described in clause 5.3, FL entities (i.e. FL task initiators, FL participants and FL servers) generally need to interact with PDL systems, for example, to store FL-related data onto PDL chains. To make this process more efficient and alleviate extra burden to FL entities, a logical entity, referred to as Ledger Storage Service (LSS), is proposed as a part of FDM-PDL Proxy (FPP). In fact, LSS is a value-added service to assist FL entities in leveraging PDL with minimum effort.

Basically, an FL entity acting as a LSS client just needs to specify high-layer requirements to LSS regarding how an FL task intends to leverage PDL systems such as:

1) what kinds of information will be stored onto PDL chains; and

2) whether the full version and/or tailored versions of FL model updates will probably be stored onto PDL chains.

Once those high-level requirements are conveyed to LSS, LSS needs to handle all the low-layer details in order to interact with PDL systems such as:

1) to decide which data is to be stored in which specific PDL chain; and

2) to determine whether a new PDL chain needs to be created. In other words, the application developers of FL applications need to focus on their business logic and all the interactions with PDL systems will be offloaded to and assisted by LSS.

In addition, LSS needs to figure out which FL participants are involved, and then contact each of FL participants on behalf of LSS clients, in order to convey corresponding instructions to those FL participants (e.g. what information needs to be put inside a PDL transaction, in what PDL transaction format, and stored in which specific PDL chain, etc.).

Also, LSS needs to make sure those FL participants have the appropriate privileges to manipulate the desired PDL chain (e.g. adding new blocks to a specific PDL chain). Accordingly, FL participants only need minimum effort to leverage PDL. In addition, in the case where a tailored version of FL model needs to be produced and stored in PDL, LSS needs to advise FL participants about what type of desired tailored operations will probably be conducted by the FL participants or if the tailoring operation needs to be done by LSS on behalf of FL participants.

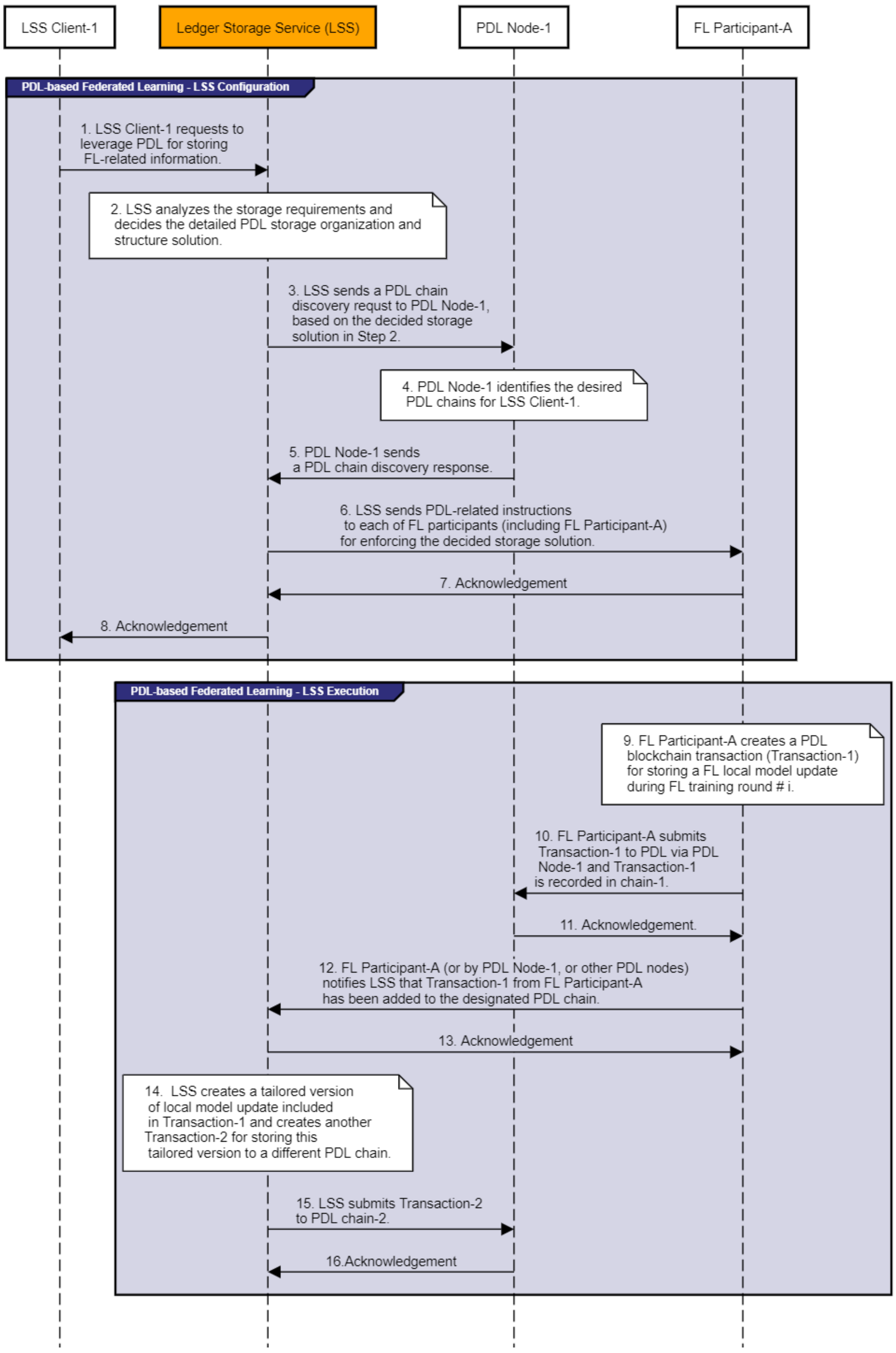
LSS Client-1
Ledger Storage Service (LSS)
PDL Node-1
FL Participant-A
PDL-based Federated Learning - LSS Configuration
1. LSS Client-1 requests to leverage PDL for storing FL-related information.
2. LSS analyzes the storage requirements and decides the detailed PDL storage organization and structure solution.
3. LSS sends a PDL chain discovery requst to PDL Node-1, based on the decided storage solution in Step 2.
4. PDL Node-1 identifies the desired PDL chains for LSS Client-1.
5. PDL Node-1 sends a PDL chain discovery response.
6. LSS sends PDL-related instructions to each of FL participants (including FL Participant-A) for enforcing the decided storage solution.
7. Acknowledgement
8. Acknowledgement
PDL-based Federated Learning - LSS Execution
9. FL Participant-A creates a PDL blockchain transaction (Transaction-1) for storing a FL local model update during FL training round # i.
10. FL Participant-A submits Transaction-1 to PDL via PDL Node-1 and Transaction-1 is recorded in chain-1.
11. Acknowledgement.
12. FL Participant-A (or by PDL Node-1, or other PDL nodes) notifies LSS that Transaction-1 from FL Participant-A has been added to the designated PDL chain.
13. Acknowledgement
14. LSS creates a tailored version of local model update included in Transaction-1 and creates another Transaction-2 for storing this tailored version to a different PDL chain.
15. LSS submits Transaction-2 to PDL chain-2.
16.Acknowledgement

Figure 7.1-1: Procedure of PDL-based Federated Learning

A detailed procedure for leveraging LSS to enable PDL-based federated learning is illustrated in Figure 7.1-1, which has the following steps:

Precondition: LSS Client-1 such as the FL task initiator of FL Task-1 has the management privilege for a specific FL Task-1. There is a PDL system in which PDL Node-1 is one of the PDL nodes such that LSS and/or FL participants can interact with PDL Node-1 for conducting PDL-related operations. In addition, there are multiple FL participants (e.g. FL Participant-A) involved in FL Task-1, and FL Participant-A is just one of them.

Step 1: LSS Client-1 sends a request to LSS to store certain information about FL Task-1 onto PDL chains. In this request, LSS Client-1 could specify its storage requirements regarding what kinds of information about FL Task-1 will probably be stored in PDL and other high-level storage needs. For example, the parameters carried in this request could include:

1) the identifier of FL Task-1;
2) the list of involved FL participants;
3) the type of FL-related information (e.g. local/global model update) to be stored onto PDL chains;
4) whether the full version or tailored version of FL models need to be stored onto PDL chains; and
5) the frequency or rate to store FL-related information to PDL chains.

Step 2: LSS verifies whether FL Task-1 is a valid FL task and make sure LSS Client-1 has the corresponding privileges for managing FL Task-1. LSS then analyses the storage requirements received in Step 1; it decides the detailed PDL storage organization and structure solution. As an example, a PDL storage organization and structure solution for FL Task-1 could specify the following details:

- One PDL chain is needed for storing a full version of the global model updates during each FL training round.
- One PDL chain is needed for storing a tailored version of global model updates during each FL training round.
- One PDL chain is needed for storing the full version of local model updates. However, a given FL participant only needs to store a full version of the local model update for every 5 training rounds.
- One PDL chain is needed for storing the tailored version of local model updates. A given FL participant can store a tailored version of the local model update for every FL training round.

Step 3: LSS sends a PDL chain discovery request to PDL Node-1 based on the decision in Step 2. In addition, LSS collects other useful information from PDL system (e.g. the involved FL participants of FL Task-1 if such information is available in the PDL system). LSS also conveys certain information or configurations to the PDL nodes such that the involved FL participants have the access privileges to operate desired PDL chains.

Step 4: PDL Node-1 identifies the desired PDL chains and conducts the needed configurations as requested by LSS.

Step 5: PDL Node-1 sends an acknowledgement, along with the identified PDL chains (e.g. chain_ID).

Step 6: LSS sends PDL-related instructions to each of FL participants (e.g. the FL Participant-A) for enforcing the decided storage solution. A PDL-related instruction could include:

1) PDL node access information such as node address;
2) PDL transaction format or template;
3) the speed of transactions that FL Participant-A can generate and send to the desired PDL chain; and
4) the type of FL-information that FL Participant-A contains in PDL transaction.

Step 7: FL participants (e.g. the FL Participant-A) sends an acknowledgement to LSS.

Step 8: LSS sends an acknowledgement to LSS Client-1, indicating that FL Task-1 is now ready for storing data in PDL system.

Step 9: FL Participant-A generates a local model update for the current FL training round #*i*. Based on the configuration, FL Participant-A knows that the full version of the local model update will be stored in PDL chain-1.

Accordingly, it creates a PDL transaction (e.g. Transaction-1) for storing a full version of this local model update based on the transaction format of PDL chain-1.

Step 10: FL Participant-A submits Transaction-1 to PDL chain-1 via PDL Node-1. Accordingly, after a certain consensus process, Transaction-1 is recorded in PDL chain-1.

Step 11: PDL Node-1 sends an acknowledgement once Transaction-1 is recorded in PDL chain-1.

Step 12: FL Participant-A (or other entities, e.g. the PDL nodes) notifies LSS that a new local model update generated by FL Participant-A is available.

Step 13: PDL Node-1 sends an acknowledgement for the notification.

Step 14: LSS obtains the new local model update. According to the configuration, LSS knows that it needs to conduct a model tailoring operation. Accordingly, LSS creates a tailored version of the local model update included in Transaction-1, which will be stored in PDL chain-2 by creating another PDL transaction (e.g. Transaction-2).

Step 15: LSS submits Transaction-2 to PDL chain-2. As a result, Transaction-2 will be recorded in PDL chain-2 after a consensus process.

Step 16: PDL Node-1 sends an acknowledgement to LSS once Transaction-2 is recorded in PDL chain-2.

7.2 Solutions for PDL-based Federated Data Collection

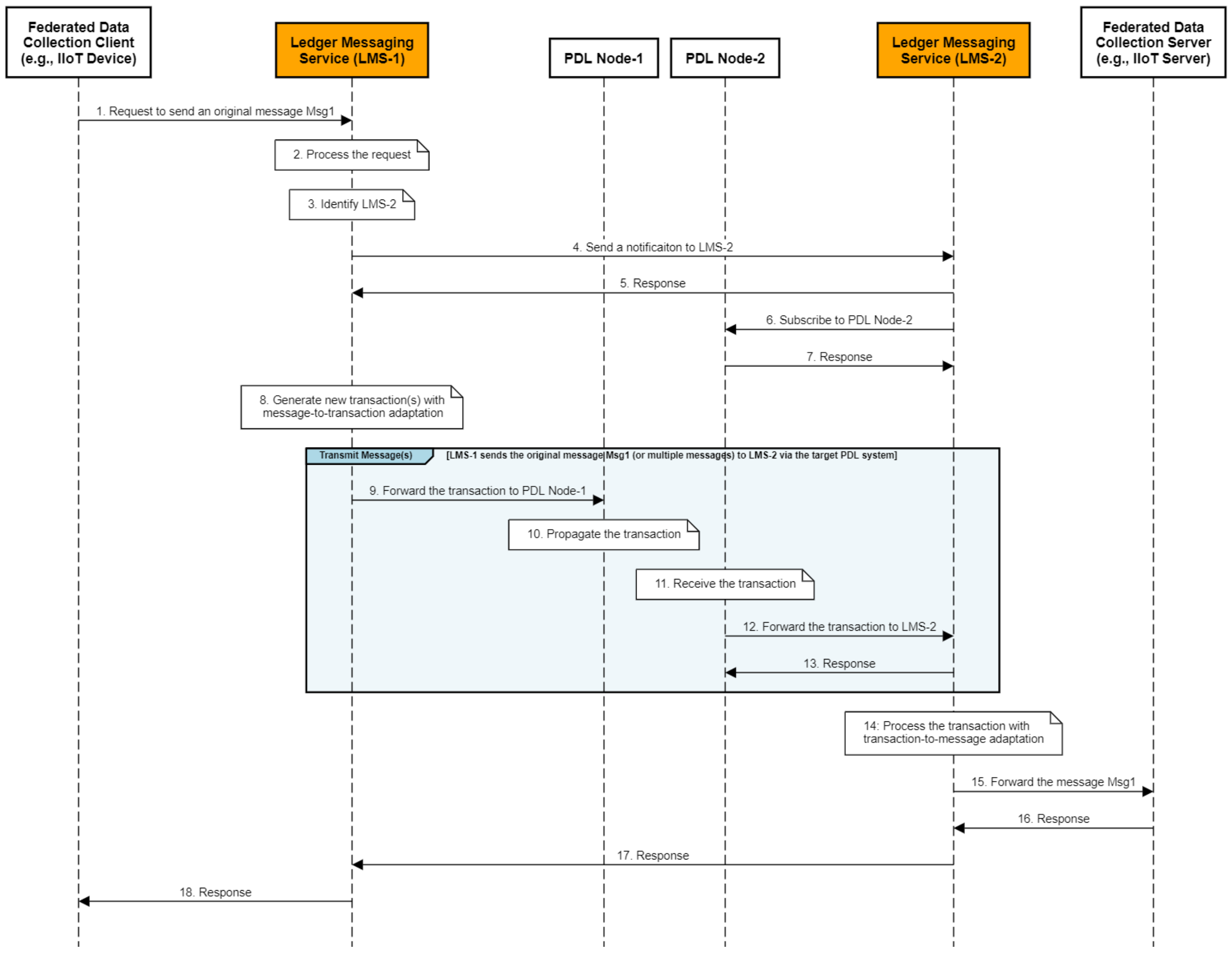
Federated Data Collection Client (e.g., IIoT Device)
Ledger Messaging Service (LMS-1)
PDL Node-1
PDL Node-2
Ledger Messaging Service (LMS-2)
Federated Data Collection Server (e.g., IIoT Server)
1. Request to send an original message Msg1
2. Process the request
3. Identify LMS-2
4. Send a notificaiton to LMS-2
5. Response
6. Subscribe to PDL Node-2
7. Response
8. Generate new transaction(s) with message-to-transaction adaptation
Transmit Message(s)
[LMS-1 sends the original message Msg1 (or multiple messages) to LMS-2 via the target PDL system]
9. Forward the transaction to PDL Node-1
10. Propagate the transaction
11. Receive the transaction
12. Forward the transaction to LMS-2
13. Response
14: Process the transaction with transaction-to-message adaptation
15. Forward the message Msg1
16. Response
17. Response
18. Response

Figure 7.2-1: Procedure of PDL-based Federated Data Collection

A detailed procedure for leveraging Ledger Messaging Service (LMS) to enable PDL-based federated data collection is illustrated in Figure 7.2-1, where a federated data collection client needs to transmit a data (i.e. Msg1) to a federated data collection server via a PDL network. PDL Node-1 and PDL Node-2 are two PDL nodes, which connect LMS-1 and LMS-2, respectively; the PDL network could have more PDL nodes that are responsible for propagating the data (i.e. Msg1) from PDL Node-1 to PDL Node-2. The procedure consists of the following steps:

Step 1: A federated data collection client sends an original message Msg1 to LMS-1. Msg1 contains the data to be transmitted to a federated data collection server via a target PDL network.

Step 2: The LMS-1 receives Msg1. LMS-1 could first look up any applicable PDL policy rules as maintained locally. Based on any found PDL policy rule, LMS-1 authenticates and authorizes if the federated data collection client is allowed to leverage the target PDL network to send Msg1 to the federated data collection server. Based on these PDL policy rules, LMS-1 could also determine an appropriate type/format for the PDL transaction to be created at Step 6 if it is not indicated in Step 1 or the type indicated in Step 1 is not allowed.

Step 3: LMS-1 could need to find LMS-2, for example, based on the address or identifier of the federated data collection server. LMS-1 could have been provisioned with the address of LMS-2; thus, this step can be skipped.

Step 4: Optionally, LMS-1 sends a notification to LMS-2 indicating LMS-2 will be receiving Msg1 via PDL Node-2.

Step 5: LMS-2 sends a response to LMS-1 as a confirmation to Step 4.

Step 6: LMS-2 subscribes to PDL Node-2 for receiving any transactions from LMS-1.

Step 7: PDL Node-2 sends a response to LMS-2 as a confirmation to Step 6.

Step 8: LMS-1 generates a new PDL transaction Txn1 according to the transaction type as determined in Step 2. Txn1 contains Msg1.

Step 9: LMS-1 sends the generated PDL transaction Txn1 to PDL Node-1.

Step 10: PDL Node-1 propagates Txn1 through the target PDL network.

Step 11: PDL Node-2 receives Txn1 from the target PDL network as the result of transaction propagation.

Step 12: After Txn1 is validated and successfully stored to the ledger (e.g. as a result of PDL consensus process), PDL Node-2 forwards Txn1 to LMS-2.

Step 13: LMS-2 sends a response to PDL Node-2 as a confirmation to Step 12.

Step 14: LMS-2 receives Txn1 and performs transaction-to-message adaptation to recover the original Msg1.

Step 15: LMS-2 forwards the original Msg1 to the federated data collection server.

Step 16: The federated data collection server sends a response to LMS-2 indicating the successful receival of Msg1.

Step 17: LMS-2 sends a response to LMS-1 directly using off-chain communications.

Step 18: LMS-1 forwards the response to the federated data collection client.

7.3 Solutions for PDL-based Federated Data Discovery and Sharing

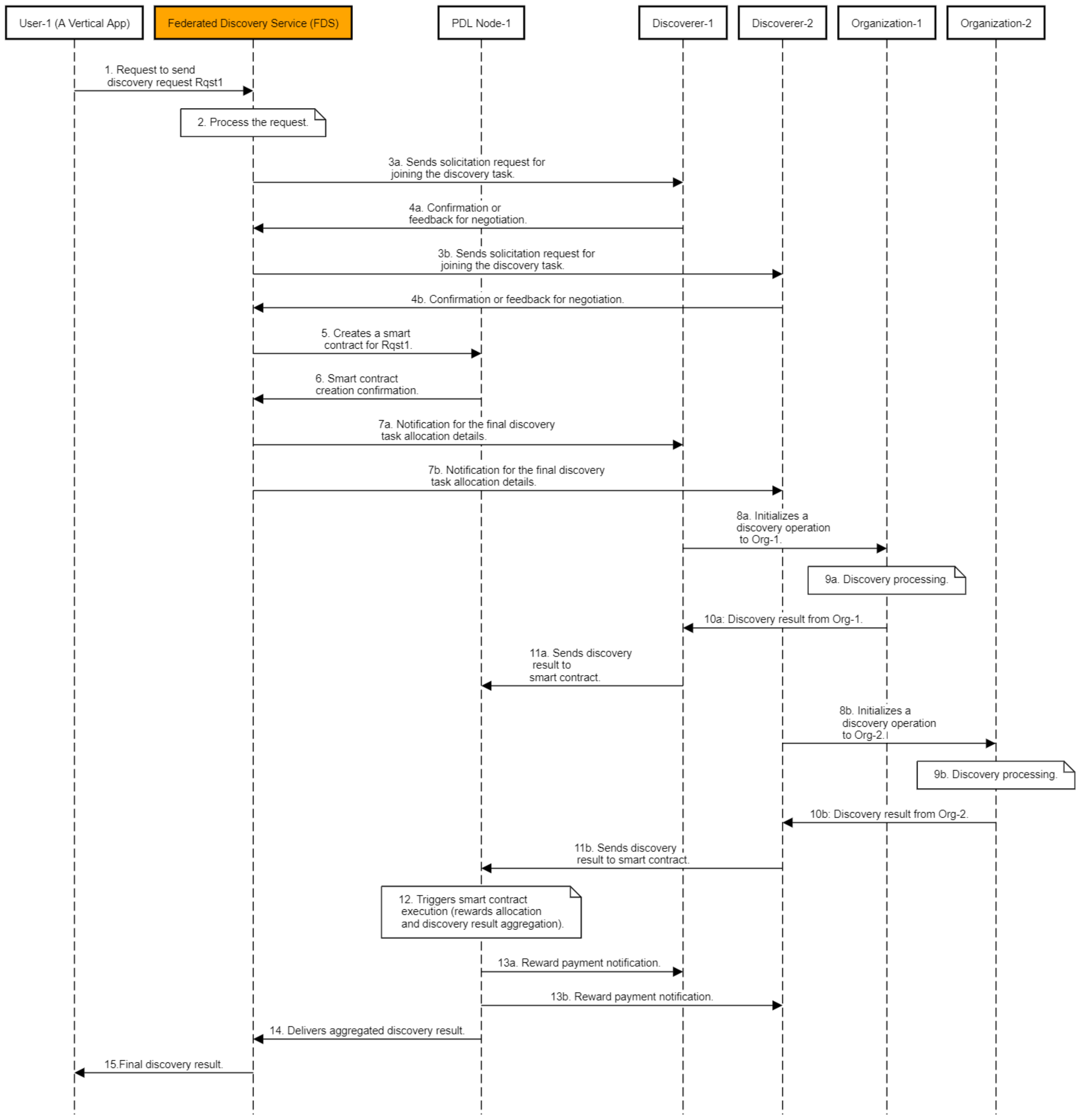
User-1 (A Vertical App)
Federated Discovery Service (FDS)
PDL Node-1
Discoverer-1
Discoverer-2
Organization-1
Organization-2
1. Request to send discovery request Rqst1
2. Process the request.
3a. Sends solicitation request for joining the discovery task.
4a. Confirmation or feedback for negotiation.
3b. Sends solicitation request for joining the discovery task.
4b. Confirmation or feedback for negotiation.
5. Creates a smart contract for Rqst1.
6. Smart contract creation confirmation.
7a. Notification for the final discovery task allocation details.
7b. Notification for the final discovery task allocation details.
8a. Initializes a discovery operation to Org-1.
9a. Discovery processing.
10a: Discovery result from Org-1.
11a. Sends discovery result to smart contract.
8b. Initializes a discovery operation to Org-2.
9b. Discovery processing.
10b: Discovery result from Org-2.
11b. Sends discovery result to smart contract.
12. Triggers smart contract execution (rewards allocation and discovery result aggregation).
13a. Reward payment notification.
13b. Reward payment notification.
14. Delivers aggregated discovery result.
15.Final discovery result.

Figure 7.3-1: Procedure for PDL-based Collaborative Federated Data Discovery

In the federated data discovery scenario, data could be stored in different locations and belong to different organizations. A given application or a User-1 issues a discovery request for discovering desired data, which could be owned and located in different organizations. In order to process or serve the discovery request from User-1, a federated discovery processing is needed, i.e. the data discovery needs to be conducted in multiple organizations. In order to do so, third-party entities acting as discoverers can contribute to serving this discovery request but each of the entities could only have the capability for accessing or conducting discovery in one or more organizations; in other words, a single discoverer could not have the full capability for identifying the needed data among all the potential organizations. For example, a Discoverer-1 could belong to an Organization-1 so that Discoverer-1 could have the discovery privilege in Organization-1. Alternatively, a Discoverer-2 could not belong to Organization-1, but Discoverer-2 is a domain administrator such that Organization-1 could also grant discover privilege to Discoverer-2. In addition, different discoverers could not know and trust each other at all.

Figure 7.3-1 illustrates a collaborative federated data discovery process, where a smart contract is leveraged to build trust between different untrusted discoverers/organizations in order to make them work collaboratively. In particular, the smart contract has the following usage:

- The user-1 could pay a service fee for its discovery request and such a service fee could be deposited in the smart contract. With this service fee, the discoverers (e.g. Discoverer-1 and Discoverer-2) could have the incentive to contribute to the discovery processing and they do not have to worry that the User-1 can refuse to pay the service fee after the User-1 obtains the discovery result (yielded by discoverers).
- In the meantime, the smart contract could also specify how the service fee will be allocated among multiple discoverers. For example, the discoverer producing high-quality discovery results or making more discovery processing effort could get a higher portion of service fee as rewards.

Pre-condition: A Federated Discovery Service (FDS) is available in the system for supporting federated discovery requests. The FDS can interact with a PDL system for creating a smart contract; alternatively, the FDS can be a part of the PDL system. There are multiple third-party entities (e.g. Discoverer-1 and Discoverer-2) that are willing to act as discoverers and participate in federated discovery processing. In particular, Discoverer-1 only has the discovery and access privilege for conducting discovery within Organization-1 while Discoverer-2 only has the discovery privilege for conducting discovery within Organization-2.

Step 1: User-1 has a certain application need and intends to identify some interesting data. User-1 intends to leverage the FDS since the desired data could reside in different organizations and be stored in different locations/nodes. User-1 sends a request (Rqst1) to the FDS to indicate desired data types and the service fee it is willing to pay.

Step 2: The FDS receives the request sent from User-1. The FDS first decides whether User-1 has the right for asking the FDS to conduct data discovery. Then, the FDS will use its own knowledge to identify which discoverers (e.g. Discoverer-1 and Discoverer-2) will probably be leveraged for serving this federated discovery request.

Step 3a: The FDS sends out a solicitation request including a data discovery proposal (e.g. desired data types and potential rewards to earn) to Discoverer-1 to ask it whether it is willing to help in processing the discovery request from the User-1. If the Discoverer-1 agrees to their received data discovery proposal, it will accept the task. Otherwise, it can also send back its suggestion for negotiating with the FDS.

Step 4a: The FDS receives the acknowledgment and feedback from the Discoverer-1.

Steps 3b and 4b are the same as Steps 3a and 4a, respectively.

Step 5: The FDS sends a transaction to PDL Node-1 (or other PDL nodes) in order to create a smart contract, which contains the agreed data discovery proposals of the involved discoverers.

Step 6: PDL Node-1 (or other PDL nodes) confirms the creation of the smart contract.

Steps 7a and 7b: For each involved discoverer (e.g. Discoverer-1 and Dsicoverer-2), the FDS sends a notification to each of them respectively for informing them of the created smart contract as well as the agreed/final discovery task.

Step 8a: For Discoverer-1, it initiates a discovery request and sends this request to Organization-1. For example, the request could indicate which types of data are to be discovered.

Step 9a: Organization-1 first needs to make sure Discoverer-1 has the right privilege to conduct data discovery within Organization-1. If so, it will accept the request and start to conduct discovery processing.

Step 10a: Organization-1 returns the discovery result to Discoverer-1. Depending on the format requirement of the smart contract inputs, Discoverer-1 could further transform or reformat the discovery result.

Step 11a: Discoverer-1 sends the discovery result to the smart contract as a smart contract trigger.

Steps 8b-11b are similar to Steps 8a-11a.

Step 12: After the smart contract receives the discovery results from all the involved discoverers, it will evaluate the quality of the discovery result. Based on that, the smart contract will automatically decide how to allocate the rewards among those different discoverers. In addition, the smart contract will also aggregate all the received discovery results in order to produce the final discovery result for the federated discovery request.

Steps 13a and 13b: The smart contract completes the rewards allocations and sends notifications to the involved discoverers (e.g. Discoverer-1 and Discoverer-2) regarding the reward payments respectively.

Step 14: The smart contract delivers the final aggregated discovery result to the FDS.

Step 15: The FDS returns the aggregated discovery result to User-1, which is the final discovery result for Rqst1.

8 Conclusions

8.1 Introduction

The present document discussed federated data management use cases and their key issues. Then PDL-based federated data management architecture and some key solutions were presented. Several operational guidelines for PDL-based federated data management will be summarized in this clause. Finally, recommendations for next steps are included.

8.2 Operational Guidelines

To leverage PDL for solving federated data management issues, the following operational guidelines could be considered:

- A proxy or an intermediary service function can be designed to facilitate and enable more efficient interactions between PDL systems and federated data management system.
- The proxy or the intermediary service function can be deployed as distributed functions integrated with PDL nodes or a standalone function serving FDM system and PDL system.
- The computing/storage capabilities of PDL nodes can be potentially leveraged for helping various data management processing.

8.3 Recommendations for Next Steps

- Data in federated data management use cases has different characteristics (e.g. type, size, etc.). Data characteristics will probably be taken into account when configuring (e.g. designing/selecting/sizing) ledgers and smart contacts for federated data management. This could be realized via the intelligent automation put on top through smart contracts.
- Specifications on the interactions between FDM and PDL could be developed.
- Specifications on using smart contracts for FDM could be developed.

History

Document history		
V1.1.1	September 2021	Publication