Towards a distributed Infrastructure for a Multimodal Visibility Service

Draft 10-2023

www.federatedplatforms.eu

FEDeRATED
NETWORK OF PLATFORMS

Co-financed by the Connecting Europe Facility of the European Union

Table of Contents

Summary

The objective of FEDeRATED is to realize a data sharing infrastructure meeting the vision and constructed along the building elements of the Digital Transport and Logistics Forum (DTLF) Subgroup 2. One of these elements is called 'Technology Independent Services'. This document provides an example of such a TIS called a Multimodal Visibility Service.

This service is specified according to the design principles and architecture FEDeRATED. This implies that all relevant aspects are specified like event data structures and event logic. Since the Living Labs and prototypes developed by these Living Labs are not yet capable to implement all functionality and there are not yet use cases that require such a service, the following choices are made for demonstration purposes:

- Event sharing – only visibility events and queries for additional data are shared. Thus, event logic and the value added functionality will not be demonstrated.
- eFTI demonstration – one of the main aspects for demonstration will be an eFTI infrastructure, both for B2B and B2G. This will not be complete but demonstrate how eFTI can be implemented.
- Additional use cases – there are at least two use cases that will be demonstrated, namely a rail use case (Deplide-Simple) and an air use case (IATA – Dutch Customs Administration).

The objective is to construct these use cases during a hackaton in October 2023 and demonstrate them at the final event (November 2023). This document will be updated according to the results of these demonstrations and based on input from (at least) the Architectural Team of FEDeRATED. A final version will be made available in the first quarter of 2024.

1 Introduction

1.1 Objective

To develop and validate the specifications for a multimodal visibility service supported by nodes implementing the so-called index functionality of the FEDeRATED architecture as (1) a basis for harmonizing the various Living Labs and (2) creating an infrastructure.

Such an infrastructure enables multiple use cases interconnecting the various solutions and Living Labs of the individual stakeholders in the FEDeRATED Action. Each LL/solution may have its own, localized interface with the infrastructure.

Relevant parts of the architecture will be applied like interaction patterns for visibility, the semantic model, and an initial setup of the Service Registry for each participant in the common LL.

Any applicable assumption for this demonstration will be given in this document.

1.2 Background

All FEDeRATED Living Labs have developed events (with a data pull mechanism) that provides supply chain visibility. All these events differ. They are based on what can be called a call of a transport means like a vessel call, a position of a transport means combined with geo-fencing, arrival or departure of cargo (trailers, containers) at a terminal (gate-in, gate-ot), or are on the level of cargo tracking for transport legs in a chain like load and arrival for individual legs in a chain of Codognotto.

These different approaches to visibility don't provide multimodal visibility, implying that a shipper, consignee, or forwarder coordinating different modalities must implement different mechanisms. This will lead to higher costs, both in processes and IT. The objective is to create a multimodal visibility service and support its implementation by the various Living Labs, thus creating an infrastructure.

The FEDeRATED architecture is the basis for developing a data sharing infrastructure in supply and logistics. This encompasses the semantic model, the Service Registry, the Index functionality, and Identification, Authentication, and Authorisation (IAA). These technical specifications must provide input to the four building blocks specified by DTLF SG2: plug and play, technical independent services, architecture, and safe, secure, and trusted data sharing.

In this case, the specification and the validation will show how plug and play can be deployed for multimodal visibility services where these services are the technical independent services. In this case, plug and play implies that each stakeholder will implement its relevant visibility services and integrate these with the generic multimodal visibility service. Technical independent means that the multimodal visibility service is specified independent of its deployment. It can be deployed by for instance openAPIs (API – Application Programming Interface) and semantic technology (RDF – Resource Description Framework).

1.3 Common pilot – multimodal supply chain visibility

FEDeRATED constitutes various Living Labs, each with its own stakeholders, business -, and use case(s). Some of these LLs already (have the intention to) collaborate. The objective of the common pilot is (1) to harmonize functionality of individual pilots and (2) to create an infrastructure for interoperability between the various LLs and their supporting platforms or solutions.

With some exceptions, the Living Labs develop a visibility solution. It is about Estimated - (ETA) and Actual Times of Arrival (ATA), and positions as a basis for ETA calculation. It is also about loading – and discharge

with their estimated – and actual times. For larger hubs like ports, it is about berthing, piloting, and towing as business services provided by a Port Authority that are in some cases mandatory.

Thus, visibility for different modalties (multimodal supply chain visibility) is taken as a common denominator since many of the existing use cases somehow address this service and it's addressed as one of the services in the FEDeRATED application form part D (technical and financial information).

This document contains the specifications of a multimodal visibility service. The first version of this document will be validated in practice, after which the specifications will be improved and can be implemented by various stakeholders.

To properly validate the service, it will be deployed using a so-called FEDeRATED node. After validation and completion of the specification with input of the validation, it is up to each stakeholder to implement and deploy the service using its own implementation. Use of the FEDeRATED node is not mandatory.

Since each user may have its own, localized interface to a node, i.e. its own APIs, the infrastructure will not work for all users when removing 'nodes' since APIs will not match. Additional effort will be required.

1.4 Technology Independent Services

One of the objectives of FEDeRATED is to provide input to the so-called Technology Independent Services. These services will support data sharing in transactional – and framework contract relations. They consist for instance of booking -, transport -, and visibility services for multimodal business activities in supply and logistics. This specification will show how such a Technology Independent Service is specified, namely a multimodal visibility service for transport. Other Technology Independent Services are a transport booking – and a transport ordering service.

Each Technology Independent Service is represented by an interaction pattern for a business activity. An as such, each Technology Independent Service can be supported by for instance a set of openAPIs, SHACLs used for validating sharing triples, and messages (data push). The various interactions of these Technology Independent Services can also be mapped to existing standards that function as data carriers. Thus, they provide so-called standards implementation guides.

Technology Independent Services can also be developed to support data sharing requirements in a community or to support a regulation. The Service Registry supports the design of these Technology Independent Services by implementing the data sharing ontology (see the note on 'Semantics').

1.5 Layered set of agreements for implementation

The objective is to specify a set of agreements providing seamless interoperability for business collaboration and compliance to regulations, the so-called protocol stack. The upper layers of this protocol stack specify the Linked Event Protocol and the Business Collaboration Protocol (next figure)

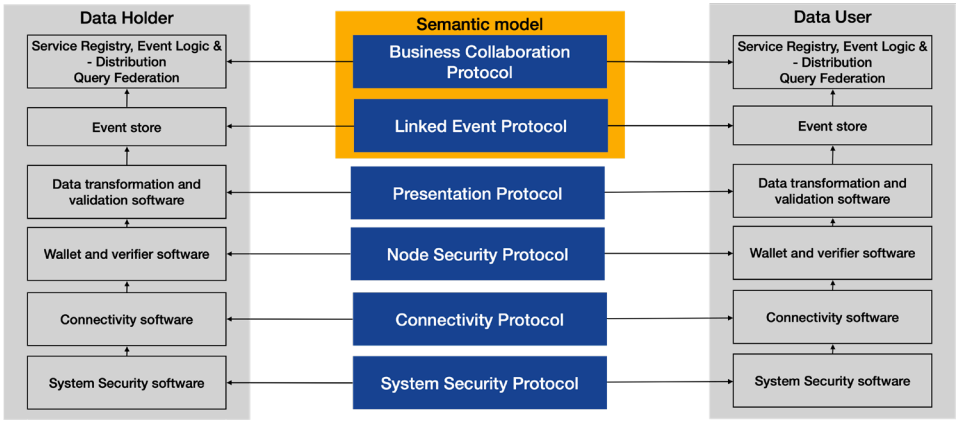


Figure 1: Protocol stack

Whereas in the common Living Lab for the multimodal visibility service some choices are made on the presentation layer and lower levels, the upper two layers can be implemented separately:

- Sharing events – this is implementing the Linked Event Protocol only. Visibility events are shared and can be validated on their content and structure according to SHACL documents only.
- Event logic – the sequencing of events is validated by the event logic specified in this document.

We will distinguish five use cases for event processing by individual stakeholders, namely:

1. **Transaction progress**: informing a customer on the start, relevant changes, and the finalization of a particular transport order.
2. **Authorities**: re-use of relevant transport data for its particular governance role (piggy backing)
3. **Physical operation**: the physical operation generates events, either manual or via sensors. These are input to the previous use cases.
4. **Leg synchronisation**: the synchronization of adjacent legs in a transport chain that do not have a transactional relation but stakeholders of each leg share the same customer.
5. **External conditions**: access to status information generated by some stakeholder in a chain, where this status information fulfils a relevant condition for further action.

The specification is structured according to the various iterations that are supported:

1. First iteration: sharing **events** only. Any event logic is either implemented by an IT system of a user or handled by a human. This implements the Linked Event Protocol. This addresses transaction progress and informing authorities.
2. Second iteration: **event logic** for validating event sequencing and completeness and correctness of performing a transport operation in line with an order. This is about synchronisation of transport legs and checking if all cargo is transported. It implements the business collaboration protocol.
3. Third iteration: **itinerary based event generation**. Actual load and unload activities are performed by humans and/or machines. These can enter or generate the associated events that are a basis for informing customers and/or adjacent legs in a transport chain. It requires sharing state information.
4. Third iteration: **(semi-)automatic event generation**. Implementation of a smart event distribution mechanism where incoming events are used to generate new events. This iteration is not yet specified in this document, but can be of value where an ETA update of a transport means is used to inform customers of the ETA of their cargo carried by that transport means.

The common Living Lab will (only) support the first iteration. It is up to participants (and others) whether

and how they will implement the other iterations.

1.6 Design choices

Although the proposal is independent of a modality and cargo type, its current focus will be mainly on road transport, crossing borders between Member States and EU borders (e.g. transport to the UK). Additional features for other modalities are included and will be realized during the IATA Hackaton in June 2023 (see section 7 of this document).

Of course, this version of the service also needs to be upgraded to reflect real-life situations.

1.6.1 **Design choices at logistics level**

The multimodal visibility service specified in this document is restricted by its functionality to support logistics processes. It can be completed with the following aspects at a later stage:

- Modality specifics – each modality will have its own way of operation, leading to potentially additional interface specifications. This additional specification may have to be supported by additional APIs or extensions to existing APIs for a modality.
- Cargo specifics – each cargo type will have additional requirements as to stakeholders involved and thus data sharing aspects. Additional APIs will have to be generated to support these requirements.
- Dangerous cargo – dangerous cargo will have its own data requirements, potentially also different per modality (e.g. (deep)sea, road, and inland waterways have the same classification, air and rail have different ones).
- Localization – each location (e.g. sea-, air- and inland port) may have its own data requirements that differ. For instance, seaports have port authorities and processes with piloting and tugging. Furthermore, localization can also be on the country level with different authorities governing specific regulation.
- Business activity – the current focus is on 'transport'. A multimodal visibility service for transport is supported. Since synchronization with other business activities like transshipment and storage/production is required, these business activities can be included later.

The previous implies for instance that dangerous cargo, bulk (dry or liquid like chemicals), and reefer transport are out of scope and can be included at a later stage. This will mainly affect data sets that can be retrieved and operations that are required.

These specifics can gradually be developed and made available as configurations. It also requires the support of interaction patterns for transshipment, that may differ from those of transport.

Any extensions may also require the review of event distribution rules, resulting in change and/or new rules.

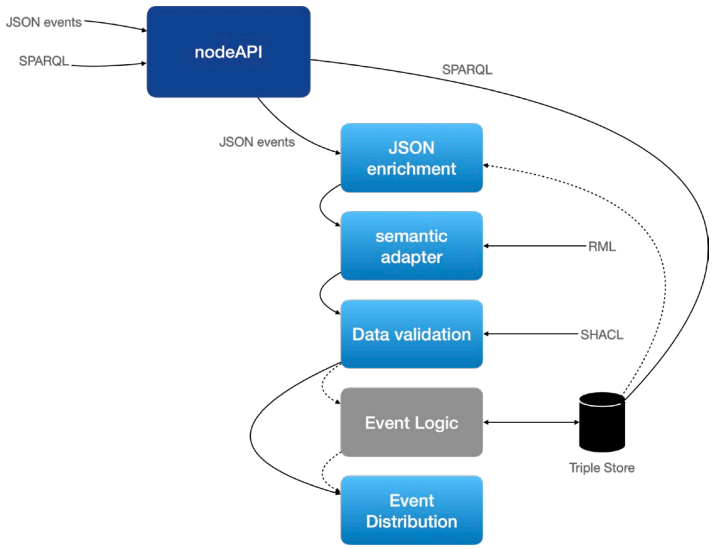
It is also feasible to implement event distribution rules that (semi-)automatically distribute events received from one node to one or more other nodes. This supports truck drivers of a carrier to report their status change (load, discharge) to that carrier, update the state to a customer, and share state data with a CA. A consigner acting as customer of a carrier may (semi-)automatically report the progress of a consignment to its customer(s), the consignee(s).

1.6.2 **Design choices at technical level**

The objective is to provide each organization with options for interfacing with a node. There are three options related to the concept 'profile' (see the results of the IATA Frankfurt Hackaton and a note on profiles):

- eventAPI – the node interfaces via a generic eventAPI with an internal IT system. Data validation by the eventAPI is minimal. Data validation is based on the SHACL files referred to by a profile. It is recommended to locally implement all openAPI calls (PUT/POST, GET).
- baseAPIs – any profile will refer to a design. This design can function as a baseAPI for interfacing with an internal IT System. Data validation according to the design is supported by the baseAPI. As such, data validation is generic and not specific to an organization. A profile provides additional constraints that can be validated with SHACL.
  Since an organization can have multiple roles, i.e. that of customer and service provider, it must implement all openAPI calls (PUT/POST, GET).
- profileAPIs – each profile is implemented by its own openAPIs. These openAPIs will validate the data. It implies that an organization will have as many openAPIs with a node as interactions supported by its profile(s).

The functionality of the nodeAPI is depicted as follows:



The nodeAPI performs data validation as specified by its functionality. This differs per type of API given here. The following functionality is performed 'behind' the nodeAPI:

- JSON enrichment – including UUIDs for concepts. In the current version, detection of duplicate concepts (e.g. containers with their unique user IDs) is not implemented, meaning that a single concept can have multiple UUIDs in the triple store.
- Semantic adapter – transforming JSON data to RDF with an RML (Rule Markup Language) document generated by the prototype tool.
- Data validation – validating the RDF input data with a SHACL file generated by the tool. The data validation functionality is a SHACL validator.
- Event logic – validating the event sequencing based on events that are already stored in the triple store. This is not yet implemented but will be specified in this document for multimodal visibility.
- Event distribution – distribution of an event to the proper destination. Event distribution is based on

rules related on a commercial relation (customer-service provider) and compliance to regulations.

In the current implementation of the node, events are stored in the triple store by the data sharing mechanism provided by Corda.

In case of profileAPIs, all data validation is part of the openAPI code and no additional SHACL validation is required. ProfileAPIs are not yet supported. They require extension of functionality of the prototype tool used to generate these openAPIs. In case of the eventAPI or baseAPI, SHACL validation is required for a particular organization to implement its profile. In case the node does not yet support organization specific APIs, the baseAPI reflects the functionality specified in this document and could be implemented.

The eventAPI reflects the so-called Linked Event Protocol. Such a generic API can be used to share and access events that have been received or send by a node. In addition to this protocol, the initial state of an interaction pattern must be configured by a node for implementing event distribution. This is required since the multimodal visibility service assumes the existence of a transport order common to a customer and service provider.

Since the eventAPI can be used to share all types of events, a consumer of that API must know what data to put in. Normally, this is specified by an openAPI. However in a generic event openAPI, this is not clear and can only be validated by SHACL.

In an organizational network where each participant publishes its (localized) openAPIs, a specific set of openAPIs would lead to a large set of those APIs. However, the openAPIs are only specific to a single user for its interface with the node. Thus, each organization will have their own specific openAPIs and is not aware of those that are required by others.

There are two additional options for interfacing with a node, namely:

- Triple endpoint. A user can upload triples to the node. These triples are validated by SHACL, that are known to a user. In this case, the semantic adapter is not required.
- SPARQL endpoint. A user of a node can of course also select the implementation of a single, generic semantic (SPARQL) endpoint, in which case also the various SHACLs (design and configuration) would be used for data validation. It implies that the function of each event primitive in an interaction pattern is part of the (RDF) data shared across this endpoint.

The current version of a node supports a SPARQL endpoint, but only for accessing and storing data in its local triple store. This endpoint cannot be used to share data between two nodes.

In case of a transactional relation, all applicable Technical Independent Services will be implemented, starting at an initial state for a business activity. In case of a framework contract, details of the framework contract could function as the initial state of a node.

1.7 Structure of this document

This document elaborates the setup and interfaces of the common LL:

- Overview of relevant stakeholders
- The infrastructure - system setup
- Specification of multimodal visibility service and its APIs
- Value added functionality
- Support of organizations
- Development plan

The current version of this document shows the integration of IATA OneRecord with the infrastructure of nodes. Future versions may include integration of others in the infrastructure.

2 Stakeholders and initial setting

This section provides details of the participants and use cases in the common pilot for validating the specifications of the multimodal visibility

2.1 Stakeholders

The following stakeholders are involved in the common LL:

- Italy – Condognotto and TSG/Grimaldi. TSG/Grimaldi have their own SPARQL endpoint.
- Spain – Ministry (Simple)
- Finland – Vediafi, Ahola
- Sweden – RISE (Deplide)
- Netherlands – Ministry of I&W (BDI)
- 51Biz – OneApp for accessing data by authorities
- IATA – integrating OneRecord with a hinterland modality with a gateway developed during the IATA hackaton (June 2023, see this document).

These stakeholders can (jointly) develop their use case(s).

Each stakeholder may choose to participate in the validation. Since not all stakeholders have implemented semantic technology (yet), they will integrate with a FEDeRATED node using openAPIs.

2.2 How to specify a use case

Like said, a use case constitutes at least two participants. The following steps must be taken for specifying a use case:

- **Stakeholders and their roles**. The logistics roles (like consignor, consignee, carrier) and business roles (these are customer and service provider) are assigned to each enterprise. In case of an authority, this will have the role of Competent Authority (CA). It may result in complex patterns involving more than two stakeholders with their logistics – and business roles. These complex situations are visualized as transaction trees reflecting the business and authority hierarchy.
- **Sequence diagrams for business collaboration and compliance.** Visualization of data flows between the business roles and with CAs. Each business role and CA is reflect by a vertical line. These sequence diagrams are independent of any platform.
- **Business data storage.** Identifying where the data is stored that is the basis for data retrieval. This might be a platform like an eFTI platform.
- **Technical sequence diagram**. This sequence diagram is required in case a platform is used for data storage. The sequence diagram will visualize how the data flows of each stakeholder with others and/or the platform.

The basis assumption is that each participant in a use case will either integrate with a platform or solution of the stakeholders in this so-called common Living Lab or have the availability of a node as will be mentioned hereafter.

2.3 Use case(s) and data

The infrastructure will be able to support many use cases, where each stakeholder will be able to select its

use case of value and thus its required APIs. A use case requires at least the participation of two organizations, either two enterprises or an enterprise and an authority. If a use case has a single participant, at least another participant must be simulated for a demonstration.

Data is only shared between the nodes of a participant in a use case and thus not externally visible. All data may be presented as part of a demonstration of the infrastructure. Preferably data that is shared reflects actual cargo flows; if this data is not available in a use case, the stakeholder of that use case must present artificial data as much as possible reflecting cargo flows.

It is recommended that any two pairs of stakeholders participating in the common pilot provide thier use case. The next pages present the use case for IATA with Dutch Customs. There are other documents specifying for instance a Codognotto - and a customs-Singapore use case.

2.4 IATA use case

One of the potential use cases is that of the airlines providing details to Dutch Customs for arrival of flights. Since the airlines implement openAPIs based on OneRecord, these will be transformed into openAPIs fitting the FEDeRATED architecture. The transformation will be developed as validation by IATA and will illustrate how each airline may interact with an EU customs authority in the future.

In this case, there is a data sharing scenario between an airline's ONE Record server and the Netherlands Customs Authority's BDI node. The purpose is to enable faster and more accurate decision-making and targeting for shipment approval and transportation. By implementing certain checks and measures, the customs authority aims to enhance security, prevent illegal activities, and optimize the approval process for shipments.

The proposed data exchange is shown hereafter (it still needs to be mapped to the specifications given in this document, section 4).

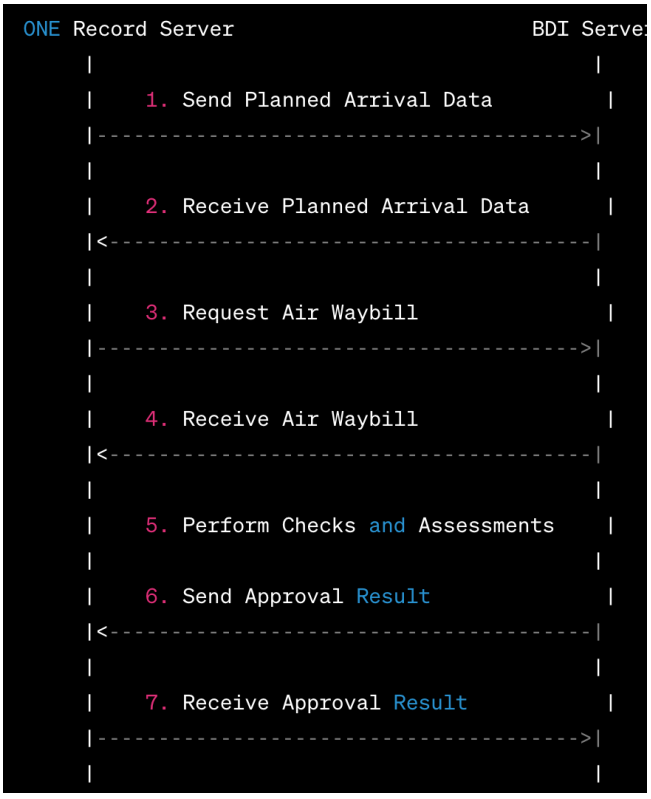


Figure 2: sequence diagram for interaction between IATA OneRecord and a FEDeRATED node

Explanation:

1. The ONE Record server sends the planned arrival data (flight details) to the BDI node.
2. The BDI node receives the planned arrival data from the ONE Record server.
3. The BDI node requests the air waybill(s) from the ONE Record server.
4. The ONE Record server sends the air waybill(s) to the BDI node.
5. The BDI node performs checks and assessments on the received air waybill(s) and other relevant information.
6. The BDI node sends the approval result (e.g., "OK") to the ONE Record server.
7. The ONE Record server receives the approval result from the BDI node.

2.4.1 Data provided by ONE Record :

1. **Planned Arrival Data**: The ONE Record server sends information about the planned arrival of a flight to the BDI node. This includes details such as the scheduled arrival time, flight number, origin, and any other relevant flight information.
2. **Actual Arrival Data**: Once the flight arrives, the ONE Record server also sends the actual arrival data to the BDI node. This includes the actual arrival time, any deviations from the planned schedule, and other pertinent information about the flight's arrival status.
3. **Air Waybill**: Upon request from the BDI node, the ONE Record server provides a copy of the air waybill associated with the shipment arriving on the flight. The air waybill contains crucial information about the shipment, including details about the goods being transported, shipper information, consignee information, and other relevant documentation related to the shipment.

The Planned and Actual Arrival data & time are pushed by the ONE Record server to the BDI node. In ONE Record these would be pubsub notifications. For BDI these are probably an Event POST on the BDI REST API.

2.4.2 Data response from BDI Node

In this scenario, the BDI server sends the following information to the ONE Record server:

1. **Approval Result**: After performing checks and assessments on the received air waybill and other relevant information, the BDI node generates an approval result. This result indicates whether the shipment is approved for further processing or if any issues or discrepancies have been found. The approval result can be communicated as a status message, such as "OK" indicating approval or any other relevant status code or message.
2. **Status Updates**: In addition to the approval result, the BDI node may also send status updates or notifications to the ONE Record server. These updates can include information about the progress of the customs clearance process, any additional requirements or actions needed from the airline, or any relevant updates regarding the shipment's status.

By sending this information back to the ONE Record server, the BDI node ensures that the airline is informed of the approval status and any necessary actions or updates related to the customs clearance process.

These approval results and status updates should normally be events that are hosted by the BDI node. When there is an update to the results and/or status, BDI will POST a message to a /notifications endpoint at ONE Record endpoint.

3 The infrastructure

This section elaborates the setup of the infrastructure, whereby a FEDeRATED node implements the Index functionality required for the multimodal visibility service and provides openAPIs that can be localized. Local APIs based on a common specification will be explained later in this document; these support plug and play.

3.1 Setup of a multimodal visibility infrastructure

The setup of the multimodal visibility infrastructure can support various use cases based on a common infrastructure with a set of openAPIs. There are two types of use cases supported by this infrastructure, namely business-to-business (B2B) visibility, and business-to-administration (B2A) governance (either voluntarily or based on a regulation like eFTI). Each B2B use case consists at least of two enterprises and potentially an authority for B2A. These use cases are preferably with potential users of the infrastructure; if not available these may be simulated.

Each participant in the common pilot will have the same capabilities, namely act as service provider, customer, and authority. Some participants may have limited capabilities, like only supporting an authority.

Each stakeholder must integrate with a so-called (FEDeRATED) node. Basically these nodes run in a single cloud environment (dockerized) for demonstration purposes, meaning that each stakeholder will have its own node. A stakeholder may also decide to implement a node in its own (cloud) environment. TNO provides the nodes for each stakeholder in TNO's cloud environment.

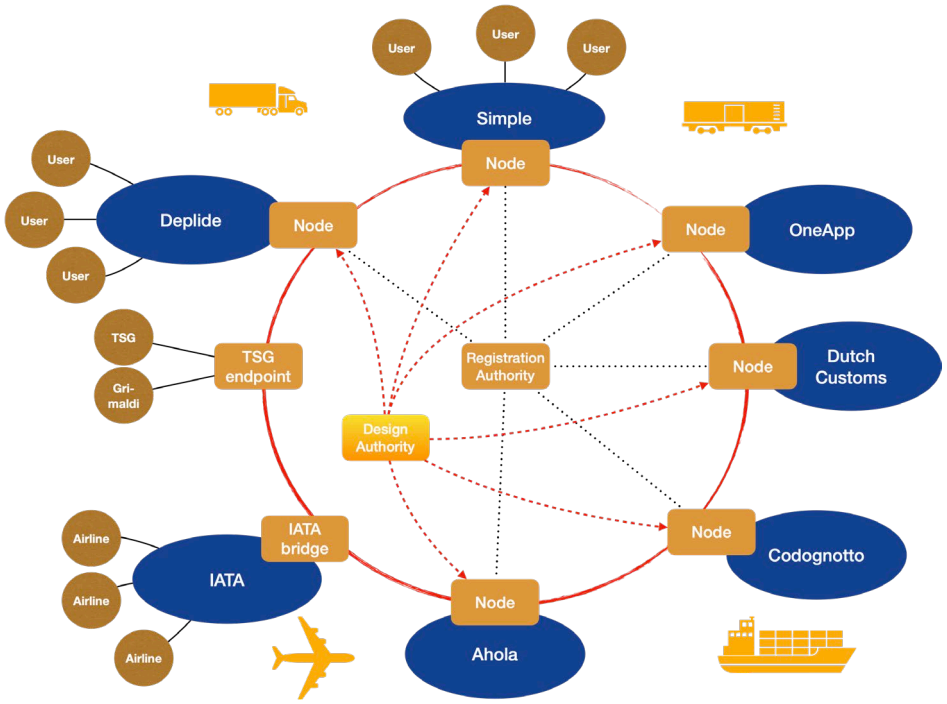


Figure 3: setup of the infrastructure

This infrastructure creates a federated network of platforms[1] implementing the visibility pattern between a customer and service provider, including access by a competent authority. The previous figure shows that TNO will at least install six nodes in its cloud environment. Eventually, the number of nodes can be extended or the network can be reconfigured. The figure also shows two stakeholders that directly interface with a peer node, e.g. TSG endpoint serving Terminal San Giorgio and Grimaldi and the IATA bridge interfacing with OneRecord (see before). These stakeholders will be trusted during the pilot, but are not registered by the proprietary Registration Authority.

The FEDeRATED Architecture and its support team acts as **Design Authority**. It will provide the specifications for each node. As of currently, TSG and IATA may take these specifications also and implement the functionality in their way. The Corda Network Manager acts as **Registration Authority**; it might be replaced with a Registration Authority issuing VCs (Verifiable Credentials), eventually.

Each stakeholder and user of platform or community can have its own set of APIs for its node (see later in this document).

In case a user has a SPARQL endpoint, the event data and queries are shared across this endpoint with the endpoint of a node. That node functions on behalf of that user in the common cloud environment. This is required since the current implementation of a node is based on Corda, which is a (freeware) COTS solution (COTS – Commercial Of The Shelve). Corda and a node have an openAPI interface and additionally Corda provides a so-called network manager and supports non-repudiation (data integrity and audit trail). The Corda network manager, however, is not able to recognize a non-Corda SPARQL endpoint as part of the network.

In case a participant in the common LL does not have a user (enterprise or authority) the participant must simulate one or more users, where these users can have a role of customer and service providers. Other participants act as user (Codognotto, Ahola, and OneApp) and yet others may decide to use existing users of their platform to share data in a demonstration setting of the common LL.

The objective of the common LL is to demonstrate one or more (fictive) use cases. Any implementation choices for operational use by stakeholders may change. For instance, each participant may choose to implement ‘node’ functionality itself.

[1] Other terminology for such an infrastructure is ‘Mobility Data Space’. Furthermore, the infrastructure will supported what has been introduced by Dutch Customs Administration and HMRC as ‘data pipeline’.

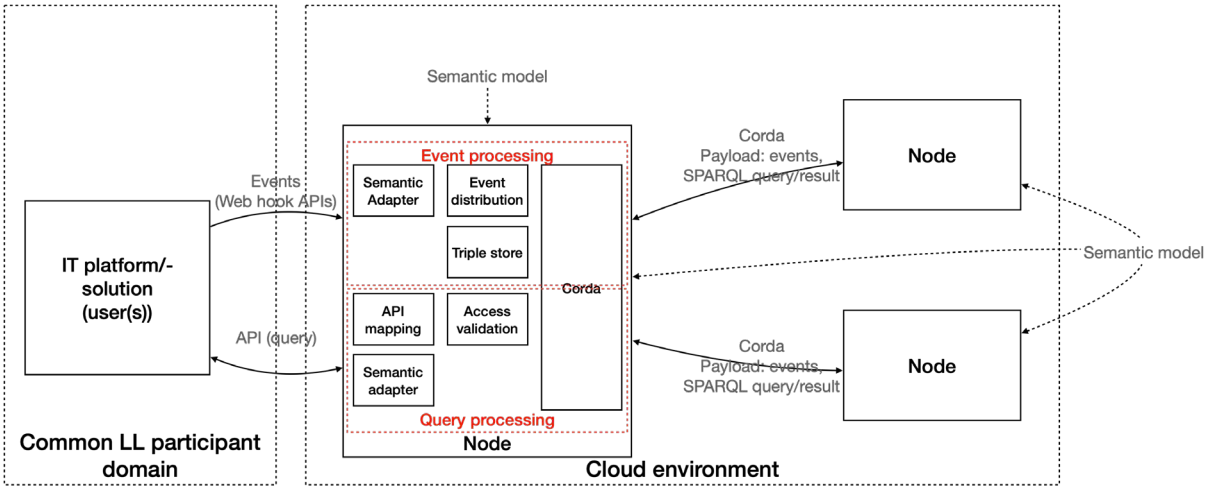


Figure 4: FEDeRATED node functionality

3.2 Node functionality

The node that will be provided as Docker/Kubernetes container via github by the Netherlands has the following functionality (development based on the current version 0.2 yet to be done, see previous figure):

- **Local openAPIs** with a platform or IT solution of a participant. There are two types of interfaces:
  - Webhook API for pushing events to a node.
  - REST/openAPI for data retrieval of a query. The query is based on events with links shared between various stakeholders.

  These openAPIs are provided as separate Docker containers and need to be assembled with the Docker installation of the BDI node. OpenAPIs can be made specific to an organization or platform integrating with that node (see section 1).
- **Node**. Generic functionality that is independent of any use case. It comprises Corda and a graph database (graphDB) implementing the semantic model. The node is available as Docker container with generic openAPIs for sharing and storing data with other nodes.
  - **Interfaces between nodes**. These are based on the current implementation of Corda by the prototype v.02 of the BDI node. Corda provides a registration mechanism (Corda Network Manager) and safe, secure, and reliable data sharing via AMQP and TLS.
  - **Data sharing between nodes**. All data is shared a triples (RDF) and SPARQL between nodes over Corda.
- **Event processing**. The capability of a node to receive (JSON) event data, transform it to RDF (semantic adapter), share it with the proper other node(s) (event distribution), and store what has been shared (triple store. This functionality is part of the Docker container with the openAPI code.
- **Query processing**. The capability of a node to validate that another node has also received a link and is allowed to receive a response to a query. The response will be retrieved via a single REST API from a user's system (API mapping). The semantic adapter will forward the response in RDF to the requesting node/user.

Two components need further configuration to support the visibility pattern, namely the semantic adapter (events, query response) and event distribution.

3.3 ONE Record – BDI hackathon architecture

During hackathon IATA will deploy the proposed architecture as below:

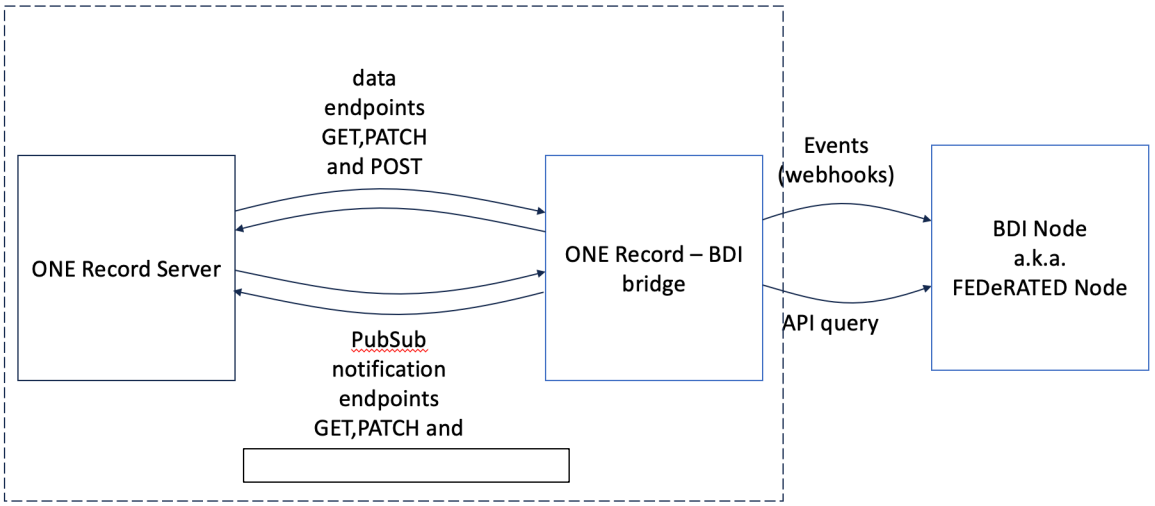


Figure 5: Integrating IATA OneRecord servers with a FEDeRATED node

There are currently no rules for subscription to flights and their ETA. These are considered as open data, enabling everyone to subscribe to the flights at airports (departure, arrival) they require. An authority will not make a subscription, this must be configured by an airline for each flight with cargo.

4 Multimodal visibility service specification

This section provides the specification of the multimodal visibility service. The specification is technology independent; tools will be applied to make technology specific deployment, for instance dockerization and generation of openAPI code from ontologies.

The specification is organized according to the layered set of agreements. First, the visibility pattern is given identifying its functional states and event primitives. Secondly, the event structure is specified supporting the Linked Event Protocol and validation of structure and content of events with SHACL. It identifies an event distribution mechanism where in a commercial setting the intended recipient is included by a sender and geographical coverage is used for compliance. This section also defines the openAPIs to support the sharing of events. Thirdly, the event logic is specified, based on the service specification. It includes event distribution based on the existence of an order between two commercial parties. This order information is entered as initial state. Event distribution for compliance is not affected by event logic.

There are still design issues with respect to the events and supporting APIs as given in the introduction.

Since the output of the specification is taken as input for localization (next section), all specifications of states, events, and queries/results will be generated as SHACL documents using existing tools. These SHACL documents are constraints to the FEDeRATED semantic model. The interaction pattern of states, events, and state transitions is an instance of the FEDeRATED data sharing ontology (see the note on the semantic model). These instances and SHACL documents are available for localization.

4.1 The multimodal visibility service.

This section presents the interaction pattern of the multimodal visibility service for transport. It is used to identify the various events that can be shared and their sequencing, where the latter is the input for event logic.

This interaction pattern is per order between a customer and a service provider. Any events shared for individual orders can be triggered by an operation at the level of a transport means. This is especially the case for 'ETA –' and 'position events' where the ETA and position of all cargo carried by a transport means is updated. This is not (yet) part of this specification.

The visibility interaction pattern (next figure) consists of activities by which events can shared between a customer and service provider, where these events can also be shared with an authority. For instance, a service provider submits a load event to its customer, followed by an ETA event. The following events are supported: load event, ETA event, Incident event, and Unload event. Their allowed sequencing is given in the following diagram, where circles represent a state (states: agreed order, in execution, completed, to be cancelled), rectangles represent data sharing processes (processes; start, ETA update, Position update, Incident/accident, complete), and envelopes with an arrow the initiation of an event by one of the roles (the blank role like 'LSP' for 'start' process) and the other the recipient (the grey role like 'customer' for the start process).

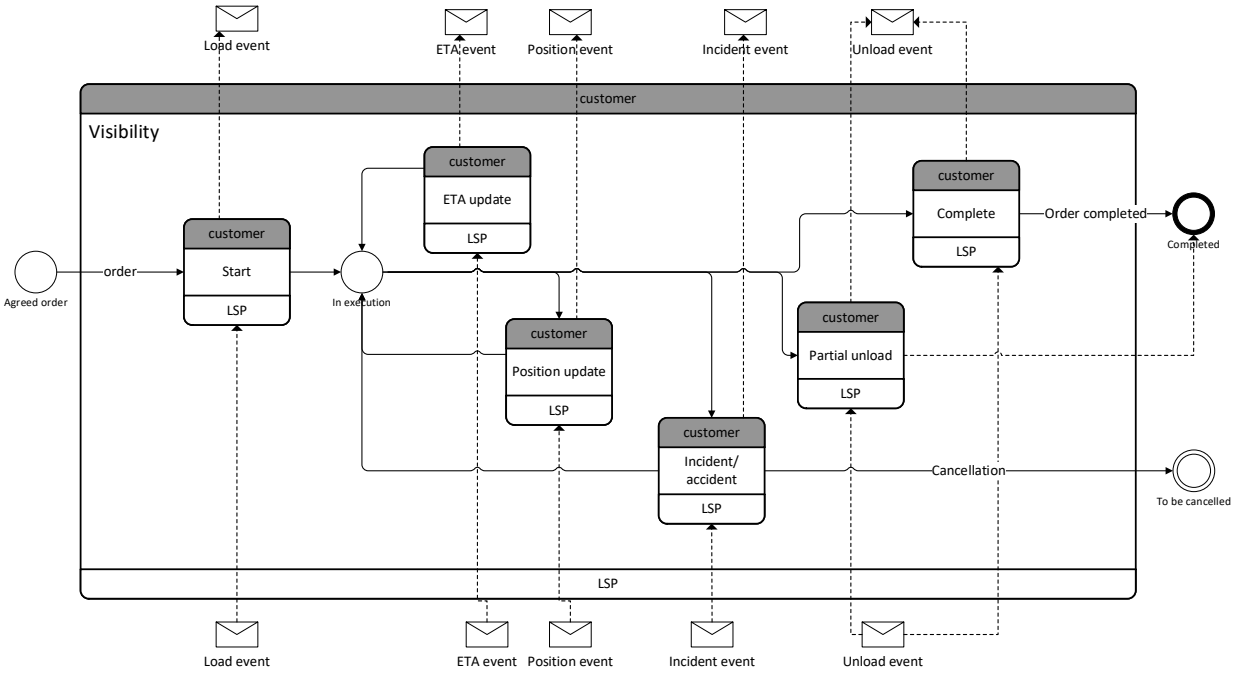


Figure 6: interaction pattern of the multimodal visibility service (specified as BPMn 2.0 choreography)

The most basic example of interactions between a customer and LSP are by sharing a load event, followed by an ETA event, and completed with an unload event.

One of the processes of the pattern shown by the previous figure requires decomposition, namely the process 'incident/accident'. The start and estimated end of an accident or incident can be provided by an event, whereas the end time can also be given by a separate event (see the description of the events). This is not yet done and will have impact on the event logic. However, a start or end of an accident or incident may not always be shared since a human may not be capable to do this. A loss or damage can be reported by a single incident event.

The service specification shows that cargo can be unloaded in steps. This is by introducing the state 'partially unloaded'. The incoming load event is processed by the appropriate transition when of the pre-condition of that transition is met. Thus, whenever an unload event is received, the pre-condition of both events is validated to decide on its result. Of course, partial unload can be related to a partial load, for instance transport of cargo with multiple trucks or trailers from between two locations. Whereas a partial unload is visualized in the choreography, the partial unload is not shown but can easily be included (leading to an extra state transition).

The states represented in the interaction pattern relate to 'transport' as the movement of cargo between two locations. These locations have different names for different modalities:

- **(Deep)sea**. The locations are Port of Loading (POL) and Port of Discharge (POD). This visibility pattern refers to loading, departure, arrival, and discharge of cargo in these ports, where the events refer to the port area. Each port will have more detailed events referring to business services of third parties (on behalf of a port authority) taking place in those port areas, e.g. tugging and piloting.
- **Air**. The locations are the airport of departure and the transit or destination airport of a flight. A flight is comparable with a voyage of a vessel, a trip of a truck or a path of a train. A flight has a slot at an airport; flights are managed and coordinated by air traffic control.
- **Road**. The locations are the Place of Acceptance (PLA, the place where the cargo is taken over by

a carrier) and the Place of Delivery (PLD, the place where the cargo is handed over by the carrier).

- **Rail**. These are the stations where the cargo is loaded onto or in a railway wagon and the station where it is handed over. A railway wagon is part of a train that has a path on the (inter)national railway infrastructure. National paths are assigned by a national Infrastructure Manager; EU paths are assigned via Railnet Europe in coordination with national Infrastructure Managers.

The next pages specify the events with an event distribution mechanism, potential queries, and event logic.

All data sets will be expressed as SHACL constraints to the semantic model and configure the semantic adapter.

4.2 Linked event protocol

This section specifies the events that implement the Linked Event Protocol (see section 1.5). It is only about validation of the event structure and content. There is no validation related to state information. All events that are shared (received or submitted) by a node are stored in its index (i.e. the triple store of that node).

4.2.1 Event structure

Conceptually, each event of the multimodal visibility service has the following structure:

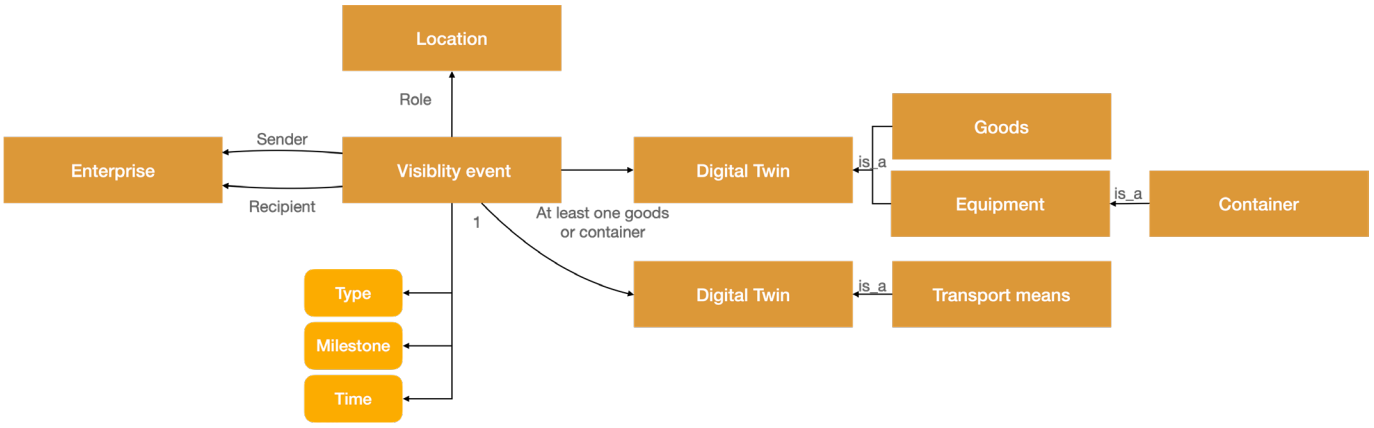


Figure 1 conceptual structure of visibility events

The figure shows that an event represents an association of Digital Twins (at least one 'Goods' or container and a transport means) at a location with a role. The role can be Place of Acceptance (PLA), Port of Loading (POL), or any other relevant to the visibility service. An event is of a type, where the type refers to its function in the choreography. Types are for instance 'loading', 'unloading', and 'position'. A type has a specific value of 'milestone' for creating (start) and ending (end) an association. Visibility events will always have the time 'estimated' (ETA event) or 'actual' ((un)loading, position) and are send by an enterprise in its role of 'service provider'.

This results in the following structure for visibility events that will be expressed by the ontology:

Visibility events			Load	ETA	Incident	Position	Unload
	UUID (event)		✅	✅	✅	✅	✅
	UUID (sender organisation)		✅	✅	✅	✅	✅
	UUID (recipient organisation)		✅	✅	✅	✅	✅
	milestone		start	start/end	start	start	end
	estimated date/time			✅	✅		✅
	actual date/time		✅		✅	✅	
	External reference		✅				
	Reference type		Document data set				
	Location						
		Location role: place of acceptance, place of delivery	PLA	PLD	Position	position	PLD
		UUID (location)	✅	✅	✅	✅	✅
	Cargo (at least goods or equipment as a rule; can be multiple)		✅	✅	✅		✅
		For goods - UUID (goods)	✅ (o)	✅ (o)	✅ (o)		✅ (o)
		for equipment - UUID (equipment; can be multiple))	✅ (o)	✅ (o)	✅ (o)		✅ (o)
	UUID (transport means)		✅			✅	
Digital Twin - transport means - (truck/vessel/airplane/barge/train)							
	UUID		✅				
	Transport means ID		✅				
	Transport means ID provider		✅				
	Transport means Nationality		✅				
	Transport mode		✅				
Digital Twin - goods							
	UUID		✅		✅		✅
	Number of packages		✅		✅ (o)		✅ (o)
	Type of packages		✅				
	remark		✅		✅		
Digital Twin - equipment - (container/trailer/wagon)							
	UUID		✅		✅		✅
	Equipment ID		✅				
	Remark		✅		✅		

An incident (or accident) is represented by three events, namely the actual start, the estimated end, and the actual end. These events can be shared separately; they are applied to calculate the delay caused by any of these events. They can be shared with at least two events: the first (milestone = start) indicates the time at which an accident or incident occurs with the estimated end and the second (milestone = end) the actual completion. Note that accidents or incidents cannot always be generated, since a human may not be capable to signal such an event. Any delays caused by a traffic jam have impact on an ETA and can be reported as such.

As the previous table shows, an incident event may give loss or damage to cargo, i.e. goods or equipment. This should be indicated when detected. There can be different types of incidents like damage, loss or theft of cargo. Accident are also processed as incidents; they might only result in delays.

The previous table shows that five types of transport means can be given, one per transport mode. Thus, the transport modality indicates the type of transport means. The identification of a transport means is assigned by an authority, that may be a national authority in for instance the case of license plates of trucks. In case of an airplane, it is recommended to use the flight number as identification; a flight number indicates the movement via air from one airport to another using an airplane. This is not correct but will do for the moment.

The table shows that three subtypes of equipment can be used, namely containers, trailers, and (railway) wagons. Any additional subtypes could be included. Goods are identified via the types of packages: all packages of the same type are grouped. Either equipment or goods are given as cargo. In case of equipment, the different subtypes of equipment can be provided.

In case the event does not have a reference to cargo (goods or equipment), the visibility event is applicable to a transport means. It may for instance give the ETA of a transport means for arrival at a location (note that

A load event may have a reference to a document data set (optional), which can be the customer order

reference. The document data set differs per modality. Road for instance uses a CMR data set, air the Air Way Bill (AWB) and sea the Bill of Lading (B/L). Such a reference may also be considered a reference to a customer order:

- For a customer -service provider business relation, the transport order can be applied.
- An authority also does not require this reference, since it can search on other criteria like 'transport means ID' (license plate of a truck, vessel code, etc.) and equipment identification (container number, license plate of a trailer, wagon number, etc.).

A CA will at least receive load and discharge events.

4.2.2 Event distribution for sharing events only

As the visibility pattern shows, all events are shared by an LSP (Logistics Service Provider, referred to as 'service provider' hereafter) to a customer, whereas some events are also shared with a competent authority (see Interface specifications). Both enterprises and competent authorities may access data based on links they have received.

At the level of the Linked Event Protocol, the recipient in its role of customer must be provided by the sender in its role of service provider. The Index of a node (i.e. the triple store of a node) only stores all events shared with other nodes (received and submitted).

Events shared between two enterprises can be duplicated for compliance reasons and shared with one (or more) Competent Authority/-ies (CA). By sharing events with CAs, those CAs can access data via the links that are shared. To select the data they require, they may want to use a reference number that they have retrieved otherwise, for instance a license plate of a truck retrieved via Automated Number Plate Recognition (ANR).

The event distribution mechanism is as follows:

- **B2B** – the sender/recipient combination must be given by the sender of an event. This is used to map to an identification for actual data exchange. At the level of event logic this mechanism is different (see later).
- **B2A** – a CA will receive all visibility events of cargo that passes and is loaded and/or unloaded in its competency domain. The following rules are implemented by event distribution for the visibility events, they can also be applied to the agreed order giving already an indication to an authority of planned logistics movements to its domain:
    - If extract(event(visibility)_PLA (Place of Acceptance), country_code) equals CA_country_code → CA will receive a load event.
    - If extract(event(visibility)_PLD (Place of Delivery), country_code) equals CA_country_code → CA will receive a load (also if PLA is not in its territory) and a unload event.
    - If extract(event(visibility)_position, country_code) equals CA_country_code (border crossing positions), → CA will receive the position event. The CA needs to extrat whether this is an exit or entry of its domain based on the route (outside scope).

Assumptions:

1. **Competent Authorities** – these will always receive load/unload events as specified by the event distribution for demonstration purposes, independent of any regulation.
2. **CA territory** – for demonstration purposes, the territory is a country. The country code is part of the UNLOCODE of PLA/PLD.

The B2A mechanism can be refined at a later stage, for instance by preventing that a CA can only access data of the load of a transport means when it is in its territory and not elsewhere. This includes details of an itinerary.

4.2.3 Query formulation

The UUIDs of events and their referenced transport means, equipment, and goods are the basis for retrieving more information, both by enterprises and Competent Authorities (CAs). Each enterprise or CA can formulate its own queries or re-use standardized queries where the output of these queries may be different for an enterprise and a CA. Examples of those queries are:

- **Retrieve general information** – based on the UUID of a load event, the order or document data set is retrieved.
- **Retrieve detailed information** – based on the UUID of a visibility event that is retrieved via for instance the UUID of a transport means or equipment, an enterprise or CA may want to receive details of the cargo of, including its agreed order (consignment data). For an enterprise this might be for instance the weight of a container; for a CA this could be a container track.
- **Retrieve specific information of a Digital Twin** – the UUID event of load/discharge must contain the UUIDs of the relevant Digital Twins, since the user defined identifier (e.g. license plate of a truck or trailer, container number) are used to by a data user like a CA for remote monitoring. The specific information may contain details of the goods or content of a trailer/container, which is specified by the query formulated by that CA, where this goods details may only be available to a consignor and not a carrier.

In this first version, the two basic queries are:

- **Business document data set query for enterprises**. The query is on the UUID of the event (state data) and retrieves all data representing a business document.
- **eFTI data – and AWB query for CAs**. This is a query for authorities with a subset of the eFTI data set as specified within the eFTI Regulation and the AWB data specified by OneRecord.

The following table lists an example of these queries. This example is not complete and does not reflect the actual situation like the eFTI, AWB, or any other data set. It serves as a basis for demonstration.

Functional data requirements and mapping to the ontology			authorities (g2b)					business (b2b)				
Functional	data properties	Semantic model ('concept.property')	road	air	inland waterways	rail	sea	road	air	inland waterways	rail	sea
	User identification		eFTI identification	AWB number		CIM number	B/L number	eCMR number	AWB number		CIM number	B/L number
Consignor		Consignor	x	x	x (customer)	x (customer)	x	x	x	x (customer)	x (customer)	x
	Consignor ID	organization.ID	o	o			o	o	o			o
	Consignor Name	Organization.name	x	x	x	x	x	x	x	x	x	x
	Consignor Address	Location.postal address	x	x	x	x	x	x	x	x	x	x
	Postal code	Location.postal code										
	Street name	Location.street name										
	City name	location.city name	x	x	x	x	x	x	x	x	x	x
	Country code	location.country code	x	x	x	x	x	x	x	x	x	x
Consignee		consignee	x	x			x	x	x			x
	Consignee ID	organization.ID	o	o			o	o	o			o
	Consignee Name	Organization.name	x	x			x	x	x			x
	Consignee Address	Location.postal address	x	x			x	x	x			x
	Postal code	Location.postal code										
	Street name	Location.street name										
	City name	location.city name	x	x			x	x	x			x
	Country code	location.country code	x	x			x	x	x			x
Carrier		Carrier	x	x	x (barge operator)	x (RU)	x	x	x	x (barge operator)	x (RU)	x
	Carrier ID	organization.ID	o	o	x	x	o	o	o	x	x	o
	Carrier Name	Organization.name	x	x	x	x	x	x	x	x	x	x
	Carrier Address	Location.postal address	x	x	x	x	x	x	x	x	x	x
	Postal code	Location.postal code										
	Street name	Location.street name										
	City name	location.city name	x	x		x	x	x	x		x	x
	Country code	location.country code	x	x		x	x	x	x		x	x
Forwarder		Forwarder					x					x
	Forwarder ID	organization.ID					o					o
	Forwarder Name	Organization.name					x					x
	Forwarder Address	Location.postal address					x					x
	Postal code	Location.postal code										
	Street name	Location.street name										
	City name	location.city name					x					x
	Country code	location.country code					x					x
Locations (pla/pld, pol/pod, etc.)												
Goods identification		Digital Twin-goods-identification	xor (goods or equipment)	xor (pieces)				xor (goods or equipment)	xor (pieces)			
	Type of goods	Dgital Twin - goods - Type of goods	x	x				x	x			
	Gross mass (kg)	Digital Twin - goods - gross maas	x	x				x	x			
	Net mass (kg)	Digital Twin - goods - net mass										
	Gross volume (m3)	Digital Twin - goods - gross volume	x	x				x	x			
	Number of packages Total Packages	Digital Twin - goods - number of units	x	x				x	x			
		Digital Twin - goods - nature of the cargo										
	Goods description (textual)	Digital Twin - goods - goods description	x	x				x	x			
Transport Equipment		Digital Twin - Equipment - Container	xor (see goods)		x	xor (container or trailer)	xor (container or trailer)	xor (see goods)		x	xor (container or	xor (containe r or
	Transport Equipment ID	Digital Twin - Equipment - Container - ID	x		x	x	x	x		x	x	x
	Transport Equipment Type	Digital Twin - Equipment - Container - type	x		x	x	x	x		x	x	x
	Transport Equipment Size	Digital Twin - Equipment - Container - size						x		x	x	x
	Transport Equipment Packed Status											
	Seal Quantity											
	Sealed Indicator		x		x		x	x		x		x
	Seals		o		o		o	o		o		o
	Seal number		x		x		x	x		x		x
	Incident code											
	Sealing Party Role Code		x		x		x	x		x		x
Transport means		Digital Twin - Transport means	x	x (flight)	x (barge)	x (train)	x (vessel or ferry)	x	x (flight)	x (barge)	x (train)	x (vessel or ferry)
	Transport means / mode type		x (road)	x (air)	x (inland waterways)	x (train number)	x (sea)	x (road)	x (air)	x (inland waterways)	x (train number)	x (sea)
	Conveyance reference number		o	x (flight number)			o	o	x (flight number)			o
	Transport means ID		x (license plate truck)	o	x (barge ID)	o	x (vessel ID)	x (license plate truck)	o	x (barge ID)	o	x (vessel ID)
	Transport means Nationality		x	o	x	x	x	x	o	x	x	x
Transport Equipment		Digital Twin - Equipment - trailer	o			xor (euipment or trailer)	xor (container or trailer)	o			xor (euipme nt or trailer)	xor (containe r or trailer)
	Transport Equipment ID	Digital Twin - Equipment - trailer-ID	x			x	x	x			x	x
	Transport Equipment Type	Digital Twin - Equipment - trailer - type	x			x	x	x			x	x
	Transport Equipment Size	Digital Twin - equipment - trailer - size						x			x	x
Transport Equipment		Digital Twin - Equipment - ULD		xor					xor			
	Transport Equipment ID	Digital Twin - Equipment - ULD - ID		x					x			
	Transport Equipment Type	Digital Twin - Equipment - ULD - type							x			
	Transport Equipment Size	Digital Twin - Equipment - ULD - size							x			
Transport Equipment		Digital Twin - Equipment - wagon				x					x	
	Transport Equipment ID	Digital Twin - Equipment - wagon - ID				x					x	
	Transport Equipment Type	Digital Twin - Equipment - wagon - type									x	
	Transport Equipment Size	Digital Twin - Equipment - wagon - size									x	

4.3 Event logic

The event logic is part of the state transitions, where each state transition is triggered by an event primitive. The 'agreed order' state data must be initially shared between a customer and service provider to trigger sharing events. This 'agreed order' is the basis of a document data set (like the eCMR) that can be produced at state 'in execution'.

4.3.1 **Data structure**

The data structure of the states and interactions (events) is specified by a prototype tool. Therefore, the data structure is flexible. However, for event logic, a data structure must be known, which imposes rules for specifying states and events.

The following data structure is required for event logic:

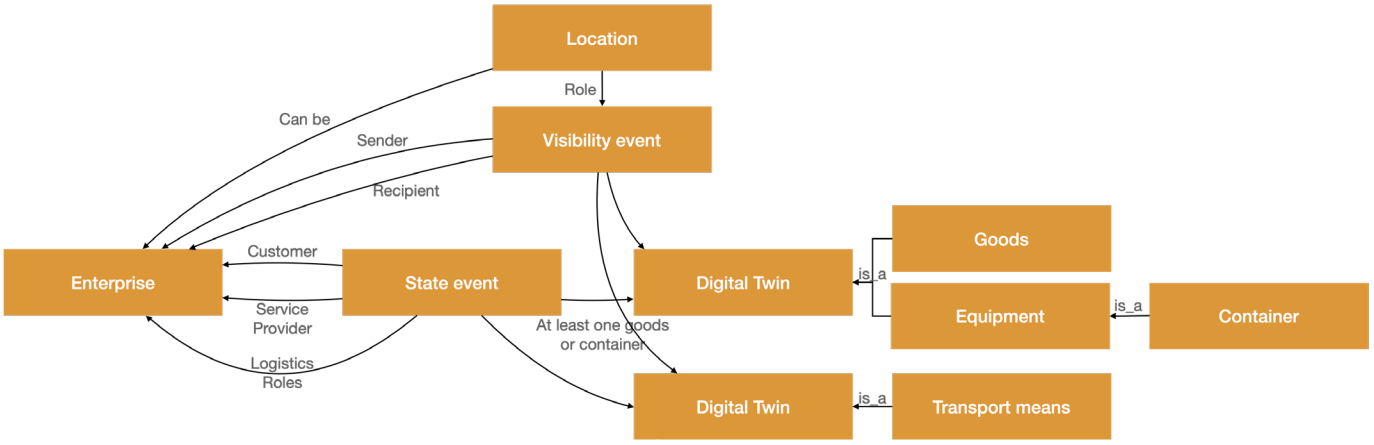


Figure 2 conceptual structure of states

The figure shows two types of events, namely a state – and visibility event. The visibility events have been specified. The state event is similar to the visibility event, but contains at least two visibility events for a 'transport' activity, namely the load and unload event. This is an extra constraint to the visibility events.

Furthermore, a visibility event represents the associations between Digital Twins (in place and time) related to a state event. This relation is given by the fact that Digital Twins of visibility events completely overlap or are a subset of those given by a state event and the sender/recipient associations of visibility events equals the customer/service provider associations of state event as specified by the choreography.

4.3.2 **Initial state of the interaction pattern (agreed order)**

The 'agreed order' state contains the following data set (functional expressed by the semantic model):

This state contains data reflecting a customer order (this table must be updated for multimodal transport; recipient must also still be included):

- General event data (event (state data) reflecting header data. This refers to cargo (goods or equipment), organizations involved (consignor, carrier, consignee), and a transport modality and/or means.
- Associations are via UUIDs (Universal Unique Identifiers)
- Each concept (organization, location, etc.) has a user interpretable identifier like an equipment id.
- Equipment is generic, in the sense that reflects a trailer, container, or any other type of equipment.
- Actual details of the movement of goods or equipment are given by two visibility events, one with the Place of Acceptance (PLA) and the other with the Place of Delivery (PLD).

- If required, any intermediate location can be included, for instance that of border crossing for cargo going into or moving out of the EU.
- Any queries on individual concepts (like Digital Twin – goods) will only result in those data properties given for these concepts.

Main structure of states	Agreed order	
	Load	Unload
Event (state data)		
UUID (event)	x	
UUID (customer)	x	
UUID (service provider)	x	
event type (value - assigned for the abstract message tree)	x	
milestone		
expected date/time		
estimated date/time		
actual date/time		
External reference (this is the user identification)	x	
Reference type (related to the abstract message tree)	Order number	
Organizations		
Organization Role: consignor, consignee, carrier	x	
UUID Organization	x	
Cargo (at least goods or equipment as a rule; can be multiple	x	
For goods - UUID (goods)	o (details)	
For goods - UUID (goods)	x (totals)	
for equipment - UUID (equipment)	o	
UUID (transport means)	x	
Person - legal person		
UUID	x	
External ID	x	
External ID provider	x	
Name	x	
Infrastructure - Location		
UUID	x	
Location code	x	
Location code type	x	
Address	x	
Postal code	x	
Street name	x	
City name	x	
Country code	x	
Digital Twin - goods		
UUID	x	
External ID	x	
Type of external ID	x	
Type of packages	x	
Number of packages	x	
Gross mass (kg)	x	
Net mass (kg)	x	
Gross volume (m3)	x	
remark		
Digital Twin - equipment		
UUID	x	
Equipment ID	x	
Equipment Type	x	
Equipment Size	x	
Transport Equipment Packed Status	x	
Remark		
Digital Twin - transport means		
UUID	x	
Transport means ID		
Transport means ID provider		
Transport means Nationality		
Transport mode	x	
Event (organization location)		
UUID (event)	x	
UUID (sender organisation)		
event subtype		
milestone	start	
expeted date/time		
estimated date/time		
actual date/time	x	
External reference		
Reference type		
UUID Organization	x	
UUID (location)	x	
Event (visibility data)s		
UUID (event)	x	x
UUID (sender organisation)	x	x
event type (value - assigned for the abstract message tree)	x	x
milestone	start	end
expected date/time	x	x
estimated date/time	x	x
actual date/time		
Locations (at least two)		
Location role: place of acceptance, place of delivery	PLA	PLD
UUID (location)	x	x
Cargo (at least goods or equipment as a rule; can be multiple	x	x
For goods - UUID (goods)	o (details)	o (details)
For goods - UUID (goods)	x (totals)	x (totals)
for equipment - UUID (equipment)	o	o
UUID (transport means)	x	x

4.3.3 **State transitions**

All event primitives that are shared as part of the multimodal visibility service update visibility details of all cargo known by state data. This allows for instance to share an ETA event for all cargo, but also to share a loss of single piece of cargo (an instance of Digital Twin that is cargo).

To enable this functionality, event primitive data is stored at the level of individual cargo (instance of all Digital Twins in an order), thus enabling the overall structure of an order with its references to logistics stakeholders. The proposed approach can be amended to handle multiple transport operations for a single order.

All event primitives must be validated before they are processed as part of a state transition. This is a separate function that is not given by event logic.

The way by which the state transitions are specified and event primitives can be shared, transport can be completed for the remaining cargo after part of it might be lost or damaged. This is by including these details at the level of a particular piece of cargo in the state 'in execution'.

The state transitions are specified as follows (see the interaction pattern of the service)

State transition	**Start**	
Input state	Agreed order	Initial upload of the state must be performed for this transition to properly execute. A load event contains all cargo (all UUIDs referring to goods or equipment) that have been loaded. Missing cargo must be detected.
Event primitive	Load event	
Pre-condition	Event (sender, recipient) exists agreed_order (service provider, customer). (For all cargo in agreed_order) exists load_event (UUID) Load_event (PLA) equals (agreed_order (cargo (event (PLA))) Load_event (actual_date/time) in period (agreed_order (cargo (event (milstone = start; estimated)))))	First check always: is a visibility event related to a state event for enterprise roles, second check on Digital Twins. Additional checks that the cargo is loaded at the place indicated by the order and the time is within the estimated period.
Error	Error with a code identifying one of the parts of the pre-condition that is not met: - Order unknown - Difference in place of acceptance - Too late loaded (too early is probably not relevant, since cargo should not be available before a planned loading time)	First error must be shared with the sender of the event to prevent any unrequired data sharing. The event is not shared. The other two errors are indicated to the sender of the event. The event is shared with the recipient, that receives the same error at reception of the event. The sender of the event may recalculate the planned date and provide it as estimate time for completion by a separate event (potentially at a later stage).
Firing rule	If too late or too early then recalculate	-

State transition	Start	
	planned date	
Post-condition (output state)	For all cargo (event (visibility details): store load_event Update event (state data) with load_event (transport means) Optional: update event (state data) with load_event (equipment_trailer)	The transport means given in the load event is updated and included in the order. If the cargo is loaded on a trailer (railway wagon can be included lateron), the trailer is added. This transition can be expanded to cover partial loading of all cargo in a business transaction (e.g. a shipment or consignment); partial unloading is already supported.

State transition	ETA update	
Input state	In execution	There can be two different transitions, where an initial update of an ETA for unload is updated at a later stage. An ETA event is applicable for all cargo given by the state. It can be given at the level of a transport means.
Event primitive	ETA event	
Pre-condition	Event (sender, recipient) exists agreed_order (service provider, customer). (ETA event (UUID)) equals (event (state data) (UUID) and state 'in execution') ETA event (estimated time; milestone = end) within period (event (state data) – for all cargo (UUID) – event (visibility details; estimated; milestone = end))	A validation that an agreed order is in execution and the ETA is still within the time window.
Error	'order in the execution state unknown' 'too late or early unload'	The first error is submitted to the sender of the event to prevent any unrequired data sharing. The second error is shared with the sender of the event after which the ETA event is shared with the recipient. At reception, the error is shared with the recipient.
Firing rule	-	-
Post-condition (output state)	For each cargo (event (visibility details)) with milestone = end: update (estimated time)	In execution

State transition	**Position update**	
Input state	In execution	It is assumed that a position is only given once with its actual state, where the result is 'milestone = start' indicating the position is passed. This could be updated by given an estimated time at which a position will be passed. A position is applicable to all cargo that has been loaded and given at the level of a transport means.
Event primitive	Position event	
Pre-condition	Event (sender, recipient) exists agreed_order (service provider, customer). (position event (UUID)) equals (event (state data) (UUID) and state 'in execution') (position event (location) not in (all cargo for event (state data) with event (visibility details) (position))	There must be an order in a state of execution and the position must not yet be given
Error	'order in the execution state unknown' 'position already shared'	This error is submitted to the sender of the event to prevent any unrequired data sharing. The sender of the event receives an error when the position is already shared and the event is not shared with a recipient.
Firing rule	-	-
Post-condition (output state)	For each cargo in (event (state data)) include event (visibility details) (milestone = start: location = position event (location)	In execution

State transition	**Incident/accident**	
Input state	In execution	Three types of incidents or accidents can be reported: - Loss of cargo - Damage of cargo - Delay caused by an accident The type of incident or accident is given by a code of the event.
Event primitive	Incident or accident event (in brief 'incident event' is given hereafter)	
Pre-condition	Event (sender, recipient) exists agreed_order (service provider, customer).	First the order should exist and secondly for loss or damage the cargo must be present.

State transition	Incident/accident	
	(incident event (UUID) exists in (event (state data) (UUID; state = in execution)) Incident event (loss or damage) and (incident event (cargo UUID)) exists in event (state data) cargo (UUID)	
Error	'order in execution unknown' 'cargo given for loss or damage unknown or not loaded'	These errors are shared with the sending actor to prevent any update with the recipient. The second error indicates a state error. First action is to synchronize states between two participating stakeholders.
Firing rule	If Incident event (accident) then calculate (ETA unload; delay) and indicate error in case ETA unload exceeds the period given by the state	This is at the moment the single transition with a firing rule where the impact of the delay is calculated. This calculation may be simple by for instance adding the delay to the estimated time of unloading (milestone -= end) or considering any other details that may occur after the accident.
Post-condition (output state)	Case - Accident: for each cargo in (event (state data)) include accident event as event (visibility details) (milestone = start: location = accident event (location); accident event (actual time) Update the estimated time for milestone = end of all cargo specified by the event (state data) - Damage or loss: for all incident event (cargo UUID) include the incident event to the applicable cargo via the UUID.	In execution

State transition	Partial unload	
Input state	In execution	
Event primitive	Unload event (partial)	
Pre-condition	Event (sender, recipient) exists agreed_order (service provider, customer). ((unload event (UUID equipment)) exist in (event (state data) and (all UUID equipment; state = in execution) or (((unload event (UUID goods) exists in (event (state data) (UUID goods; state = in execution)) and ((unload event (UUID goods – number of	Partial unloading is indicated via the event. The unload must also fit in the time window of the estimated/planned data, since it is the basis for synchronization with adjacent legs. The location must also equal the expected.

State transition	Partial unload	
	packages)) equal or less than (event (state data) (each UUID goods – number of packages)) and (for each cargo in (event (state data) (location for visibility event with milestone = end, expected date given) equals (unload event (location)	
Error	'not all cargo unloaded' 'not all packages unloaded' 'unloading at a different location than expected' 'unloading too early or too late as planned'	Following type of errors: - Not all equipment or goods are unloaded - Not all packages of goods are unloaded (missing packages that are not reported) - Cargo is unloaded at a different location than expected by a customer - Unloading too early or too late. In case of a partial unload, only the completion will specify the final time and its position with respect to the time window (too late or too early). These errors may not necessary give rise to an action by a sender, like the unloading location might change and has to be reported. In case of unloading of partial shipments this is reported via the transition 'partial unload'
Firing rule	-	-
Post-condition (output state)	For each cargo in event (visibility details) include event (visibility details) (milestone = end: location = unload event (location) in event (visibility details)	Only those cargo will be in state 'completed' that are reported. All others remain in the state 'in execution'

State transition	Complete	
Input state	In execution	
Event primitive	Unload event (final)	
Pre-condition	Event (sender, recipient) exists agreed_order (service provider, customer). ((event (state data) (all UUID equipment; state = in execution) exist in (unload event (UUID equipment)) or (((event (state data) (UUID goods; state = in execution) exists in (unload event (UUID goods)) and ((event (state data) (each UUID goods – number of packages) equals (unload event (UUID goods – number of packages)) and (for each	All cargo that was loaded is reported by unloaded by a single unload event, unless a loss has been shared during transport with an incident event. The unload must also fit in the time window of the estimated/planned data, since it is the basis for synchronization with adjacent legs

State transition	Complete	
	cargo in (event (state data) (location for visibility event with milestone = end, expected date given) equals (unload event (location)	
Error	'not all cargo unloaded' 'not all packages unloaded' 'unloading at a different location than expected'	Three types of errors: - Not all equipment or goods are unloaded - Not all packages of goods are unloaded (missing packages that are not reported) - Cargo is unloaded at a different location than expected by a customer These errors may not necessary give rise to an action by a sender, like the unloading location might change and has to be reported. In case of unloading of partial shipments this is reported via the transition 'partial unload'
Firing rule	-	-
Post-condition (output state)	For each cargo in (event (state data)) include event (visibility details) (milestone = end: location = unload event (location)	Completed, where any damage to particular cargo is stored as mentioned by the event.

Any pre-condition can be implemented by SPARQL queries where event data is used to query state data. A positive result of the query makes the pre-condition 'true'; a negative must generate an 'error', where depending on the error action is taken (see the specifications).

The pre-condition is an update to the state data with the event data. Existing events (visibility) at the level of cargo may have to be updated or new events (visibility) must be inserted. For instance, it will be good to keep track of any ETA updates by including them to each digital twin representing cargo.

In case any 'ETA –', 'position –', or 'accident/incident event' is shared in the context of an order after the unload event has been processed and the state is 'completed', these events are discarded. In case this is at the sender (i.e. the service provider), they are not shared with the recipient (i.e. the customer). In case this is at the recipient (i.e. the customer), they are considered to be out of sequence and can be discarded. Any data contained by those events could have caused a delay or damage or loss of cargo, which is already detected when processing the unload event.

Any unload event must be preceded by a load event. If a sender still tries to share an unload event without having shared a load event, a warning must be given and the unload event will not be shared. If a recipient receives an unload event without having received a load event, this is an error (the unload event can only be processed when the state is 'in execution', see the transitions). In case of such an error, synchronization of state between sender and recipient might be necessary.

4.3.4 Event distribution associated with event logic

Uploading and synchronizing the initial state between two nodes that want to share events, i.e. that have a commercial relation based on an order, serves as a means to distribute events. The sender node needs

to relate an event to its state, detect the relation in the shared business transaction and use its identification as recipient of an event.

This event distribution must be executed at sending an event before the event logic is executed since the event as such will be shared with the intended recipient. This recipient in its turn executes the event logic and updates its state.

Instead of sharing events, another solution could be to share updated state information from a sender to a recipient, when a sender wants to share an event. It means that event logic is only executed by the sending node and not the recipient one. This solutions of sharing updated state information would work for multimodal visibility, since only one of the nodes will send events to another (from service provider to customer). However, in future situations, a customer may also share events with its service provider and update its state data at a similar time it is updated by the service provider. This may lead to out-of-sync states. Event logic must cater for that when it occurs.

Thus, before any actual visibility events are shared, at least the node of a service provider must contain the agreed order (see initial state specification). The agreed order with its state data and visibility events containing the expected and estimated date/times must be stored by the index. In this case, the service provider may also share this event with its customer, being the consignor by retrieving the customer_UUID from the event and matching it with the Corda identifier of the customer node.

It can also be an option that a customer enters the agreed order with visibility events as mentioned, in which case the event is distributed to the Corda node of the service provider by matching the service_provider_UUID from the event with its Corda node identifier.

Similar, all other concepts like locations, organizations, and Digital Twins have UUIDs, meaning that relevant data is only stored once in an index. If for instance multiple orders are transported by a single truck or trailer (LTL – less than truck load), the UUID of the truck or trailer is stored only once. In this case, each order and its visibility events is only shared by a service provider with a single customer of that order. The operator of a transport means may provide the event to a node, which distributes it to the relevant CAs and customer.

After this initial state information is shared, the rules for event distribution are: a customer participating in an order will receive all relevant visibility events for that order from its service provider. These are the events that are formulated by the visibility pattern. It implies that the existence of an order must be present in the index (graphDB).

- The data structure of an agreed order and load/discharge events are given hereafter. The following rules must be implemented:
    - Consignor UUID is the customer; carrier UUID is the service provider.
    - Consignee UUID is the receiving party that may have to be informed.
    - Each event has a UUID. The UUID of a sender of any event shown in the event table is the UUID of the carrier.
- The rule is as follows:
    - If event_UUID equals agreed order_event(visibility)_UUID as stored in the index and event_sender_UUID equals agreed order_event(state data)_carrier_UUID, then share event with agreed orders_event(state data)_consigner_UUID.
    - An extra check may be performed by comparing the UUIDs of a transport means, equipment, and goods in as stored by the agreed order in the index.

4.4 Generic openAPIs

The set of openAPIs given here is based on a generic approach supporting the Linked event protocol and the concepts of the data sharing ontology. This is still for further research on its feasibility.

4.4.1 **Linked event APIs**

The following APIs will be specified as openAPIs and supported by a node (naming of the APIs will be generated and thus differ from that given here; the specification of these APIs is for further discussion):

- **Put Event** – update of the agreed order state with a visibility event. There is stil a design issue where either each type of event is support by an API or a generic API is applicable where its meaning is based on the data carried by the event. There is event logic involved in processing these events, as will be specified hereafter.
- **Get Event** – retrieval of one or more events. This is a local function of a node, whereby a user of a node is able to retrieve one or more (visibility) events. Different options will be supported:
    - Retrieval of the last event – retrieval of the last event that has been shared. This can be restricted to an order.
    - Retrieval of order events – retrieval of all visibility events that have been shared in the context of an order.
    - Retrieval of events (with a time window) – retrieval of all events in a given period, e.g. a particular day, the last hour, etc.
    - Retrieval of events shared with a particular other user – retrieval of all (visibility) events that have been shared with a particular user.
- **Get Data** (user_known_identifier) – a generic API where a user known identifier is the parameter to search for additional data as stored by an index of a node. This can be for instance a consignment identifier, license plate of a truck or that of a trailer, resulting in a query on the UUID of the event(state data). There are two variants for this query, namely that of a business document data set for an enterprise and of an eFTI data set for a CA. This may result in two separate APIs.
  A user identifier must be known to a node and or shared with the node that requires access to the data. The following rules are applicable:
    - **Data user node**. Based on the user identifier, the associated UUID with all related UUIDs and their user defined identifiers are returned. For instance, a query can be formulated on an consignment or eCMR identifier, returning all associated data of that eCMR in the node. Next, the individual UUIDs can be queried to access data of a data holder, i.e. the one that shared the events. Additional data can only be retrieved for state events that are not yet in the completed state. Furthermore, the user identifier of the local query must be stored in this node. The result of the query per user identifier is:
        - **User identifier of a Digital Twin** – the UUID of any order of event (state data) in which the UUID of that Digital Twin is present and that is not yet completed is retrieved. If the user identifier is not known or there is not an order that is open, a message is returned to the data user. In case of the UUID of a truck or trailer, one (FTL) or more (LTL) UUIDs of orders might be retrieved.
        - **User identifier of event (state data)** – the user identifier must be present either at the level of an event(state data) that is not yet completed or any other object (Digital Twin, location, organization) for orders that are not yet completed. The result is a query to the data holder node with the user identifier known by the IT backend system, where the data holder is identified as sender of event(state data).

- **Data holder node**. The assumption is that this node provides (access to) additional data stored in a backend system (federated querying will not be supported yet). A query for data retrieval by a data user can only be answered if the UUID in that query exists in the data holder node and is shared with the data user that poses the query. If the latter validation rule is true, the specified openAPI to retrieve data from an API backend is initiated with the user identifier. OpenAPIs can be generated at different levels:
  - **State data** – additional order data is retrieved like a transport order that is the basis for an eCMR or another type of transport document;
  - **Digital Twin data** – additional data at the level of individual digital twins is retrieved, for instance a trailer (with its cargo) or a container.
  - **Organization – or location data** – providing additional data of parties involved in a business transaction.

4.4.2 Additional APIs for event logic

The following APIs are included to support synchronization of state information in case of any malfunctioning, loss of events, or processing of events:

- **put State** – the upload of the (initial) state to a node that will be shared with another node in a use case. The event(state data) must have a UUID that is not known by a node receiving the event(state data).
  This API can also be used to update the state as perceived by one of the stakeholders. If this is done, this may lead to an error since the state perceived by a node might differ from the one perceived by a user of a node. Thus, it is best to first retrieve the state of a transaction of a node and analyze differences before a forced state change is made.
- **Get State** – an API operating on ones' node only for retrieval of a state shared with another node, including the events that are shared. A node will have a SPARQL endpoint implementing the functionality of this API, thus providing full flexibility of querying. However, each query result must be processable by a human (via a GUI) or an IT system which requires predefined queries. The following local queries are formulated (there might be more local queries than specified here like the query on a trailer number for retrieval of its trip, as a means to implement governance of cabotage legislation):
  - Retrieval of all agreed orders that have not yet reached the final state (completed).
  - Retrieval of the state of orders that is not yet completed and shared with a particular peer node. This may retrieve all running orders of a service provider with a particular customer or of a customer with a service provider.
  - Retrieval of the state of a particular order. A user identifier for that order must be known and stored in the node.

4.5 Additional conditions - release

One issue that is not yet described is that next steps in the process may only be performed when other conditions are true. These are for instance in transshipment from a deepsea vessel to another transport means for incoming cargo. For instance, the following conditions must be met:

- **Commercial release** – transport and handling charges of the previous transport leg and trans-shipment have been paid. A bank can produce such a release; other relevant stakeholders require that such a release token is published by an authorised bank.

- **Customs release** – especially for incoming cargo, customs has the ability of inspection and has to issue a release relevant for a terminal operator and carrier. Like a commercial release, relevant stakeholders need to know that such a release token is provided by the appropriate customs authority.

Further research is required as to the support of these tokens by Verifiable Credentials. Such tokens would identify the holder like customs.

These conditions will be part of the state transitions to validate compliance. Of course, they need to be validated at physical hand-over of cargo.

    -

5 Value added functionality.

The introduction to this document has identified four iterations, namely that of simply sharing events, introduction of event logic, and generation of events from two perspectives. The previous section has introduced event sharing and – logic as two iterations, each with a proposal for their APIs. This section will introduce the other two iterations, namely:

- Generation of events from an itinerary perspective
- Automatic distribution of events to customers and for leg synchronization in a chain

Like itineraries, other physical actions can be specified. An example is the stuffing and stripping of containers by a stuffing center. These will also result in events.

5.1 Itineraries

This is about a physical action generating events to relevant stakeholders. These physical actions are represented by (physical) state transitions that can be modelled as follows for 'transport':

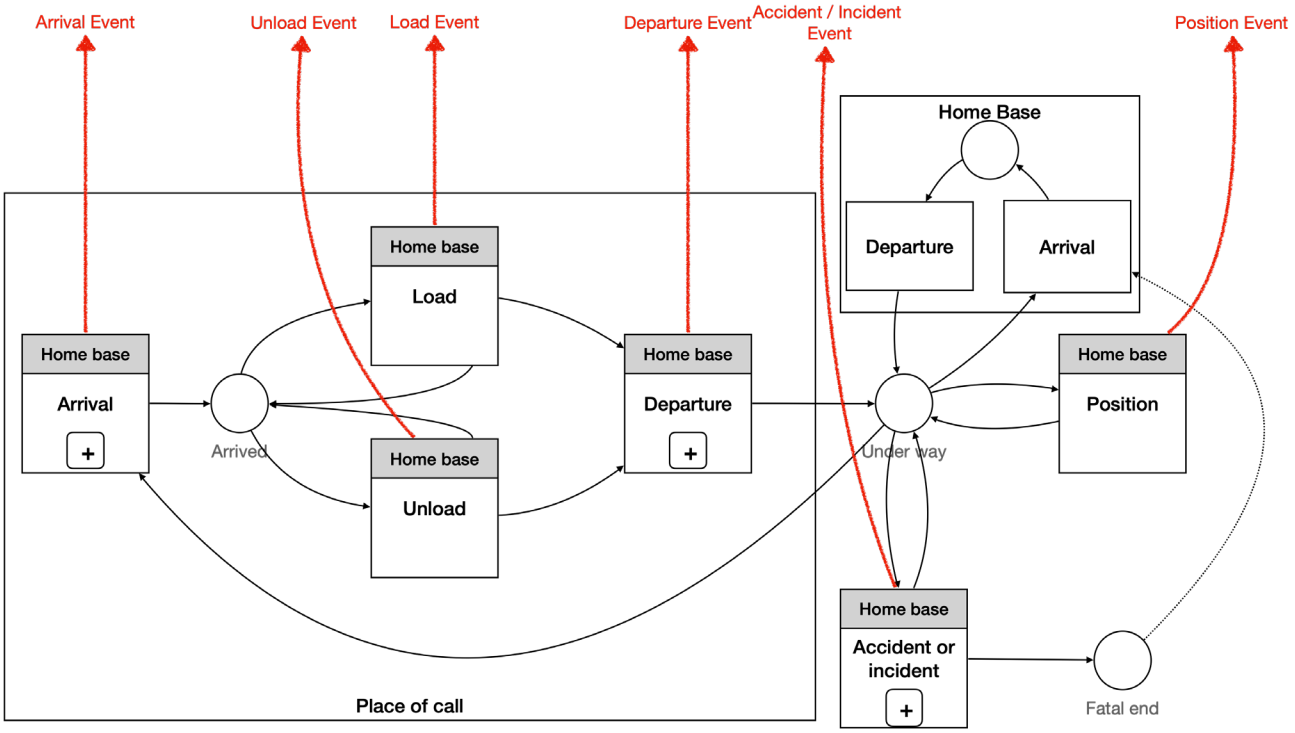


Figure 3 a transport itinerary

It shows that various transitions can take place based on actions that are known to the operator of a transport means, for instance to load a container at a certain location. These are part of the pre-condition. In case of a so-called milk run, a transport means will have an itinerary but no instructions for loading. These may depend on certain conditions like a token raised by a place of call. This is not considered at this moment.

The visibility events are identical to the ones that are specified in section 4.3. Additional to those, there are an arrival and departure event: an arrival event constructs the association between a transport means (and its cargo) and a location and a departure event ends this association. This is shown as follows:

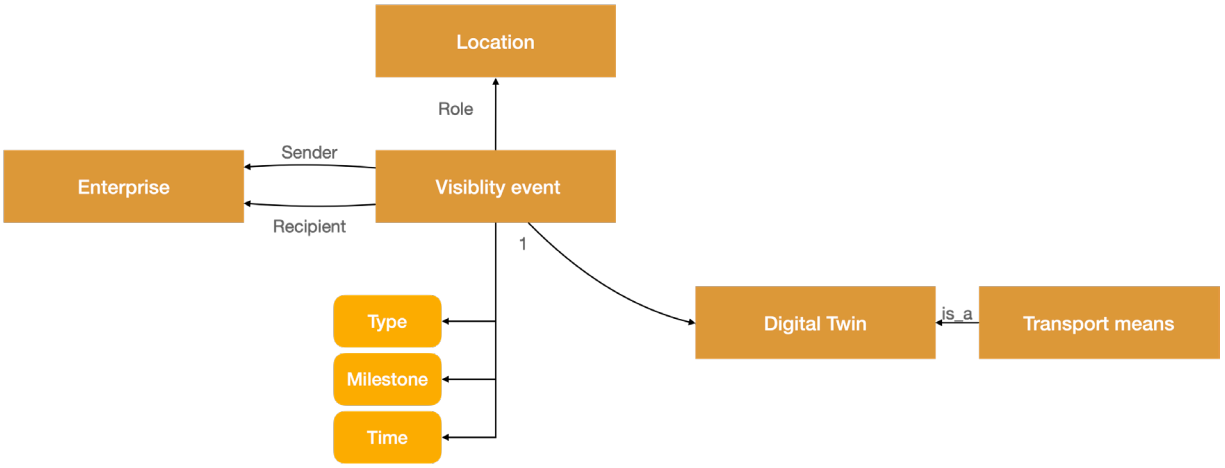


Figure 4 arrival and departure event structure

Arrival and departure events can have synonyms like ‘gate in’ and ‘gate out’ respectively to reflect arrival and departure at a terminal or a warehouse with multiple gates.

The assumption is that a transport means will have an itinerary with load/unload instructions. These might be amended during its itinerary. Load/unload actions can only be performed by a place of call, that functions for instance as a transshipment location. In such a case, the events resulting from a physical action are generated by the operator of that location. This is for instance the case for load/unload of vessels and barges. The events are generated to a customer via the IT system of the transshipment location.

In other cases, like for road transport, the operator of a transport means will have on-board software supporting the generation of events to its home base.

These types of updates of load/unload and changes in itinerary data are not visualized, but is supported by on-board software of a transport means and/or IT solutions of a transshipment operator.

The states are specified as follows:

- Arrived - a transport means (and its load or empty) has arrived at a location.
- Under way – a transport means (and its load or empty) is underway utilizing an infrastructure.
- Fatal end – a transport means cannot complete its itinerary and is either moved to its home base or must be repaired/destroyed.

An arrival (and departure) transition can be decomposed depending on the area of a place of call. For instance, a port area can be considered a place of call, whereas in that port area several sub-locations must be distinguished, for instance for piloting, tugging, and load/unload actions. The same is applicable to accidents or incidents that require additional actions by for instance emergency operators.

The figure of transport itineraries consists of transitions that generate events. These transitions are specified as (only state transitions generating events are shown):

(transitions will be completed)

State transition	**Arrival**	
Input state	-	
Pre-condition		

State transition	**Arrival**	
Error	-	
Firing rule	Generate arrival_event	
Post-condition (output state)		

State transition	**load**	
Input state	-	
Pre-condition		
Error	-	
Firing rule	Generate load_event	
Post-condition (output state)		

State transition	**unload**	
Input state	-	
Pre-condition		
Error	-	
Firing rule	Generate unload_event	
Post-condition (output state)		

State transition	**departure**	
Input state	-	
Pre-condition		
Error	-	
Firing rule	Generate departure_event	
Post-condition (output state)		

State transition	position	
Input state	-	
Pre-condition		
Error	-	
Firing rule	Generate position_event	
Post-condition (output state)		

State transition	Accident/incident	
Input state	-	
Pre-condition		
Error	-	
Firing rule	Generate accident/incident_event	
Post-condition (output state)		

5.2 Leg synchronization in a logistics chain

5.2.1 Cases for leg synchronisation

Whereas in the previous section, the multimodal visibility service specifies event sequencing and supporting APIs including event logic in bilateral collaborations, this part is about utilizing received events for generating new events. The following figure shows an example of a transaction tree of roles and responsibilities for transport of a container via a port by sea to another country.

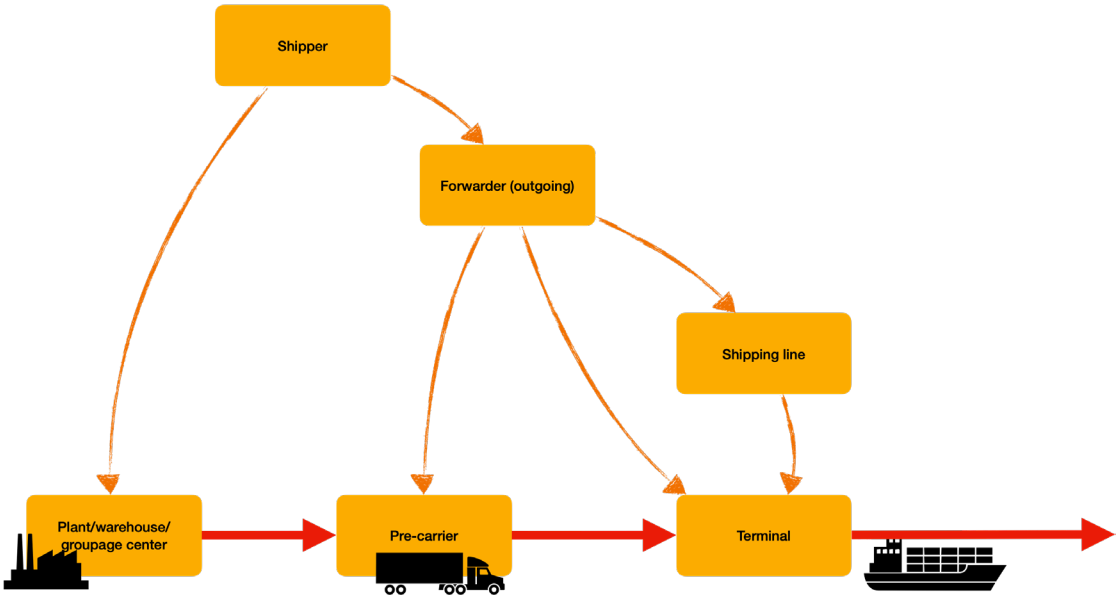


Figure 5 example of a chain

Event sequencing in this chain may take the following:

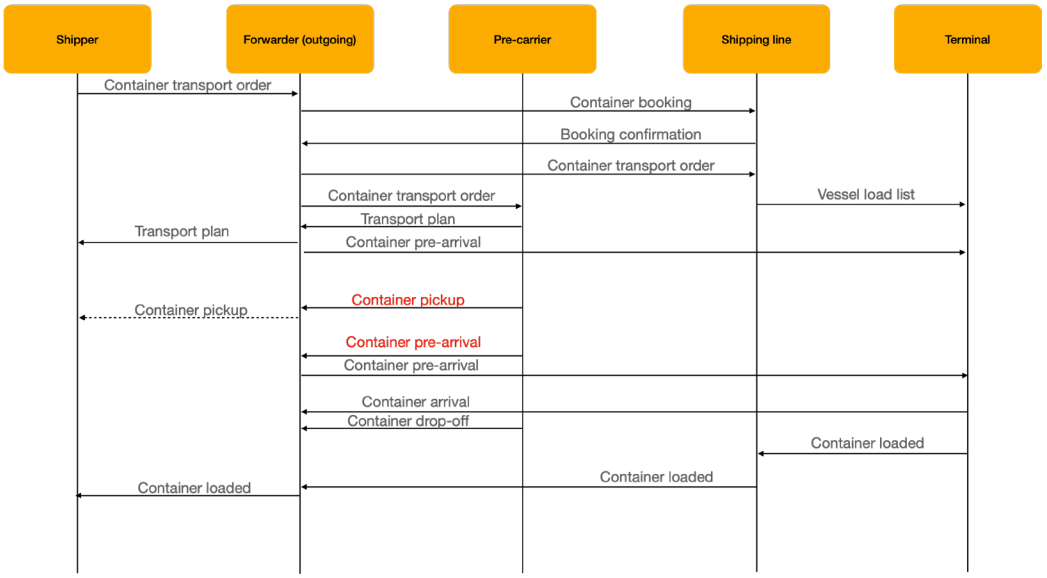


Figure 6 event sequencing in the example

There are two examples, the events 'container pickup' and 'container pre-arrival', that can trigger informing a customer and next leg (semi-)automatically. The pre-arrival will inform the terminal of the estimated time of arrival of the pre-carrier; the pickup will inform a customer that the container is on its way.

This example shows two situations, namely updates of expectations received from a customer and updates to next legs and a customer based on a progress event received from a service provider. Additionally, there is the case whereby a next leg is a type of activity with a fixed schedule like flight or train departure. This requires the previous leg to be finished on time. These three situations are visualized as follows.

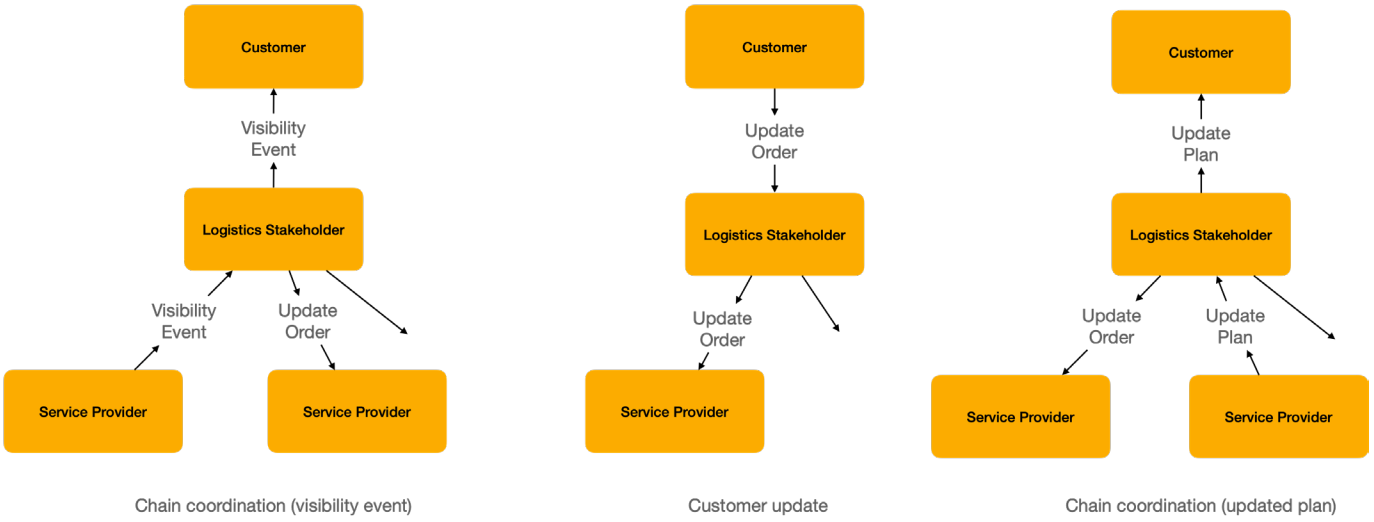


Figure 7 three potential situations for automatically informing chain participants

The previous figure shows updates of expectations by a customer that can be based on progress made by previous legs in a chain. These can be updates on the expected time at which a next leg could start, based on an ETA of a previous one.

The simplest case of the chain coordination is where a carrier informs a customer on delivery of goods,

whereas this customer informs its customer. This reflects for instance an eCommerce delivery activity or the example given before of the container pickup and pre-arrival.

Only the first situation will be specified, since it deals with visibility events where the other two situations are relevant to ordering.

Chain coordination is triggered after reception of a visibility event of a service provider, where the ETA for and/or the completion of the activity is given (or the ETA is calculated). The relevant state transitions of event logic are extended by producing an internal event as part of their firing rule. The following state transitions are relevant:

- ETA update – this can affect the execution of the next leg (or provide an indication of completion to a customer)
- Incident/accident – similar as with ETA update, with the addition that a next leg may have to be cancelled and the activity cannot be completed.
- Completed – this will provide an indication to a customer, whereas the next leg will already have received the relevant cargo.

The firing rule of these transitions is updated with 'generate (internal_event), resulting in the following transition

State transition	**Internal transition**	
Input state	-	-
Event primitive	Internal event (UUID_receivedVisibility_event)	The UUID of the original visibility event is the trigger. The assumption is that it has been processed successfully by the recipient.
Pre-condition	ReceivedVisibility_event (Digital Twin (UUIDs)) exist in State_event (receivedVisibility_event (recipient = (customer and serviceProvider not equal receivedVisibility_event (sender))) or (recipient = (serviceProvider and customer not equal receivedVisibility_event (sender))) (state_event (serviceProvider; visibility_event (unload, location) = receivedVisibility_event (location)) or (state_event (customer; visibility_event(load, location) = receivedVisibility_event (location))	First check: the Digital Twins of the received visibility event participate in another state event. Second check: validate if the received visibility event represents the final leg of a chain or there is an adjacent leg.
Error	-	If the pre-condition is false, no action is required.

Firing rule	If (ETA event) and finalLeg then generate (ETA event) to customer
	
	If (ATA event) and finalLeg then generate (unload event) to customr
	
	If (ETA - or ATA event) and (ETA or ATA is not in adjacentLeg (plannedPeriod)) then generate (orderEvent(expected time = (ETA
		The generated output by the firing rule depends on the type of received visibility event. If it is an ETA or ATA of the final leg, the customer must be informed.
		
		If it is an ETA or ATA relevant to an adjacent leg (i.e. the ETA or ATA is not within the agreed period), that service provider receives an

State transition	Internal transition	
	or ATA)) to adjacentLeg (serviceProvider) If receivedVisibility_event = incidentEvent (damage or loss) and finalLeg then duplicate incidentEvent (recipient = customer) If receivedVisibility_event = incidentEvent (damage or loss) for all adjacentLegs cancel(adjacentLeg)	update. The previous rule might be upgraded in case the there is the delay is too long and the adjacent legs must be cancelled. In case of an incident event with (total) damage or loss, at least the customer must be informed. If there are adjacent legs, these must be cancelled.
Post-condition (output state)	All generate events are stored for a customer The agreed_order state with a Service provider is update in case an orderEvent is shared. The agreed_order state is towards 'cancelled' in case a cancelation is shared.	

The previous process may require an update if the location given by an unload or ETA event that has been received is not equal to the unload location given by the customer and there is no adjacent leg. In that case, the adjacent leg may have to be cancelled or start at the new location. An example is where a container that has been discharged in Antwerp was expected to be transported from Rotterdam to its destination.

In the previous case, it could also be that all adjacent legs are cancelled and a new adjacent leg to the destination is to be organized.

6 Development plan

6.1 Milestones

The development plan has the following milestones:

- **IATA Hackaton** (June 2023) – initial version. Initial stakeholders from Spain, Finland, IATA, and the Netherlands develop a first version of the multimodal visibility service and demonstrate its use. At least IATA and the Dutch team will share data, where the Dutch team will operate as Dutch Customs. The other pairs still must set up their use case. From the Dutch perspective, the hackaton is succesfull if Dutch Customs can share and access data of airlines via the IATA bridge.
  In terms of functionality, the following approach is taken:
  - VCs – at least a demonstrator is setup, at most it is integrated with the FEDeRATED node.
  - Data sharing – at least the events that are specified in this document can be exchanged and mapped with existing events of stakeholders and B2B/G2B queries (AWB, eFTI, eCMR, etc.) support data access. There is not event logic, event distribution is simplified based on location codes (see before), and no federated querying (query to the data source via one or more data holders).

  Additionally, the potential of generating a video applying AI will be explored, thus showing the functionality.
- **Evaluation** (June 2023)– the IATA Hackaton will be evaluated by the FEDeRATED architecture group and next steps will be formulated.
- **FEDeRATED Hackaton** (October 2023) – set up of a common pilot with the participants given in this document.
- **Final event** (November 2023) – all functionality of this document must be implemented by at least the FEDeRATED node, including the Service Registry for localization as part of the VCs. It must be easy to include a new node in the infrastructure (on-boarding), upgrade the capabilities of a node, and upgrade the functionality of the infrastructure. This must be demonstrated and potentially is the basis for a training to interested users. This document serves as a specification to deploy a multimodal visibility service.
- **Final delivery or results** (first quarter 2024) – this document is discussed within the FEDeRATED Architecture Team and amended accordingly for delivery as an example of a Technology Independent Service.

The first iteration of the Service Registry will only constitute the SHACL documents, since support of interaction patterns requires an extension of the tool (Semantic Treehouse) that is currently applied for constructing the Service Registry. A second iteration is expected to support interaction patterns.

Basically, the Service Registry for localization implies specification of additional constraints to the SHACL documents of an interaction pattern. An approach is to use an existing module where the basic functionality is to include these extra constraints. In fact, it means deleting constraints in the SHACL documents of the multimodal visibility service (states, events, etc.) by selecting those that are applicable for a local interface, and thus create a new SHACL that is a subset of the existing one.

The planning is organized according to these milestones. Detailed planning of towards the final event is after the evaluation of the hackaton and completion of the specification.

The planning towards the hackaton is to complete all openAPIs for the events and the specified queries, a simple tool for localization, and the operation of multiple nodes in the Azure cloud environment of TNO.

6.2 Activities for preparation of the FEDeRATED hackaton

The proposed approach to the FEDeRATED hackaton is to implement the openAPIs described in this document. These openAPIs are generic, they are made specific by including specific SHACL validations. Thus, everyone will have the same openAPIs, but the functionality of these APIs is limited by the SHACL validation.

The SHACL validation is a combination of modality and cargo type, see section 4. There are five modalities (sea, air, road, rail, inland waterways) and four cargo types ((sea)containers, trailers, goods or pieces (pallets, packages, etc.), ULDs (Uniform Load Devices)), leading to a large number of variations. There are two constraints: ULDs are only transported by air; (sea)containers are not transported by air.

The objective of the FEDeRATED hackaton is to prototype eFTI. This implies the following SHACLs must be generated for the (un)load – and ETA event (the proposal is not to implement the incident/accident or position event): road, truck, trailer (optional), goods, (sea)containers Additionally, a query like the one implemented for Codognotto will be implemented for data retrieval by an authority.

Some Living Labs may require additional SHACLs:

- Deplide – Simple: a SHACL for rail, train, wagon, container. There is no SPARQL formulated yet.
- Airline (IATA) – Dutch Customs: a SHACL for air, pieces, and ULD. A SPARQL for AWB data based on One Record may have to be specified.

The following activities (with proposed due dates) must be performed for the FEDeRATED Hackaton:

Activity	Responsible	Due date
Installation of nodes in the TNO Azure environment	Stephan	
Specification of queries and results (at least a B2B and G2B	Wout	See this document
Generation of openAPIs and SHACLs	Theodor	
Configuration of the nodes with the SHACL and RML	Stephan supported by Theodor	
Generation of SHACLs for the queries	Theodor	
Availability of the openAPIs to hackaton participants	Stephan	
Simple configuration of the event distribution mechanism (it must be included in the previous step)	Stephan	
Integration of the LLs with the published openAPIS and SHACL	All participants	During the hackaton?

Activity	Responsible	Due date
Presentation or video of the solution	Wout, Theodor, and other FEDeRATED participants of the hackaton	During or after the hackaton
Finalization and discussion of the document	FEDeRATED Architecture group	First quarter 2024

7 Concluding remarks

This document illustrates how a Technology Independent Service can be specified applying the concepts of the FEDeRATED semantic model, not only those of event and Digital Twin, but also the data sharing concepts. It also shows that a proper application of these concepts will result in an operational data sharing infrastructure, although not all functionality is supported yet.

The example of TIS given in this document, a Multimodal Visibility Service, is not yet complete and has not yet been validated with users. This would be a second step after validation by the FEDeRATED Living Labs. The document also shows that an initial implementation could be sharing only visibility events, whereas later event logic and value-added functionality can be implemented, resulting in a smart data sharing infrastructure.

By taking the physical environment as leading, i.e. events are generated when physical actions are performed, and implementing the value added functionality, all relevant stakeholders can be informed. This might require additional services like ETA calculation services. The result will contribute to synchronization of logistics activities, thus improving efficiency and potentially contributing to effectiveness.

Concluding, a stepwise approach can be taken contributing to business performance.