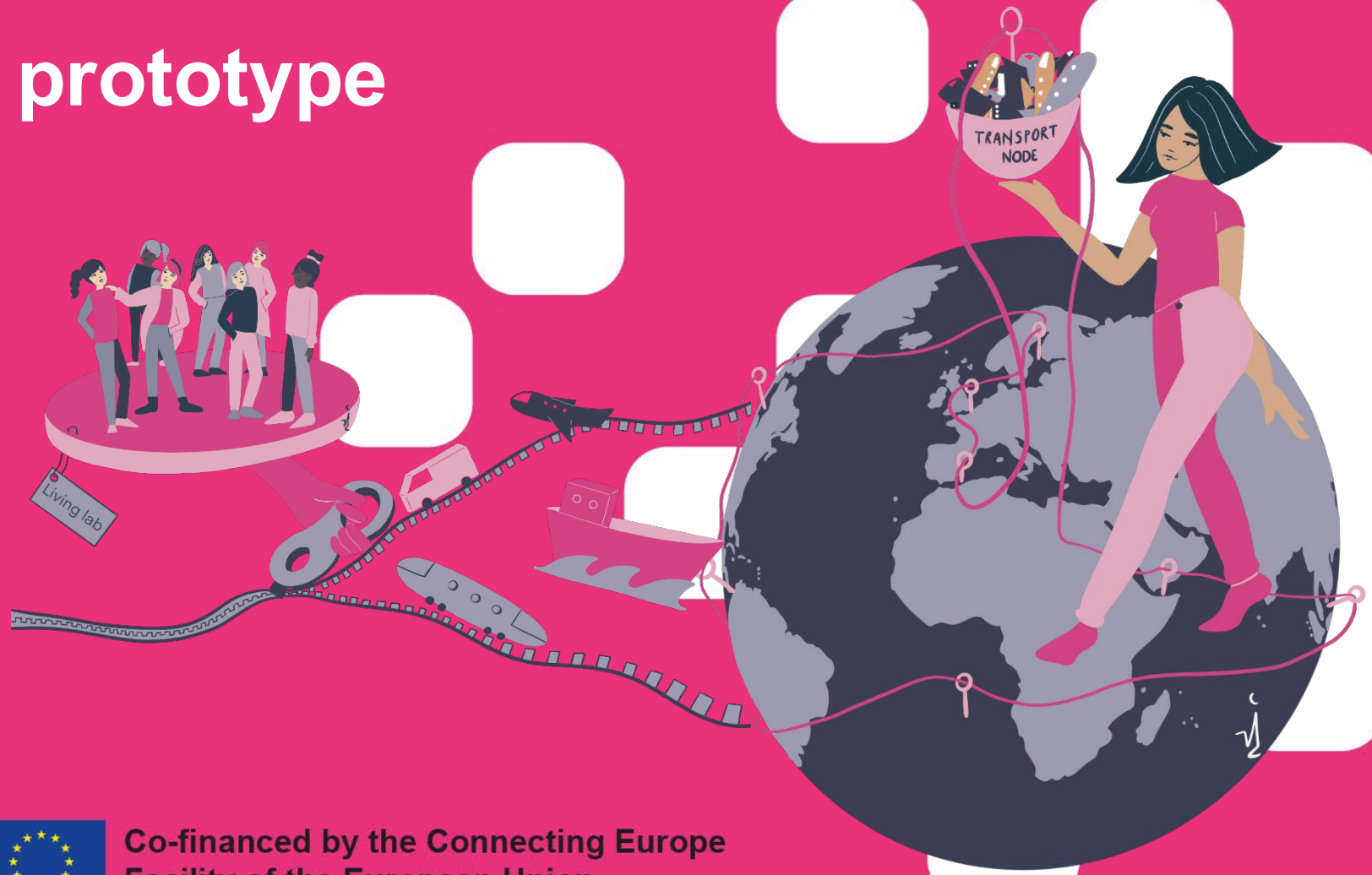
FEDeRATED node prototype development

From concept to implementation



Wout Hofman
(wout.hofman@tno.nl)

FEDeRATED
NETWORK OF PLATFORMS

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

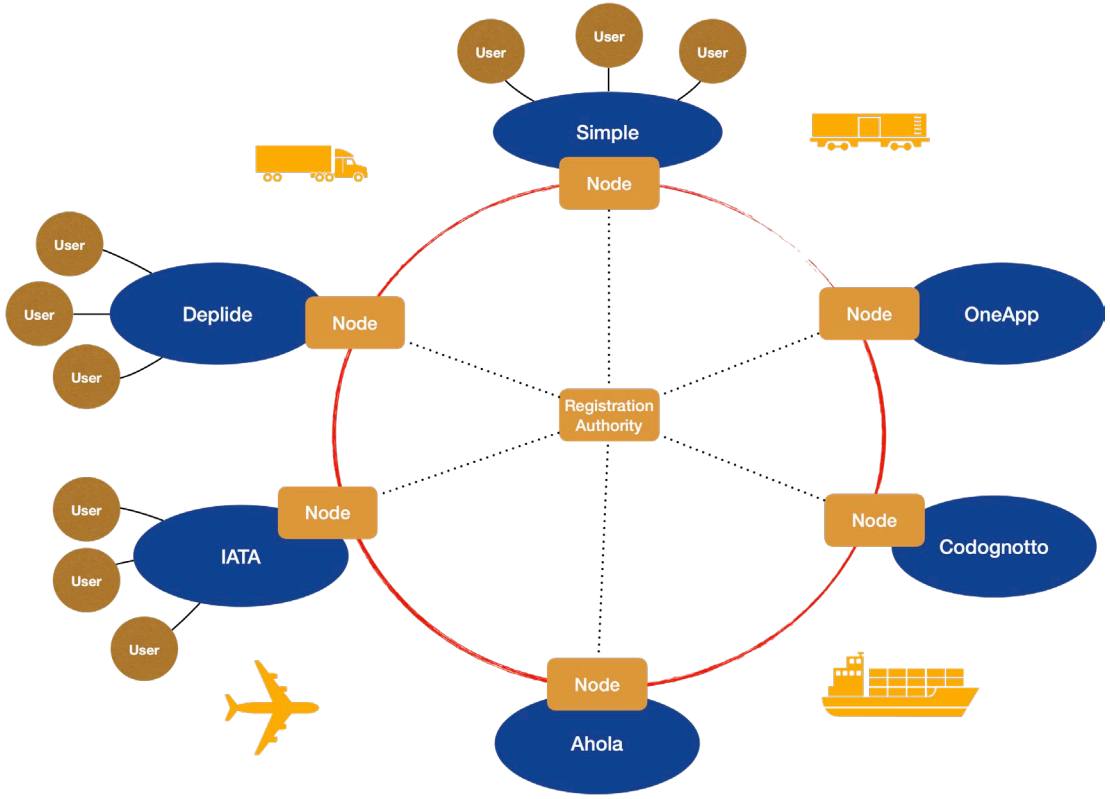
How to become part of the network with a node

Each participant:

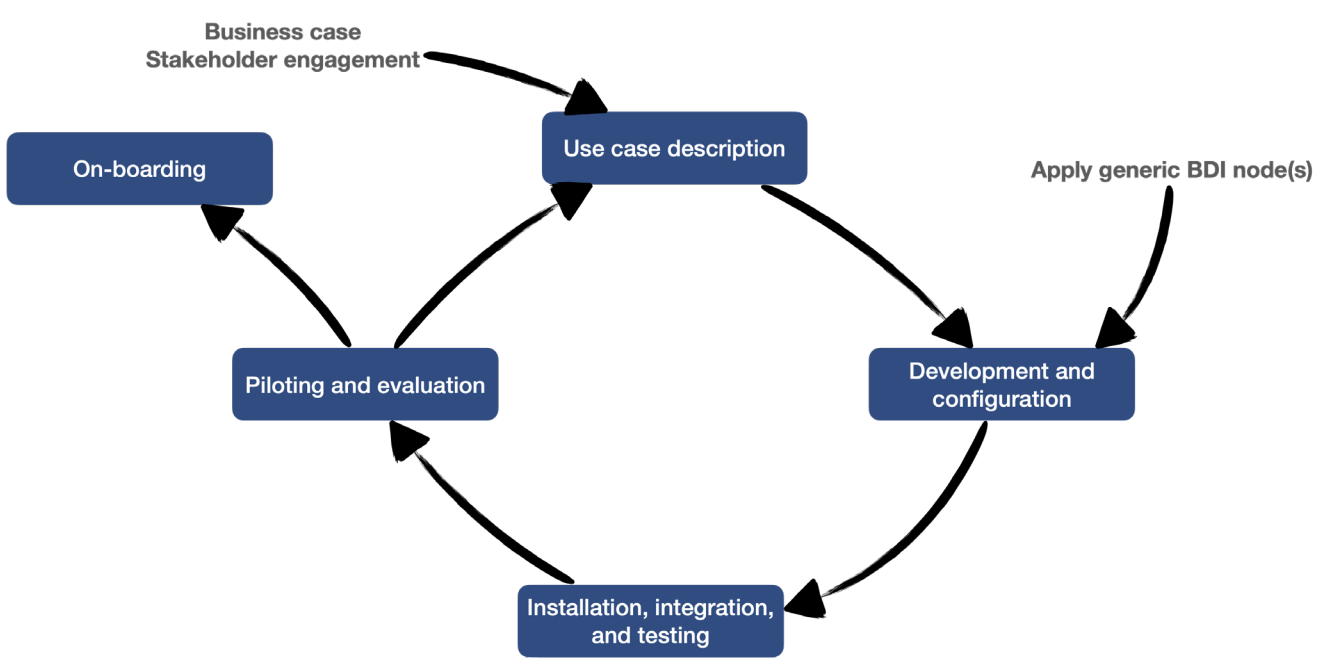
- Downloading and installing a ’node’
- Integrate (open)APIs of a node with internal IT systems

Use case driven Semantic Adapter and APIs:

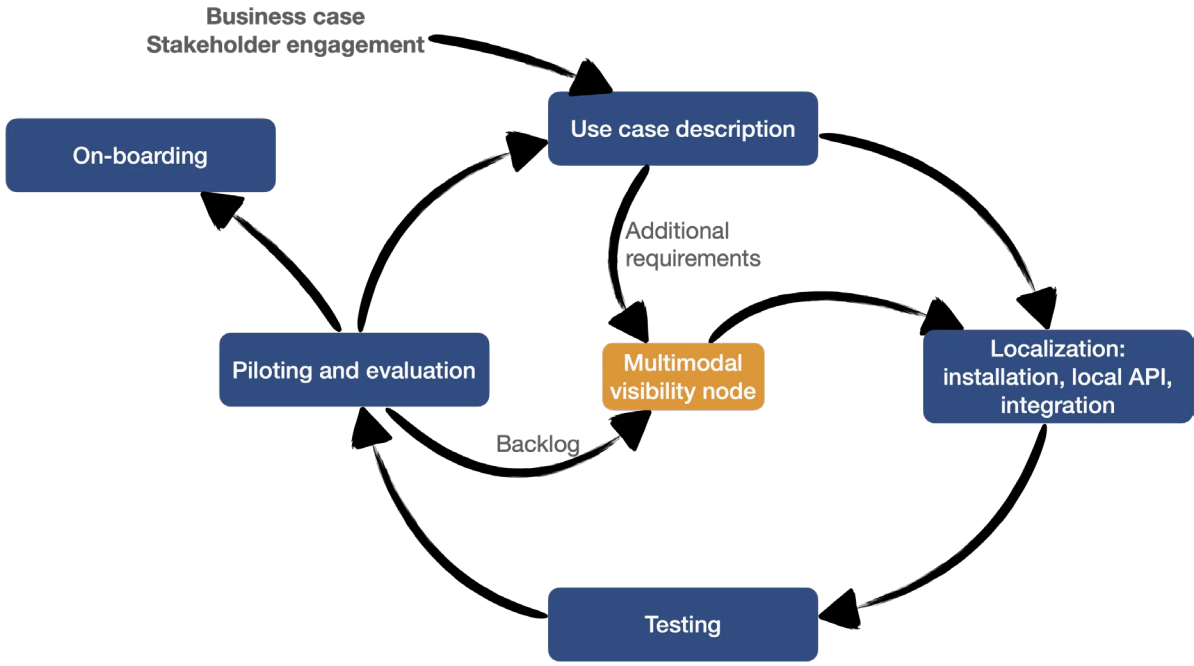
1. Common to all participants for their use case
    - Identical APIs for a use case
    - Use case specific semantic adapter
2. Integration of a common ‘multimodal visibility’ use case (not yet supported):
    - Localization by each participant (local APIs) to cover its modality/cargo type/…
    - A common multimodal semantic adapter.



From case description to on-boarding



Use case specific node

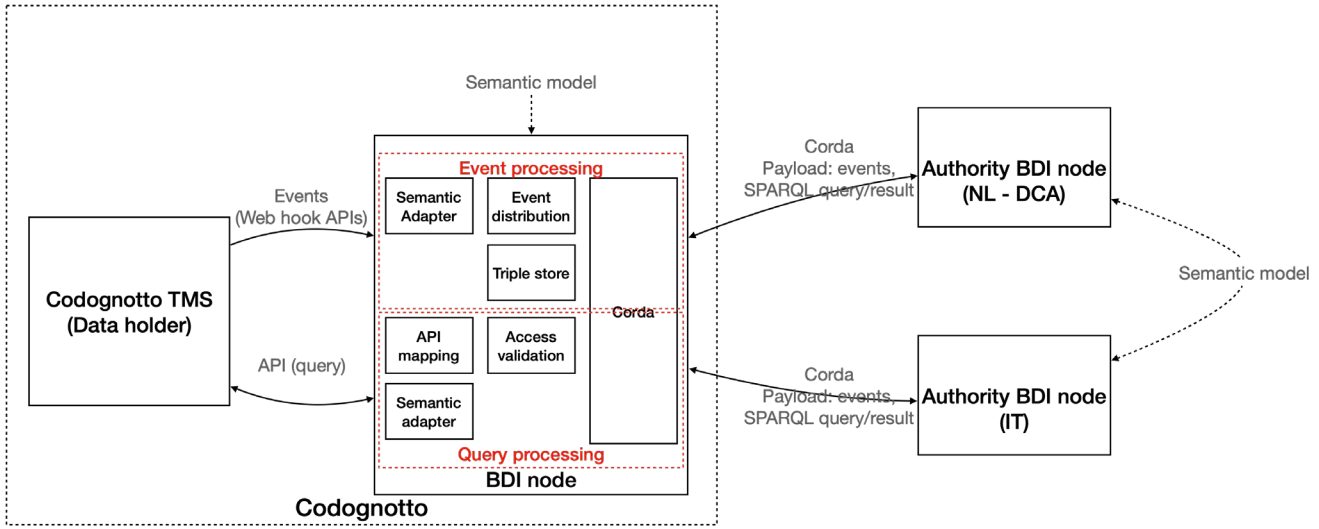


Applying a multimodal visibility node

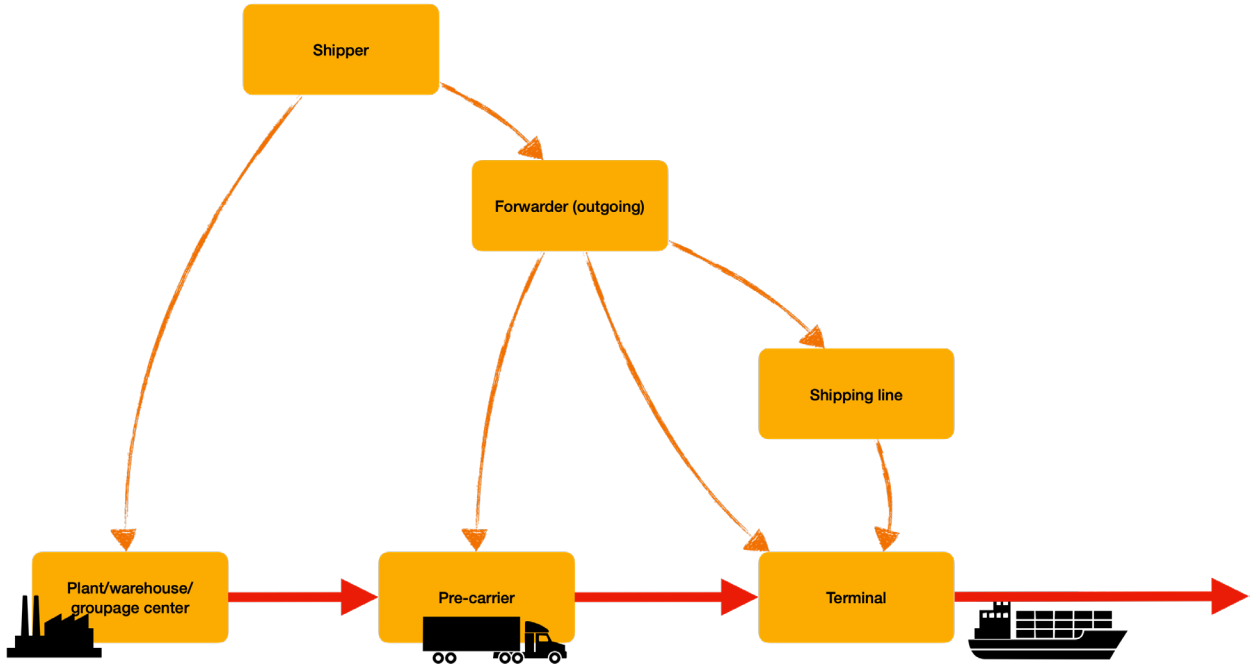
Case description– simple – and complex use cases

- Complex use case (next slide):
  - Hierarchy of commercial relations and compliance (transaction hierarchy)
  - Event sequencing (e.g. UML sequence diagrams) → event logic and - distribution
- Functional specification of events
- Functional specification of linked data (e.g. eCMR data set) → SPARQL query
- Assign data holder/data user roles
- Specify technical setting (systems and platforms to be integrated with a BDI node)

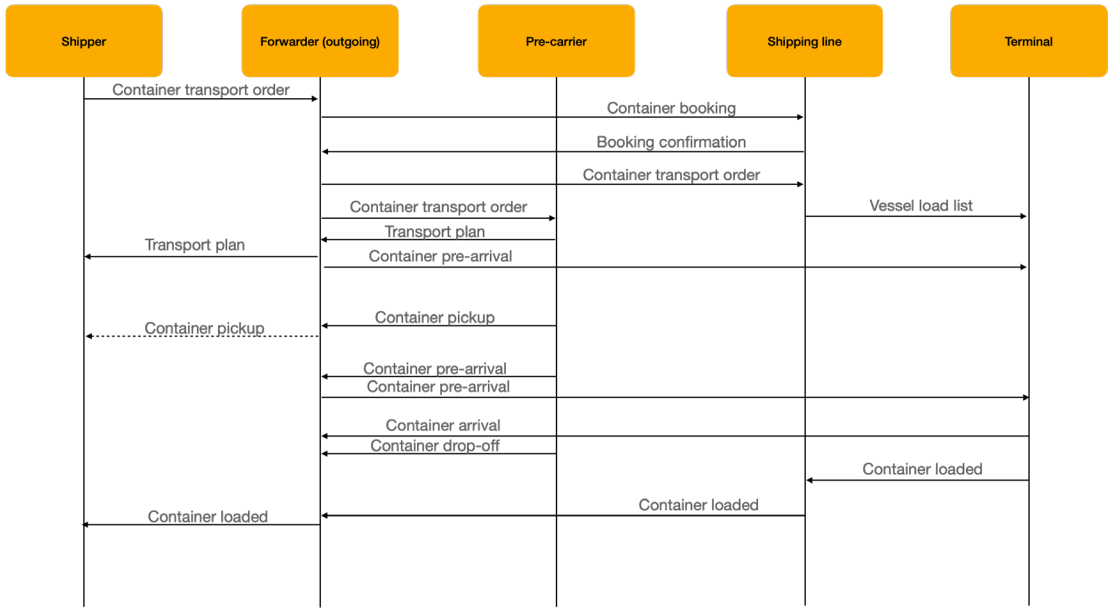
	Trip	Border crossing	Arrival	Load	Arrival	Load	arrival
Event (trip)							
UUID (event)	✅		✅	✅	✅	✅	✅
UUID (sender organisation)	✅		✅	✅	✅	✅	✅
milestone	start / end		start	start	end	start	end
estimated date/time	✅						
actual date/time			✅	✅	✅	✅	✅
UUID (location)	✅		✅ (rail terminal)	✅ (rail terminal)	✅ (ferry terminal (R'dam))	✅ (ferry terminal (R'dam))	✅ (ferry terminal (UK))
External reference	✅		✅				
Reference type	trip number		eCMR data set				
UUID Digital Twin (truck)	✅						truck or trailer
UUID Digital Twin (trailer)			truck or trailer	train	train	ferry	✅
UUID Digital Twin (goods)	✅		✅	✅	✅	✅	
Event (border crossing)							
UUID (event)	✅	✅					
UUID (sender organisation)	✅						
milestone	start	end					
estimated date/time	✅						
actual date/time		✅					
UUID (location)	✅	✅					
Organisation							
UUID organization	✅						
Organization name	✅						
Infrastructure (location)							
UUID	✅						
Name and address (example)	✅						
Digital twin - goods							
UUID Digital Twin	✅						
conssignment ID	✅						
item number	✅						
Digital twin - transport means - truck/trailer							
UUID Digital Twin	✅						
License plate	✅						
Country of registration	✅						
Digital Twin - equipment (trailer)							
UUID	✅						
Equipment ID (trailer license plate)	✅						
Event (truck composition)							
UUID (event)	✅						
UUID (sender organisation)	✅						
milestone	start						
estimated date/time							
actual date/time	✅						
UUID Digital Twin (truck)	✅						
UUID Digital Twin (trailer)	✅						



Example of a transaction hierarchy and sequence diagram



Transaction hierarchy



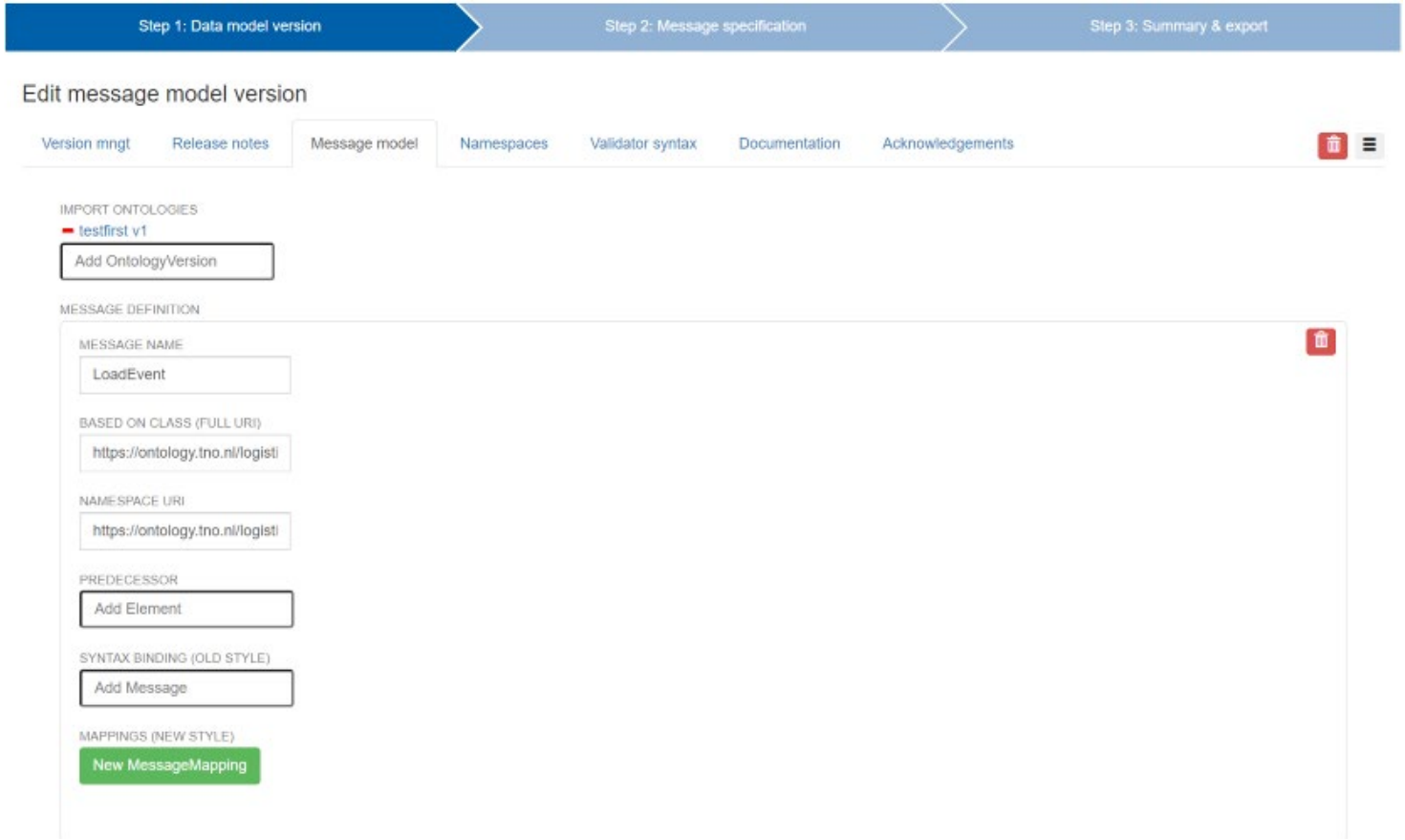
Sequence diagram

Development

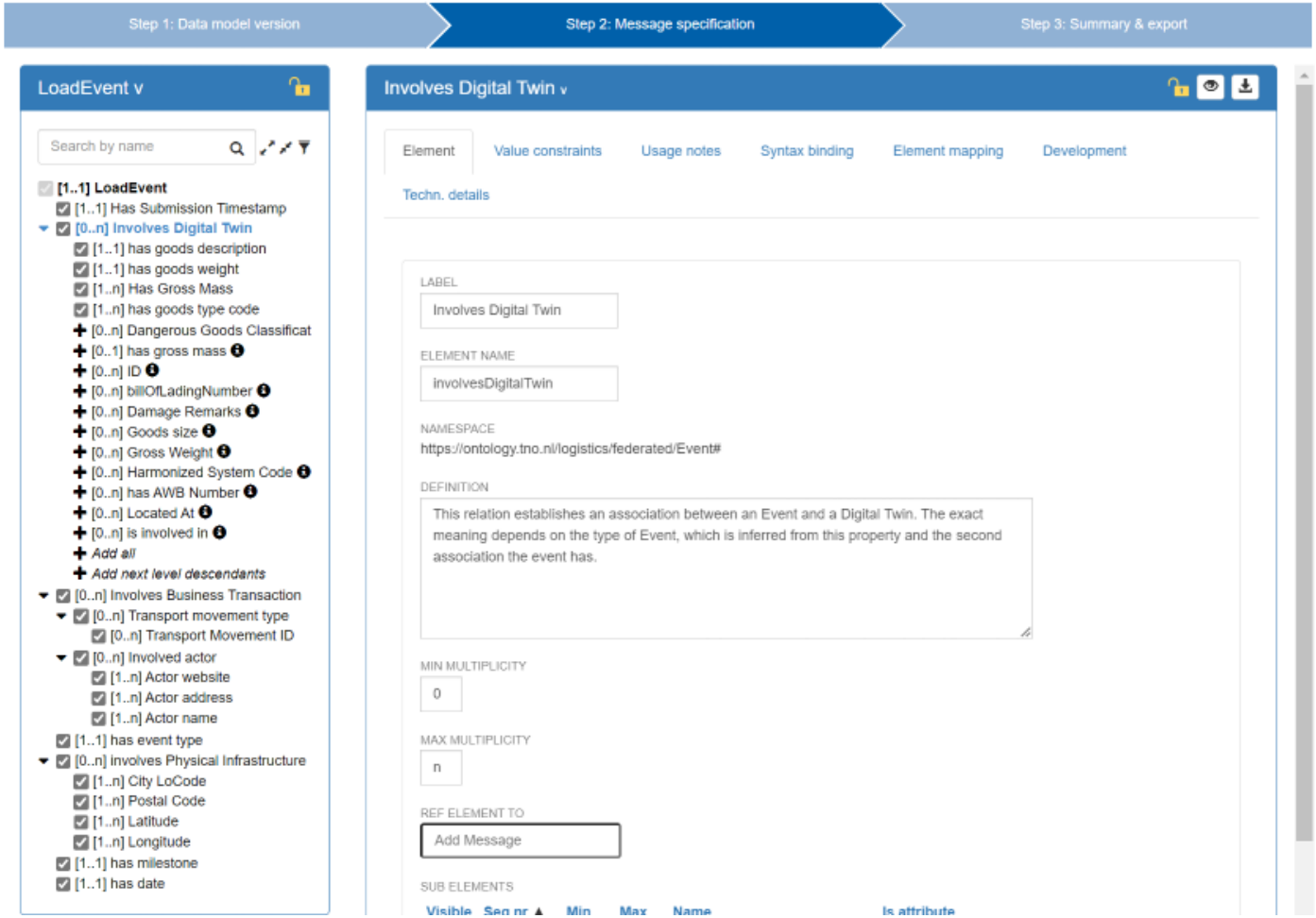
- Map functional specifications to the FEDeRATED ontology (initial version of Service Registry)
  - Events and SPARQL queries
  - Generate SHACL (SHACL validation)
  - Generate technical specification openAPIs
  - Generate RML (Rule Markup Language) for mapping JSON to RDF
  - Generate API code and semantic adapter → basis for integration with back office system(s)
- Configure event distribution – change code in API code
- Potentially include event logic (not yet supported) → requires specific code, inclusion of a 'pipeline' like Apache Camel

Slide with screendumps of STH from Theodor specifying the events
Example various steps and example APIcode generated by Stephan

STH FIT Wizard: Step 1



STH FIT Wizard : Step 2



STH FIT Wizard : Step 3 (Open API Specification)

Step 1: Data model version | Step 2: Message specification | Step 3: Summary & export

Configure your Semantic API using the following output

XML syntax | JSON syntax | RDF syntax | CSV syntax | OpenAPI Specification syntax

Open API Specification

◉ YAML ○ JSON

```
openapi: 3.0.0
info:
  title: LoadEvent
  version: undefined
  description: 'This OAS Specification was generated by Semantic Treeh
servers:
  -
    url: 'https://example.org/api/v1'
    description: 'The Example API server.'
paths:
  /LoadEvent:
    post:
      summary: 'Create a resource of type LoadEvent.'
      description: 'Create a resource of type LoadEvent.'
      requestBody:
```

Message Example


```
{
  "hasSubmissionTimestamp": "2023-04-18T16:20:55+02:00",
  "involvesDigitalTwin": [
    {
      "goodsDescription": "b91af5 37562",
      "goodsWeight": 1,
      "hasGrossMass": [
        1
      ],
      "goodsTypeCode": [
        {}
      ]
    }
  ],
  "involvesBusinessTransaction": [
```

STH FIT Wizard : Step 3 (RML Mapping)

Step 1: Data model version | Step 2: Message specification | Step 3: Summary & export

Configure your Semantic API using the following output

XML syntax | JSON syntax | RDF syntax | CSV syntax | OpenAPI Specification syntax

JSON schema — YAML / JSON

```
id: 'https://ontology.tno.nl/logistics/fe
$schema: 'https://json-schema.org/draft/202
title: 'LoadEvent version undefined'
description: 'Generated by Semantic Treehous
required:
  - hasSubmissionTimestamp
  - hasTimestamp
  - hasMilestone
  - hasDateTimeType
additionalProperties: false
properties:
  hasSubmissionTimestamp:
    type: string
    format: date-time
  involvesDigitalTwin:
```

Example

```
{
  "hasSubmissionTimestamp": "2023-04-18T16:2
  "involvesDigitalTwin": [
    {
      "goodsDescription": "05b05e 68d15",
      "goodsWeight": 1,
      "hasGrossMass": [
        1
      ],
      "goodsTypeCode": [
        {}
      ]
    }
  ],
  "involvesBusinessTransaction": [
```

RML

```
@prefix rml: <http://semweb.mmlab.be/ns/rr
@prefix ql: <http://semweb.mmlab.be/ns/ql#>
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ns0: <https://ontology.tno.nl/logist

[]
  rml:logicalSource [
    rml:source "http://www.example.com/root"
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$"
  ] ;
  rr:subjectMap [
    rr:termType rr:BlankNode ;
    rr:class ns0:LoadEvent
  ] ;
```

Step 4 – code generation

- OpenAPI code for testing
- Semantic Adapter:
  - JSON → RDF
  - RDF → JSON-LD

Step 5 – containerization (Docker, Kubernetes)

Where to find the code?

https://github.com/tno/federated-bdi

- Source code
- Technical documentation
- Unit and integration tests
- Gitlab CI pipeline
- Configured for a demonstration use case

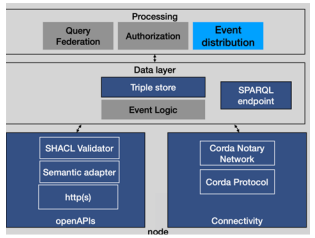
https://github.com/federated-bdi/docker-bdi-node

- Docker node

https://github.com/federated-bdi/Kubernetes-bdi-node

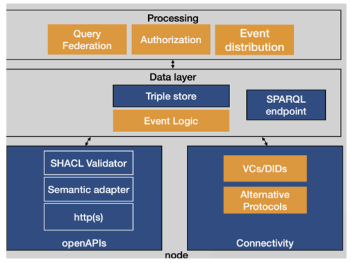
- Kubernetes node

The node functionality



Existing

- openAPIs with internal IT systems
- Sharing of triples (RDF) between nodes (events with links to data)
- Simple semantic adapter (openAPIs to triples)
- Data validation (correctness and completeness according to specifications)
- Data storage with a triple store providing an endpoint for querying (SPARQL)
- Simple data distribution mechanism (type of 'smart contract')
- Connectivity: Corda-based
- On-boarding: Corda network manager
- Non-repudiation (log/audit trail): Corda Notary Network



Planned

- Improved data distribution mechanism
- Authorization based on links
- Access policy evaluation based on semantic model
- Event logic to validate states
- Query federation (data provenance)
- Support of other connectivity protocols
- VCs/DIDs for on-boarding
- Other types of non-repudiation (log/audit trail)
- Improved semantic adapter