MASTERPLAN
CONFIGURATION AND INTEGRATION OF A NODE

EventAPIs

This is quick guide for configuration and integration of a node with a backend system. It is about eventAPIs as supported by the openAPIs of the node. These openAPIs are identified as follows in the FEDeRATED Reference Architecture (chapter 9)

Index APIs	Description	Node APIs	Backend APIs
eventAPIs	A set of openAPIs to share events and access data of links shared by these events.	POST Event GET Event GET Data GET State POST State	POST Event GET Data

The Node APIs are those supported by a node and can be called by an IT backend system. The Backend APIs need to be implemented by an IT backend system and will be called by a node. The POST Event API of the Backend APIs is a webhook API; it can be configured.

The logic of these openAPIs is as follows:

- POST Event Node API – an IT backend system of a data holder posts an event with (optional) link(s) to its node. This event is shared with a data user node
- POST Event Backend API – a data user receives an indication that a new event has been received in its node, including the identification (UUID) of that event.
- GET Event Node API – an data user retrieves the event from the node, including the optional links.
- GET Data Node API – the data user requests data provided by a link of an event. This data is to be retrieved from the data holder that posted the event to its node. This is called 'link dereferencing'.
- GET Data backend API – upon receiving of a link for which data is requested, a data holder node initiates this API to retrieve data from its backend system. This data is converted to semantic data (RDF) by a data holder node and shared with the data user node as response to the previous GET Data Node API.

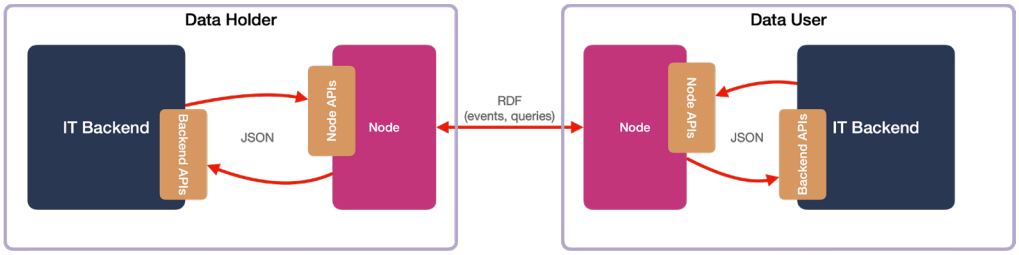
The GET – and POST State Node APIs support a type of publish and subscribe mechanism. It is not always applicable, especially not for providing data to supervising authorities. These two Node APIs are replaced by what is called 'event

distribution': a configuration by which a supervising authority receives events for a regulation in its competence area. An example is:

- Authority: Customs Authority.
- Regulation: Battery Regulation.
- Competence Area: the Netherlands.
- Policy: a data structure specifying the access rights for the combination of the previous data properties.

The 'policy' is the basis for the GET Data Backend API available to a supervising authority. In case this policy is re-using commercial data, it is a filter on that data set. Of course, an enterprise must implement the mandatory data requirements of the policy.

The setting is as follows. Each IT Backend system must implement the Backend APIs that are called by a node. An IT Backend system can call the Node APIs. Data is shared between nodes by triples (RDF). Between an IT Backend system and a node, JSON is the current syntax supported.



The JSON structure between IT backend systems and a node reflects the structure as implemented by RDF. Additional scripts can be developed that support other syntaxes and/or divergent JSON structures.

A setup consists of minimal two nodes, a data holder – and data user node. Both nodes can integrate with IT back-office systems of each organisation and/or can have a GUI. The integration and GUI is not part of provision of a node. Integration with an IT back-office system is according to the (data) interface provided by the semantic adapter.

The following steps must be taken:

1. **Installation** – each participant downloads and installs a node in its own domain. As alternative solution, a node can temporarily be installed (during a project) in the TNO domain.
2. **Configuration** – this is by applying Semantic Treehouse (prototype of the Service Registry) for generating RML (RDF Mapping Language) and SHACL (SHApe Constraint Language) files.
3. **Integration** – this is about the integration with an IT backend system.

Installation is via the GitHub repository with an installation manual.[1] Installation requires the setup of non-repudiation, i.e. the node(s) that implement an immutable log and audit trail. This can be a node operated by an authority.

Integration is based on the configuration of a node since configuration defines the events with links to access additional data.

In case integration with IT backend systems is considered at a later stage in a pilot or Living Lab, a generic GUI can be applied. It interfaces with a node. In this case, a temporary data store needs to be installed that contains the relevant data. This can be the triple store of a node.

Configuration

Configuration is about specifying data that is accessible via sharing links with events. Thus, the structure of these data sets and events must be specified.

These data – and event structures are the payload of the eventAPIs and their support by a node. The Node APIs are generic in supporting all types of events and sharing data. The Node APIs are configured as follows:

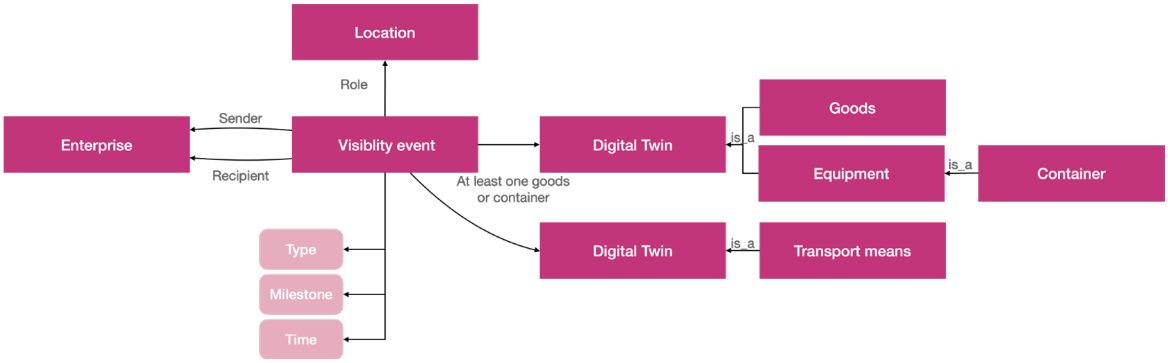
- Semantic Adapter – transformation of between the JSON eventAPI payload and RDF. This is by RML (RDF Mapping Language).
- Validator – data quality validation (completeness and correctness). This is by SHACL.

The Backend API GET Data can be generated as Open API Specification document for its implementation by an IT Backend System.

The **specification steps** are:

1. **Events**. The events with links to data must be agreed between the data holder and – user or specified by the policy of a supervising authority (see before). There are two types of events that can be specified, namely:
    a. **Transaction events**. These are the basis for a commercial transaction between enterprise. They provide information as to whom has provided a service or product. They are not (necessarily) required by a supervising authority.
    b. **Visibility – or association events**. These specify an association (in time and place) between two real-world objects that are represented by Digital Twins. For instance, a container transported by a vessel or a battery used in a car. These types are relevant to enterprises and supervising authorities. A generic structure for visibility events is shown in the following figure. A similar figure can be made for the association between products e.g., a battery in a car.

[1] GitHub - Federated-BDI/FEDeRATED-BDI: Data sharing infrastructure prototype made for the FEDeRATED project, federatedplatforms.eu. Docker and Kubernetes versions available in a separate repository.



2. **Data retrieval – access policy**. Based on links shared via events, the data that can be pulled by a data user from a data holder is defined. Like said, this is link dereferencing. It can be data of any Digital Twin referred to via the event that has been shared. In case of a visibility – or association event, these are details of for instance a container or a battery. Thus, the data set as a basis for an access policy is an extension of that of the event that has been modelled.
   This data set is retrieved from the data holder IT backend system (via the GET Data Backend API) and available to the data user IT backend system (via the GET Data Node API).

When specifying the events and the required data set for access policies, it is recommended to take events and data that are supported by existing APIs of a data holder and see how far they match data (search) requirements of a data user.

Using the prototype Service Registry tool (Semantic Treehouse), RML and SHACL for configuration of a node for event sharing and link dereferencing, and an OAS for the access policy (GET Data Backend API) is generated. The OAS can be implemented by a data holder on its IT backend system.

Data

The recommendation is to have real data for testing and validation of the functionality. Participants of a use case, preferably with a data holder role, must provide this data sets.These data sets must be available for testing but can also function as inspiration during development and intermediate testing of the functionality by a developer. Thus, it is recommended to have this data as soon as possible available.

Testing

It is recommended to initiate the deployment of a node in ones own environment as soon as possible. Potentially organisational and technical issues need to be addressed before a node can be deployed in one's own IT environment.

Testing is the final step before actual deployment of the use case. The focus is on testing the configurations and the integration with IT backend systems. These are tested in two steps:

1. **Testing of the configurations** – the semantic adapter and SHACL validator for the events and queries are tested individually.
2. **Lab testing** - two (or more nodes, depending on the use case) must be installed and events and links for link dereferencing need to be shared between those nodes.
3. **Live testing** – this is about testing the nodes in their intended technical environments.