9

i3-MARKET Semantic Model Repository and Community

The results are shared not only with project partners but also with stakeholders and community in open-source repositories. As part of open-source assets, the data models, documentations, and files used in the i3-MARKET project are made available, such as the following:

- The i3-MARKET *data pack* is the set of files, schemas, and metadata model diagrams that represent the way the i3-MARKET semantics is organized and structured; it also contains the metadata in two different formats, e.g., ttl and Jason-ld. owl.
- The i3-MARKET *semantic model* info is the documentation that describes in detail all the taxonomies and vocabularies from needed domains used in i3-MARKET and that describes and represents all the relationships between them to build the graph representation of the i3-MARKET semantic model.
- The *support* repo is the mechanism for how the data model is maintained following the interoperability requirements in i3-MARKET. If you want to contribute or have any suggestion for improving the semantic models, visit the open-source repositories and contact authors and members.
- The *model files are shared in i3-MARKET GitHub/Gitlab repositories* with release versions where each section contains the online machine-readable files in OWL and other formats for online accessibility. The files are maintained and updated regularly to keep the latest version of the model files up to date.

The code as well the models and vocabularies are available open-source via the establishment of the i3-MARKET spaces on Gitlab available at: https://gitlab.com/i3-MARKET-V3-public-repository/ and GitHub available at: https://github.com/i3-MARKET-V3-public-repository/.

i3-MARKET semantic model governance process, which is defined as the support and evaluation process to include semantic improvements, is as follows:

- **Request for changes or updates:** Identify any changes prior to a *major* release, which should be considered private and usually is on testing and pre-consensus/staging.
- **The evaluation of any type of update request:** A review from editors and community, approves participation, and updates. In particular terms, vocabularies, ontologies or initiate a model extension in the i3-MARKET OSS project.
- **The communication of the results from technical experts:** A tagging version using alpha, beta, and gamma versions and then tagged as major is used here.
- **Evaluation of contributions for new commits:** Technical experts, PM, TM, TPMs, WPLs, and TaskLs assess and evaluate the contribution, including documentation at the initiated project in i3-MARKET OSS.
- **Reports and changes report:** The technical board issues a short report, explaining the rational on the rejection in exceptional cases; this step can include rejecting/cancelling project participation.

It is possible to find a more complete definition of the attributes used in the data offering description schema template as used in the semantic engine API in Appendix A.

9.1 Semantic Engine (SEED)

The semantic engine and framework solution is available and integrated into the i3-MARKET Backplane. Another concept is the metadata semantic registry stored in a registry database (like MongoDB). With this feature, the Backplane can rely on the metadata registry storage capacity to collect the semantic information about the assets and information for the marketplaces and stakeholders that can be created, searched, retrieved, and manipulated for external and internal operations.

Semantic engine framework:

From an operational perspective, i3-MARKET envisages semantic engine components (e.g., SEED) to manage query mechanisms on top of the registry catalogues, including complex discovery and retrieve checks that

make sure, e.g., that the necessary information is retrieved by the actors and services. Also very important are the functionalities related to the creation and registration of the data offering descriptions and the management of local and federated registries. The data offerings can be shared by providers/marketplaces in the i3-MARKET network and the engine can search, discover, and retrieve the data offerings, which are authorized, from all the nodes/marketplaces.

The engine also has functionalities and interfaces that are used in conjunction with other Backplane components/systems to compile and fill information and details for the functionalities, for example, for notification manager, smart contract manager, data access & transfer, and BESU.

Semantic engine and metadata framework:

a. Data offering creation
b. Data offering discovery
c. Data offering registry
d. Federated discovery on different instances
e. Management of data sharing agreement and service agreement parameters to comply with contract manager operations
f. Alignment with entities and IDs in Backplane information models

We developed and implemented dedicated software components for semantic engine system as SEED, which is in charge of managing the semantic metadata, descriptions, queries, discoveries, retrieving, creating, and mapping descriptions and manipulating registries, federated queries, and component interactions and interfaces. To make easier the interface and use of functionalities, we present the external operations via APIs that are more agnostic and easier to use also for non-semantic experts.

9.2 Technical Requirements

For the semantic storage, the following high-level capabilities have been defined:

1. Semantic metadata management:
   The semantic engine (SEED) relies on a local MongoDb and Hyperledger-BESU. All the information, for instance, data provider, data offerings, consumer, and querying offering, are stored as semantic data.

Name	Description	Labels
Metadata storage (MongoDb)	The registry storage (MongoDb) is responsible to store semantic data and process the queries. The storage should provide either the REST endpoint or client connector so that other components can access to the data	**Epic**
Spatial and text data storage	To support spatial and full-text search queries, the semantic data manager should be able to index spatial and full-text data	**Epic**

Name	Description	Labels
Save semantic data	As a subject, I want to save my semantic metadata so that I can query and update it later Subject: Data Consumer, Data Provider	**Epic**

2. Offering registration:
   The semantic engine exposes APIs to register, query, and update offering. A data provider can regist er offering, for instance, datasets and the price for data, etc.

Name	Description	Labels
Offering registration	Offering registration is a component that allows the user to manage their data offering. More specifically, it provides the following functionalities: Register the data offering – Retrieve all the offerings – Update/delete offerings – Subscribe to an offering	**Epic**

3. Offering discovery:
   The semantic engine exposes APIs to query the existing offerings in i3-MARKET Backplane. A data consumer can query datasets, prices for any dataset, offering, etc.

Name	Description	Labels
Offering discovery	Offering discovery is a component that allows the data consumers to search the offering data available on the marketplace. The data consumer has to specify the characteristics of the data they are looking for. The offering discovery module will then process the data request and returns a list of available offering data that meet their requirements. More specifically, the offering discovery should provide the following functionalities: Register the consumer data request – Retrieve all the data request of a consumer – Process a data request – Update/delete a data request – Subscribe to a data request	**Epic**

4. Vocabulary management:

Name	Description	Labels
Vocabulary management: Semantic model management	Vocabulary management is a component that is used to manage the i3-MARKET semantic data model. More specifically, the vocabulary management should provide the following functionalities: View and search the concepts of i3-MARKET – Allow the user to propose a new concept – Allow the administrator to add a new concept – Allow the administrator to update/delete an existing concept	**Epic**

Backlog release – features:

Name	Description	Labels
Semantic data manager: registry storage	The registry/semantic storage is responsible for storing semantic data and processing the queries. The storage should provide either the REST endpoint or client connector so that other components can access the data	**Semantic Data Storage Offerings**

Features:

Name	Description	Labels
Save semantic data	As a subject, I want to save my semantic metadata so that I can query and update it later Subject: Data Consumer, Data Provider	User Story

Tasks:

Name	Description	Labels
Define a semantic description template	As a data provider, I want to create a semantic description of my offering data so that I can register it to i3-MARKET. It would be desirable that the semantic engine should provide a semantic description template so that the data provider can easily fill in to register the offering data	User Story

9.3 Solution Design/Blocks

Figure 9.1 shows the final version, which is defined as:

- Components and functionalities of semantic engine and framework
- New versions of i3-MARKET semantic models

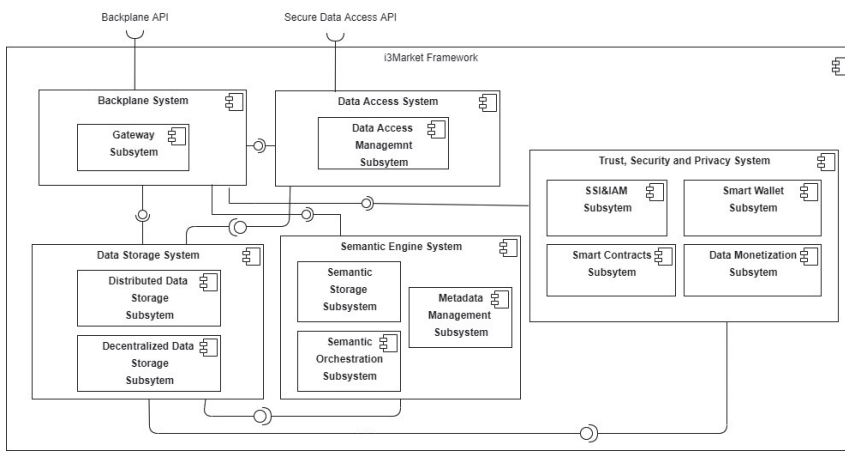


Figure 9.1 High-level Backplane block diagram.

– Semantic vocabulary management environments

Figure 9.1 shows that we use BESU SEED-INDEX library in order to retrieve all registered nodes addressed in the network and hence enabling the federated query search.

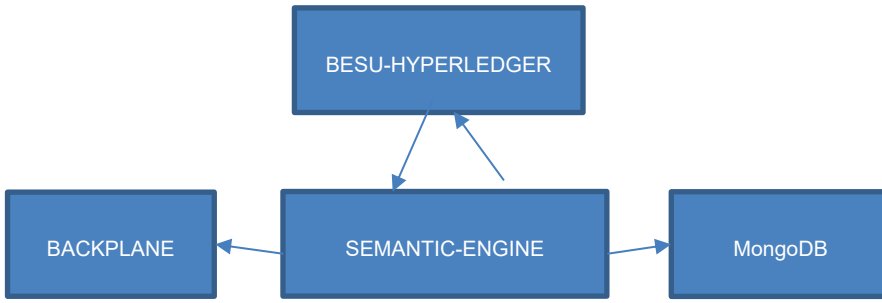


Figure 9.2 High-level Backplane block diagram.

9.4 Building Block High-level Picture

The specific component diagrams are shown in Figures 9.3–9.6.

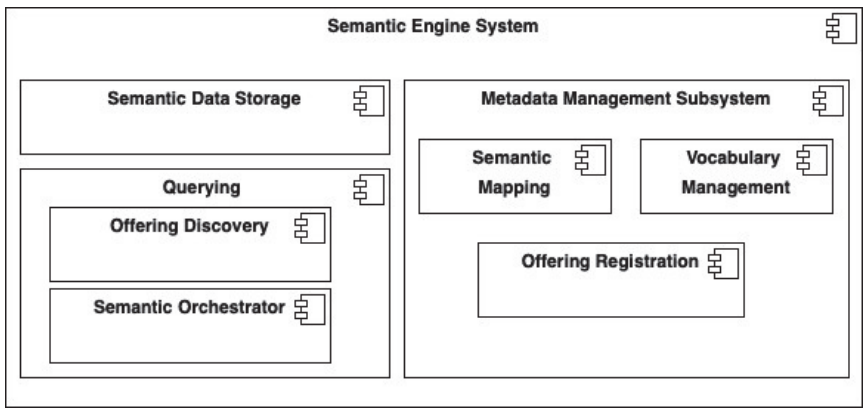


Figure 9.3 High-level operations of the semantic engine system.

For the semantic subsystems in charge of dealing with "semantic data management", we can highlight the following parts:

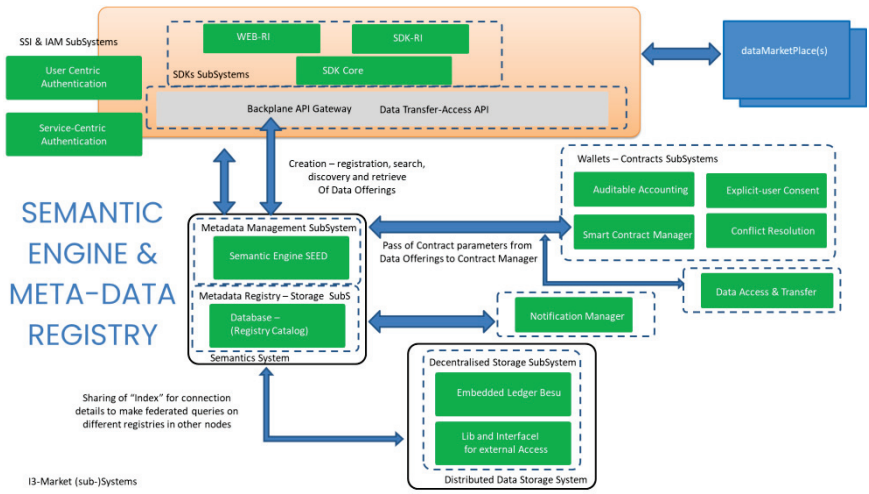


Figure 9.4 Main interfaces and interactions of the semantic engine system.

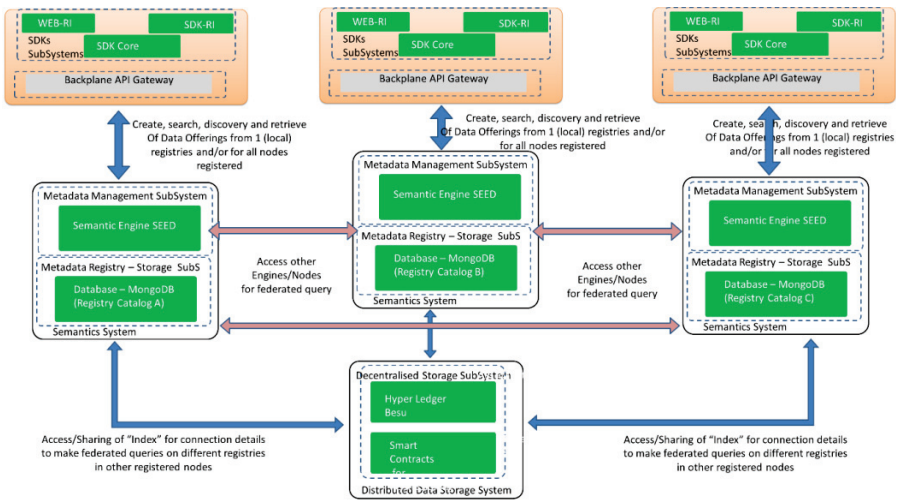


Figure 9.5 Main operations and interactions for the federated functionalities of the semantic engine system.

– Semantic data storage: This component on receiving JSON pushes the data to MongoDb database. MongoDb is a NoSql document-based database.

- Semantic mapping: This component does semantic mappings and transforms data received from API endpoints.
- Vocabulary management: This environment keeps and manages all of the vocabularies, defined as i3-MARKET semantic model, used in different operations of the semantic engine. The i3-MARKET Semantic Model is available using the GitHub and Gitlab repositories where the models/files are stored, shared, managed, and described, and the documentations is available in the developer portal.
- Offering registration: This component is basically REST APIs exposed as endpoints. Semantic engine exposes different endpoints for offering registration. Examples are:
    - register data provider;
    - register data offering of a data provider;
    - update data offerings;
    - deleting a data offering;
    - query existing offerings, etc.
- Offering discovery: This component is basically REST APIs exposed as endpoints. Semantic engine exposes different endpoints for offering discoveries and retrieving. Examples are:
    - retrieve a list of data offerings;
    - discover data offerings by providers;
    - discover data offerings by parameters;
    - discover data offerings by category;
    - discover data offerings by active state;
    - discover data offerings by shared state;
    - discover data offerings by text;
    - discover data offerings by keywords/text;
    - discover data offerings in federated search by category;
    - discover data offerings in federated search by active state;
    - discover data offerings in federated search by shared state;
    - discover data offerings in federated search by text;
    - discover data offerings in federated search by category;
    - search for particular metadata, etc.

Figure 9.6 shows a detailed landscape of the current set of microservices (cubes), API's (little yellow rectangles), components (blue rectangles), and storages (white rectangles) on i3-MARKET.

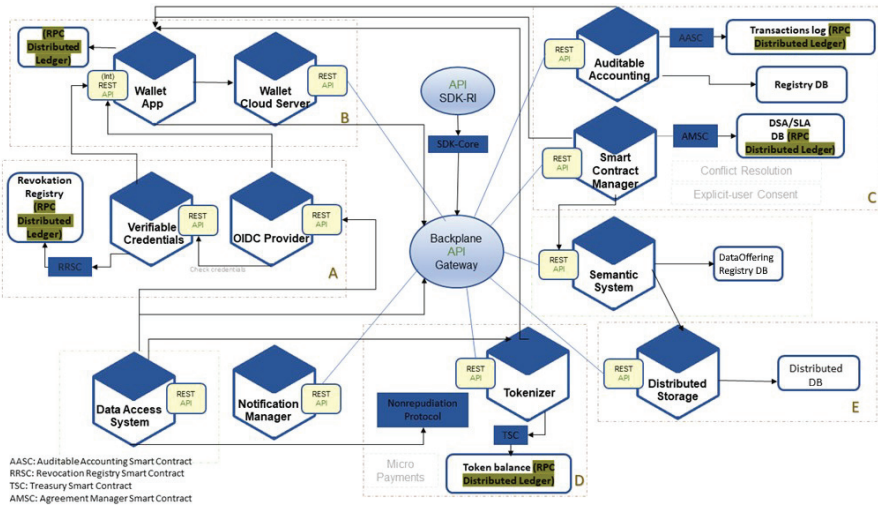


Figure 9.6 i3-MARKET services layout.

9.5 Diagrams

Data offering registration:

The diagram in Figure 9.7 shows that a data provider first must have to authenticate with i3-MARKET Backplane through a gateway. Once a provider is successfully authenticated, the provider can see all the APIs exposed by the semantic engine – called (SEED) – in the Backplane swagger interface. A provider can register an offering using registration endpoint using the template for data offering description. The Backplane internally communicates with SEED and dispatches create request to it. The engine, on receiving requests from Backplane, maps the incoming data into RDF according to the semantic data model and stores data into local registry catalogue database and sends back the response to the Backplane that offering is registered. The Backplane notifies the client/provider that offering has been successfully registered as represented in Figure 9.7.

Data offering discovery/deletion/update sequence diagram:

When a data provider interacts with i3-MARKET Backplane and has successful been authenticated, s/he can perform the following tasks:

- retrieve offering by providing offering ID;

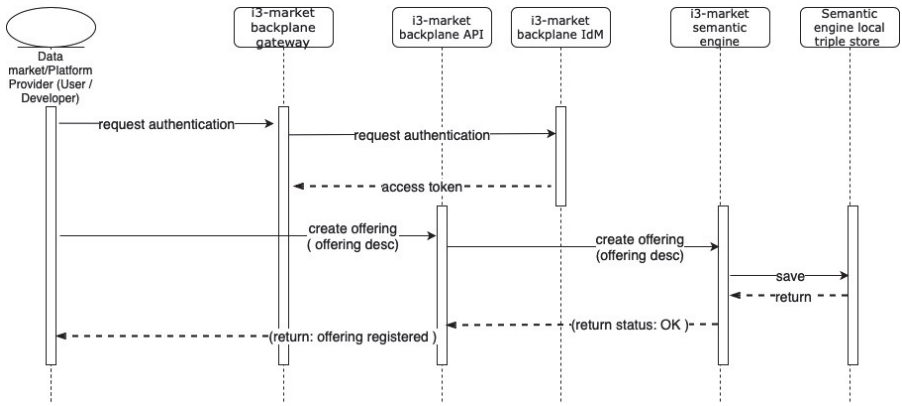


Figure 9.7 Sequence diagram for registering a data provider.

- retrieve a list of all offerings registered by a data provider using its provider ID;
- retrieve a list of all offerings filtered by category, which are registered not only in local instance of SEED (semantic engine) but also other instances of SEEDs running in the i3-Market cluster;
- update an offering;
- delete a particular data offering by providing its ID;
- the user can also download the data offering template.

The figure shows the sequence of messages used to perform different tasks. For instance, when a user wants to retrieve a particular offering s/he provides the offering ID and the Backplane sends this offering ID towards the SEED. On receiving an offering ID, the SEED executes a query on MongoDB. If the offering with the given ID is registered in the local storage, the SEED constructs the results (i.e., the requested offering) and sends back towards Backplane, where results are presented to the user. Similarly, if the user is interested to find all the offering registered by a data provider, s/he provides the provider ID and SEED looks all the offerings registered in local repository and sends back the results towards Backplane where results (list of all offerings registered by a data provider) are presented to users.

The SEED, by interacting with BESU, also can distribute the queries towards all other instances of SEEDs running on the i3-MARKET cluster. For example, if a user is interested to find offerings not only locally but also those that are registered on other instances in the i3-MARKET cluster. The

SEED engine transparently finds the offerings, filtered by category, from all the i3-MARKET instances.

Figure 9.8 shows that user can also update an offering by giving the description of an offering s/he queried by either offering by ID or data Provider ID. User has to copy the retrieved offering to the endpoint, where s/he can update any field. On receiving the updated offering, the SEED updates all the data against that particular data offering in the local storage. A data provider can also delete an offering by giving offering ID in the endpoint. Upon receiving the delete request from Backplane, the SEED executes a delete query on the local storage and the particular offering is permanently deleted from it.

To retrieve a query template, the user has to use the endpoint shown in Figure 9.8. On receiving template request, the SEED generates the offering template fully compliant with the semantic data model.

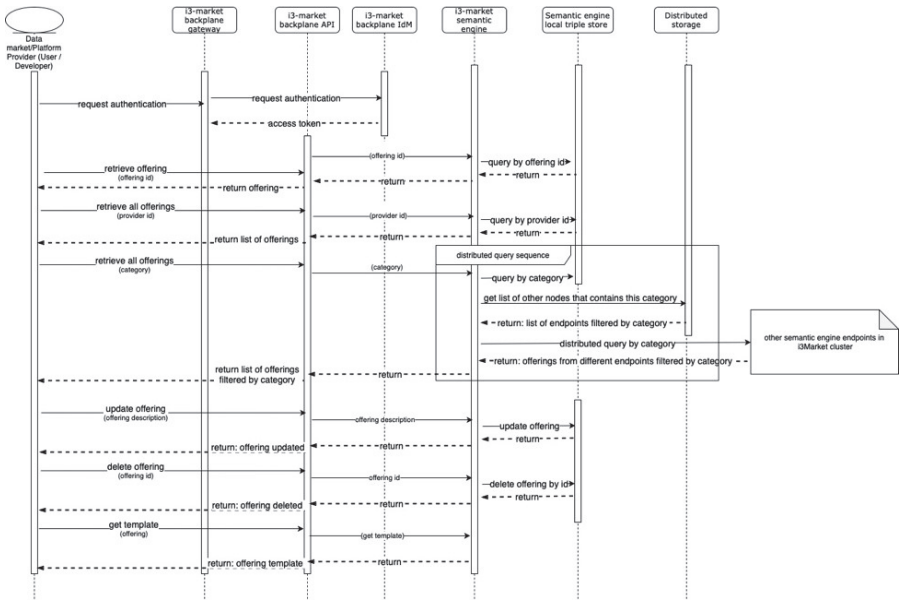


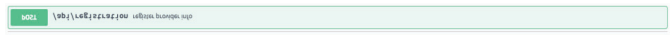
Figure 9.8 Sequence diagram for querying, deleting, and updating data offerings.

9.6 Interfaces

The semantic engine currently has many functionalities via APIs, which include: registration, searching, retrieving, updating, and deletion of different data offerings and delivery of info to other components.

Register data provider:

A data provider, when for the first-time interacts with the system, can register its information in the i3-MARKET. Following is the endpoint address and the request for registration of data provider in the semantic repository. The user must provide a "providerID" field, the ID which was provided to user at the authentication process.

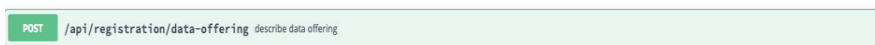


```
{
  "providerId": "string",
  "name": "string",
  "description": "string",
  "organization": [
    {
      "organizationId": "string",
      "name": "string",
      "description": "string",
      "address": "string",
      "contactPoint": "string"
    }
  ]
}
```

Listing 9.1 Data provider template.

Register data offering:

When a user is registered as data provider, the next step would be to create and register data offerings (semantic descriptions) for the data assets that they want to share/sell. Below is the API pointer and the request template the user can use to register data offerings. It is important to note that the value of "provider" in the offering template should be the "ID" of the data provider.




```
{
  "context": {
    "core": "http://i3-MARKET.eu/Backplane/core/",
    "dcat": "http://www.w3.org/ns/dcat#",
    "pricingModel": "http://i3-MARKET.eu/Backplane/pricingmodel"
  },
  "dataOfferingId": "string",
```

```
  "provider": "string",
  "marketId": "string",
  "owner": "string",
  "providerDid": "string",
  "marketDid": "string",
  "ownerDid": "string",
  "active": true,
  "ownerConsentForm": "string",
  "inSharedNetwork": true,
  "personalData": true,
  "dataOfferingTitle": "string",
  "dataOfferingDescription": "string",
  "category": "string",
  "status": "string",
  "dataOfferingExpirationTime": "string",
  "version": 0,
  "createdAt": "2022-12-19T15:20:56.816Z",
  "updatedAt": "2022-12-19T15:20:56.816Z",
  "contractParameters": {
    "interestOfProvider": "string",
    "interestDescription": "string",
    "hasGoverningJurisdiction": "string",
    "purpose": "string",
    "purposeDescription": "string",
    "hasIntendedUse": {
      "processData": true,
      "shareDataWithThirdParty": true,
      "editData": true
    },
    "hasLicenseGrant": {
      "transferable": true,
      "exclusiveness": true,
      "paidUp": true,
      "revocable": true,
      "processing": true,
      "modifying": true,
      "analyzing": true,
      "storingData": true,
      "storingCopy": true,
      "reproducing": true,
      "distributing": true,
      "loaning": true,
      "selling": true,
      "renting": true,
      "furtherLicensing": true,
      "leasing": true
    }
  },
  "hasPricingModel": {
    "pricingModelName": "string",
    "basicPrice": 0,
    "currency": "string",
    "hasPaymentOnSubscription": {
      "paymentOnSubscriptionName": "string",
      "paymentType": "string",
      "timeDuration": "string",
      "description": "string",
      "repeat": "string",
      "hasSubscriptionPrice": 0
    },
    "hasPaymentOnApi": {
      "paymentOnApiName": "string",
      "description": "string",
      "numberOfObject": 0,
      "hasApiPrice": 0
    },
```

```
    "hasPaymentOnUnit": {
      "paymentOnUnitName": "string",
      "description": "string",
      "dataUnit": 0,
      "hasUnitPrice": 0
    },
    "hasPaymentOnSize": {
      "paymentOnSizeName": "string",
      "description": "string",
      "dataSize": "string",
      "hasSizePrice": 0
    },
    "hasFreePrice": {
      "hasPriceFree": true
    }
  },
  "hasDataset": {
    "title": "string",
    "keyword": "string",
    "dataset": "string",
    "description": "string",
    "issued": "string",
    "modified": "string",
    "temporal": "string",
    "language": "string",
    "spatial": "string",
    "accrualPeriodicity": "string",
    "temporalResolution": "string",
    "theme": [
      "string"
    ],
    "distribution": [
      {
        "title": "string",
        "description": "string",
        "license": "string",
        "accessRights": "string",
        "downloadType": "string",
        "conformsTo": "string",
        "mediaType": "string",
        "packageFormat": "string",
        "dataStream": true,
        "accessService": {
          "conformsTo": "string",
          "endpointDescription": "string",
          "endpointURL": "string",
          "servesDataset": "string",
          "serviceSpecs": "string"
        },
        "dataExchangeSpec": {
          "encAlg": "string",
          "signingAlg": "string",
          "hashAlg": "string",
          "ledgerContractAddress": "string",
          "ledgerSignerAddress": "string",
          "pooToPorDelay": 0,
          "pooToPopDelay": 0,
          "pooToSecretDelay": 0
        }
      }
    ],
    "datasetInformation": [
      {
        "measurementType": "string",
        "measurementChannelType": "string",
        "sensorId": "string",
```

```
        "deviceId": "string",
        "cppType": "string",
        "sensorType": "string"
      }
    ]
  }
}
```

Listing 9.2 Data offering template.

(When the data offering template is created, the system can use the above JSON request and store it, but the system can be updated in case to manage the data offerings as Json-ld in the registry storage.)

```
{
"@context": {
    "core": "http://i3-MARKET.eu/Backplane/core/"
    "dcat": "https://www.w3.org/ns/dcat.jsonld"
    "pricingmodel": "http://i3-MARKET.eu/Backplane/pricingmodel"
  },
  "id": "#####-#######-#######-###" OR "http://i3-MARKET.org/resource/#####-#######-#######-
###"
  "type": "http://i3-MARKET.eu/Backplane/core/DataOffering"

  "provider": "#####-#######-#######-###"
  "marketId": "#####-#######-#######-###",
  "owner": "#####-#######-#######-###",
  "dataOfferingTitle": "_field",
  "dataOfferingDescription": "string",
  "category": "Other",
  "status": "e.g. Activated, InActivated, ToBeDeleted, Deleted",
  "dataOfferingExpirationTime": "NA",
  "contractParameters":
    {
      "id": "http://i3-MARKET.org/resource/#####-#######-#######-###"
      "type": "http://i3-MARKET.eu/Backplane/core/ContractParameters"

      "contractParametersId": "string",
      "interestOfProvider": "NA",
      "interestDescription": "NA",
      "hasGoverningJurisdiction": "NA",
      "purpose": "NA",
      "purposeDescription": "NA",
      "hasIntendedUse":
        {
          "id": "http://i3-MARKET.org/resource/#####-#######-#######-###"
          "type": "http://i3-MARKET.eu/Backplane/core/IntendedUse"
```

```
        "intendedUseId": "string",
        "processData": "true OR false",
        "shareDataWithThirdParty": "true OR false",
        "editData": "true OR false"
      }      ,
    "hasLicenseGrant":
      {
        "id": "http://i3-MARKET.org/resource/#####-#######-#######-###"
        "type": "http://i3-MARKET.eu/Backplane/core/LicenseGrant"

        "licenseGrantId": "string",
        "copyData": "true OR false",
        "transferable": "true OR false",
        "exclusiveness": "true OR false",
        "revocable": "true OR false"
      }
  }  ,


"hasDataset":
  {
    "id": "http://i3-MARKET.org/resource/#####-#######-#######-###"
    "type": "http://www.w3.org/ns/dcat#Dataset"

    "datasetId": "string",
    "title": "_field",
    "keyword": "_field",
    "dataset": "_field",
    "description": "_field",
    "issued": "date-time",
    "modified": "date-time",
    "temporal": "_field",
    "language": "_field",
    "spatial": "_field",
    "accrualPeriodicity": "_field",
    "temporalResolution": "_field",
    "distribution": [
      {
        "id": "http://i3-MARKET.org/resource/#####-#######-#######-###"
        "type": "http://www.w3.org/ns/dcat#Distribution"

        "distributionId": "string",
        "title": "_field",
```

```
        "description": "_field",
        "license": "_field",
        "accessRights": "_field",
        "downloadType": "_field",
        "conformsTo": "_field",
        "mediaType": "_field",
        "packageFormat": "_field",
        "accessService":
          {
            "id": "http://i3-MARKET.org/resource/#####-#######-#######-###"
            "type": "http://www.w3.org/ns/dcat#DataService"

            "dataserviceId": "string",
            "conformsTo": "_field",
            "endpointDescription": "_field",
            "endpointURL": "_field",
            "servesDataset": "_field",
            "serviceSpecs": "_field"
          }
      }
    ],
    "datasetInformation": [
      {
        "id": "http://i3-MARKET.org/resource/#####-#######-#######-###"
        "type": "http://i3-MARKET.eu/Backplane/core/DatasetInformation"

        "datasetInformationId": "string",
        "measurementType": "_field",
        "measurementChannelType": "_field",
        "sensorId": "_field",
        "deviceId": "_field",
        "cppType": "_field",
        "sensorType": "_field"
      }
    ],
    "theme": [
      "_field"
      "_field"
      "_field"
    ]
  }
}
```

Query a registered data offering by offering ID:

Figure 9.9 shows the endpoint to fetch a particular offering registered in store. A data provider must provide the "offering ID". This offering can further be used in other endpoint (i.e., /semantic-engine/api/registration/update-offering) if the user wants to update this offering.

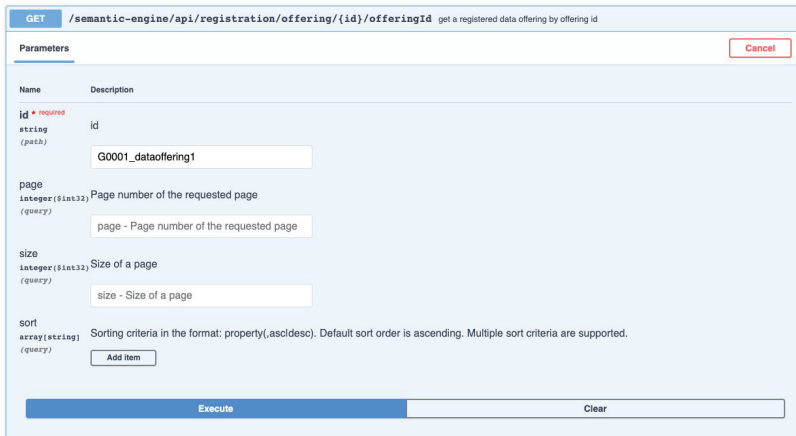


Figure 9.9 Get offering by offering ID.

Query a list of all registered data offerings by provider ID:

The following endpoint is used to fetch all the offerings registered by a data provider – see Figure 9.10. In this endpoint, the user must provide the

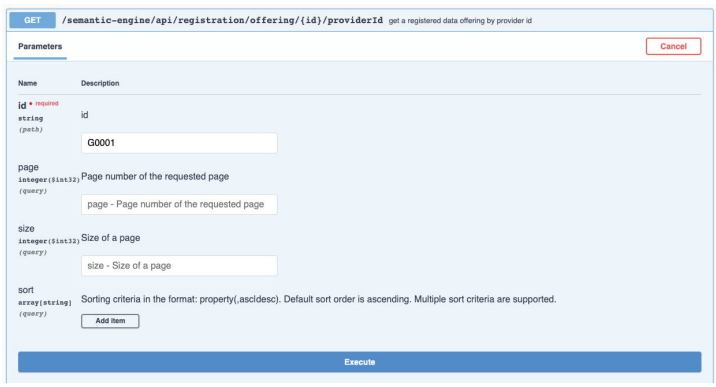


Figure 9.10 Get a list of offerings by provider ID.

"Provider ID". The current release includes retrieval of list of offerings with complete data offering. This might affect the query performance if the data in the storage is increased.

Query a list of all registered data offerings by category:

In the i3-MARKET project, we use different nodes in the cluster and each node has its own semantic engine instance running on it. Furthermore, each instance of semantic engine may have its own data by categories from different pilots (e.g., manufacturing, automotive, wellbeing, etc.) or multiple. Consider a use-case where someone is looking for data offerings registered in i3-MARKET on different nodes. This endpoint allows the user to transparently fetch all the data offerings based on the "category" from i3-MARKET cluster. In summary, this endpoint performs federated query in a distributed nature and brings back the results from different instances in i3-MARKET; see Figure 9.11.

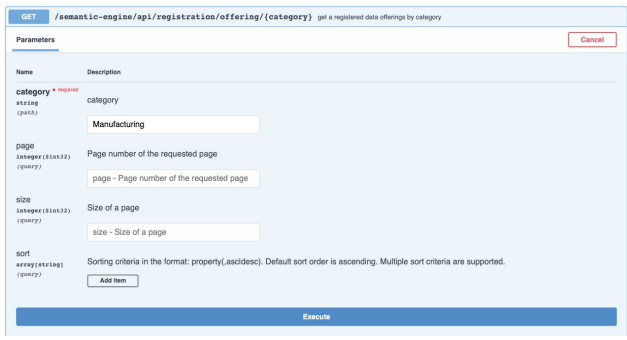
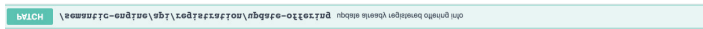


Figure 9.11 Get a list of offerings by category.

Update a data offering:

The following endpoint is used to update an already registered data offering.



For example, if a specific user can update any field which s/he wants to update, it is important that the user do not change/update the fields with -id attributes, e.g., dataOfferingId, pricingModelId, etc., because such attributes are used internally by the semantic engine to link the data.

Delete a data offering:

This endpoint can be used to permanently remove an offering from the repository. The user must provide the "Offering ID" of the data offerings they want to delete; see Figure 9.12.

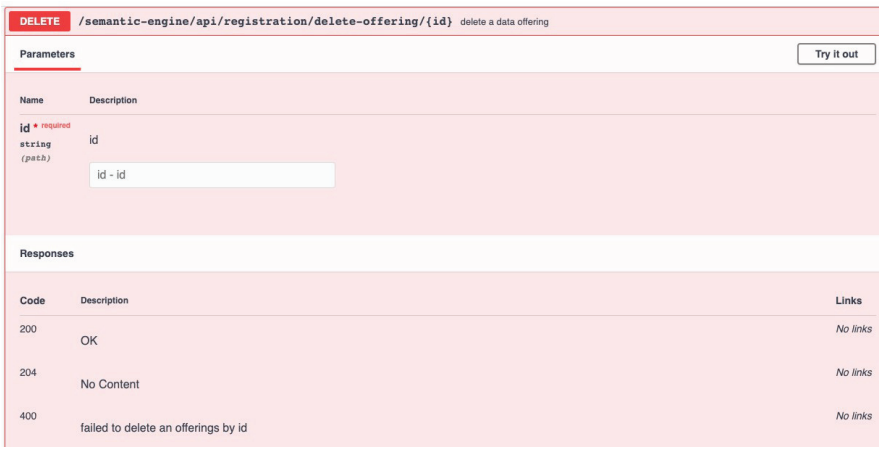


Figure 9.12 Delete offering by ID.

Download data offering template:

Figure 9.13 shows that endpoint is used to download the offering template.

Figure 9.13 Get data offering template.

Query list of offerings by active state:

Figure 9.14 shows an endpoint used to search data offerings that are "active" and so made available to be seen and searched by their providers.

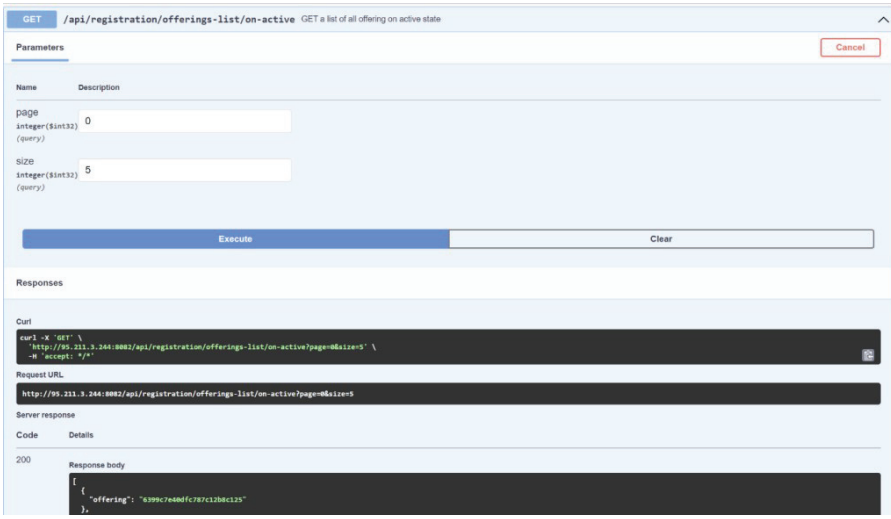


Figure 9.14 Query list of offerings by active state.

Query list of offerings by shared state:

Figure 9.15 shows the endpoint to look for data offerings that are set or not available to be shared in the network by the data marketplaces.

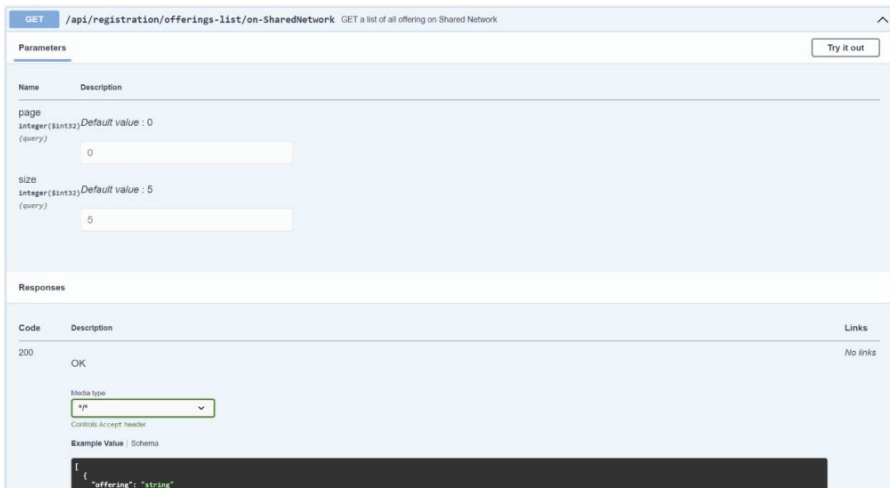


Figure 9.15 Query list of offerings by shared state.

Query offerings based on text/keyword:

Figure 9.16 shows an endpoint can be used for text searches.

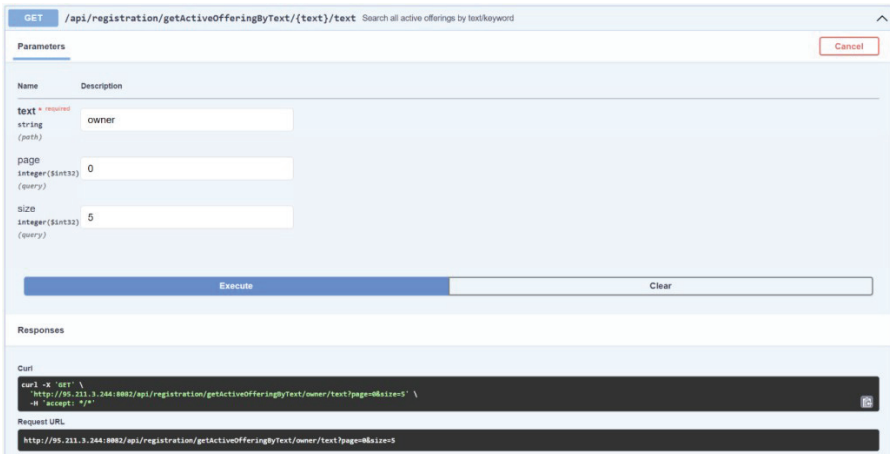
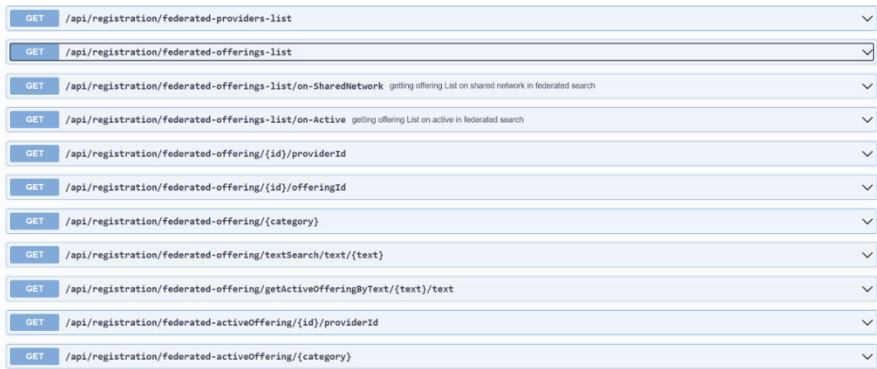


Figure 9.16 Query offerings based on text/keyword.

Query offerings in federated network:

The semantic engine is able to search, discover, and retrieve data offerings not only in single instance of a marketplace but also throughout the entire nodes of marketplace belonging to the i3-MARKET network via federated queries. This is possible using the information that each semantic engine manages via SEED-INDEX in the shared BESU blockchains, where there is info about each node/engine. With such details, each semantic engine can search and retrieve the "shared, active" data offerings from the other data marketplace to be consulted by consumers and expanding the availability of offered assets from one marketplace to the entire network.

Following are some of the endpoints:



To be noted, most of the endpoints work the same as in local node. The main difference is that now we can search from the cluster or network of registered marketplaces/endpoints.

SDK(-RI) semantic engine services:

Once the functionalities of the semantic engine from the internal API interfaces are mapped and reflected in the Backplane API gateway, they are available to be used via the i3-MARKET development kits in languages like Java and JavaScript (among the others) using directly the SDK-Core and/or the SDK-RI services.

9.7 Background Technologies

Data Catalogue Vocabulary (DCAT) — Version 3:

W3C (World Wide Web Consortium) recommendation:

DCAT is an RDF vocabulary designed to facilitate interoperability between data catalogues published on the Web. This document defines the schema and provides examples for its use.

DCAT enables a publisher to describe datasets and data services in a catalogue using a standard model and vocabulary that facilitates the consumption and aggregation of metadata from multiple catalogues. This can increase the discoverability of datasets and data services. It also makes it possible to have a decentralized approach to publishing data catalogues and makes federated

search for datasets across catalogues in multiple sites possible using the same query mechanism and structure.

https://www.w3.org/TR/vocab-dcat-3/

Also, its extension DCAT-AP:

The DCAT Application Profile for data portals in Europe (DCAT-AP) is a specification based on the Data Catalogue Vocabulary (DCAT) developed by W3C.

This application profile is a specification for metadata records to meet the specific application needs of *data portals in Europe* while providing semantic interoperability with other applications on the basis of reuse of established controlled vocabularies (e.g., EuroVoc) and mappings to existing metadata vocabularies (e.g., Dublin Core, SDMX, INSPIRE metadata, etc.).

DCAT-AP provides a common specification for describing public sector datasets in Europe to enable the exchange of descriptions of datasets among data portals. DCAT-AP allows:

- *Data catalogues* to describe their dataset collections using a standardized description, while keeping their own system for documenting and storing them.
- *Content aggregators*, such as the European Data Portal, to aggregate such descriptions into a single point of access.
- *Data consumers* to find datasets more easily through a single point of access.

DCAT-AP has an extension GeoDCAT-AP for describing geospatial datasets, dataset series and services. Another extension, StatDCAT-AP, provides specifications and tools that enhance interoperability between descriptions of statistical datasets within the statistical domain and between statistical data and open data portals.

https://joinup.ec.europa.eu/solution/dcat-application-profile-data-portals-europe/release/200

RDF Store

As part of the marketplace persistence framework back-end layer, we need to use and deploy a database that is able to store our semantic (meta)data in the best way. This database represents the main registry and repository where all the semantically annotated (meta)data are uploaded and saved.

In the persistent database, it is needed to store all the (meta)data descriptions created and collected by marketplace stakeholders, e.g., with the

information about providers, consumers, offering descriptions, and recipes. In our research for a semantic interoperability in i3-MARKET, we decided to model our providers, consumers, data offering descriptions, and parameters following an RDF schema model, annotated with our *i3-MARKET Semantic Core Model* and represent and exchange data in JSON serialization format.

So due to the nature of such kind of (meta)data, we need to choose the best solution for storing, managing, accessing, and retrieving information.

RDF triple-store is a type of graph database that stores semantic facts. Being a graph database, triple-store stores data as a network of objects with materialized links between them. This makes RDF triple-store a preferred choice for managing highly interconnected data. Triple-stores are more flexible and less costly than a relational database, for example.

The RDF database, often called a semantic graph database, is also capable of handling powerful semantic queries and of using inference for uncovering new information out of the existing relations. In contrast to other types of graph databases, RDF triple-store engines support the concurrent storage of data, metadata, and schema models (e.g., the so-called ontologies). Models/ontologies allow for the formal description of the data. They specify both object classes and relationship properties, and their hierarchical order as we use our i3-MARKET models to describe our resources.

This allows creating a unified knowledge base grounded in common semantic models that allow combining all metadata coming from different sources, making them semantically interoperable to:

- create coherent queries independently from the source, format, date, time, provider, etc.;
- enable the implementation of more efficient semantic querying features;
- enrich the data and make it more complete, more reliable, and more accessible;
- enable to perform inference as triple materialization from some of the relations.

In the following paragraphs, we are going to give some more information and examples about the semantic data formalization, query interface, and the interface of the semantic framework backend layer within the Backplane.

MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses

JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. (https://www.mongodb.com/).

Main features:

- Ad-hoc queries:

MongoDB supports field, range query, and regular-expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.

- Indexing:

Fields in a MongoDB document can be indexed with primary and secondary indices or index.

- Replication:

MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data. Each replica set member may act in the role of primary or secondary replica at any time. All writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can optionally serve read operations, but that data is only eventually consistent by default.

- Load balancing:

MongoDB scales horizontally using sharding. The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more replicas.) Alternatively, the shard key can be hashed to map to a shard – enabling an even data distribution.

Semantic data model and serialization formats:

Linked data is based around describing real-world things using the resource description framework (RDF). The following paragraphs introduce the basic data model and then outline existing formats to serialize semantic data models.

The semantic descriptions are generated following the *i3-MARKET Core Model*, annotated with the *i3-MARKET Domain Models*, and mapped with the *i3-MARKET Application Model* vocabularies and then loaded into a registry-store.

Semantic data model:

Figure 9.17 represents an RDF triple. RDF is very simple, flexible, and schema-less to express and process a series of simple assertions. Consider the following example: "Sensor A measures 21C". Each statement, i.e., piece of information, is represented in the form of *triples* (RDF triples) that link a *subject* ("Sensor A"), a *predicate* ("measures"), and an *object* ("21C"). The subject is the thing that is described, i.e., the resource in question. The predicate is a term used to describe or modify some aspect of the subject. It is used to denote relationships between the subject and the object. The object is, in RDF, the "target" or "value" of the triple. It can be another resource or just a literal value such as a number or word.

Figure 9.17 RDF triple in graph representation describing "Sensor A measures 21.8°C".

Since objects can also be a resource with predicates and objects on their own, single triples are connected to a so-called RDF graph. In terms of graph theory, the RDF graph is a labelled and directed graph. As the illustration, we extend the previous example, replacing the literal "21.8C" by a resource "measurement" for the object in the RDF triple in Figure 9.18. The resource itself has two predicates assigning a unit and the actual value to the

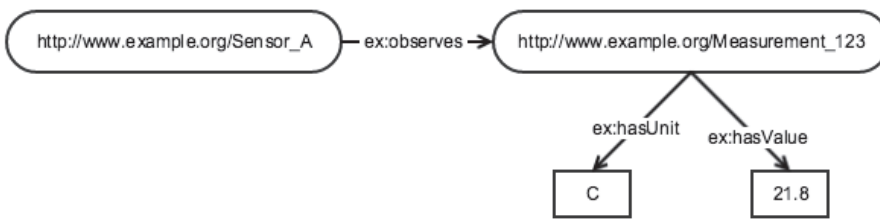


Figure 9.18 Simple RDF graph including the example RDF triple.

measurement – see Figure 9.18. The unit is again represented by a resource and the value is numerical literal. The resulting RDF graph looks as follows:

Serialization formats:

The RDF data model itself does not describe the format in which the data, i.e., the RDF graph structure, is stored, processed, or transferred. Several formats exist that serialize RDF data; the following overview lists the most popular formats, including a short description of their main characteristics and examples. Figure 9.18 shows a simple RDF graph to serve as the basis.

RDF/XML:

The RDF/XML syntax is standardized by the W3C and is widely used to publish linked data on the Web. On the downside, however, the XML syntax is also viewed as difficult for humans to read and write. This recommends consideration of:

a) other serialization formats in data management and control workflows that involve human intervention;
b) the provision of alternative serializations for consumers who may wish to examine the raw RDF data.

The RDF/XML syntax is described in detail as part of the W3C RDF Primer. The MIME type that should be used for RDF/XML within HTTP content negotiation is application/rdf+xml. The listing below shows the RDF/XML serialization for the RDF graph.

RDF/XML serialization example:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:ex="http://www.example.org/"
<rdf:Description rdf:about=" http://www.example.org/Sensor_A">
        <ex:title>21.8°C</ex:title>
</rdf:Description>
</rdf:RDF>
```

Turtle: Turtle (Terse RDF Triple Language) is a plain text format for serializing RDF data. It has support for namespace prefixes and other shorthands, making Turtle typically the serialization format of choice for reading RDF

triples or writing them by hand. A detailed introduction to Turtle is given in the W3C Team Submission document Turtle. It was accepted as a first working draft by the World Wide Web Consortium (W3C) RDF Working Group in August 2011, and parsing and serializing RDF data is supported by a large number of RDF toolkits. The following listing shows the serialization listing for the example RDF graph in Turtle syntax.

Turtle serialization example:

```
@prefix : <http://www.example.org/> .
:Sensor_A :measures "21.8°C"
```

N-Triples: The N-Triples syntax is a subset of Turtle, excluding features such as namespace prefixes and shorthands. Since all URIs must be specified in full in each triple, this serialization format involves a lot of redundancy, typically resulting in large N-Triples particularly compared to Turtle, but also to RDF/XML. This redundancy, however, enables N-Triples files to be parsed one line at a time, benefitting the loading and processing of large data files that will not fit into main memory. The redundancy also allows compressing N-Triples files with a high compression ratio, thus reducing network traffic when exchanging files. These two factors make N-Triples the *de facto* standard for exchanging large dumps of linked data. The complete definition of the N-Triples syntax is given as part of the W3C RDF test cases recommendation. The following listing in Table 7.1 represents the N-Triples serialization of the example RDF graph.

N-Triples serialization example:

```
<http://www.example.org/Sensor_A> <http://www.example.org/measures> "21.8°C"@en-UK
.
```

JSON-LD: Used as main data model for the metadata in i3-MARKET.

Jason-LD (https://json-ld.org/) – Many developers have little or no experience with linked data, RDF, or common RDF serialization formats such as N-Triples and Turtle. This produces extra overhead in the form of a steeper learning curve when integrating new systems to consume linked data. To counter this, the project consortium decided to use a format based on a common serialization format such as XML or JSON. Thus, the two remaining options are RDF/XML and JSON-LD. JSON-LD was chosen over

RDF/XML as the data format for all linked data items in BigIoT. JSON-LD is a JSON-based serialization for linked data with the following design goals:

- **Simplicity:** There is no need for extra processors or software libraries, just the knowledge of some basic keywords.
- **Compatibility:** JSON-LD documents are always valid JSON documents; so the standard libraries from JSON can be used.
- **Expressiveness:** Real-world data models can be expressed because the syntax serializes a directed graph.
- **Terseness:** The syntax is readable for humans and developers need little effort to use it.
- **Zero edits:** Most of the time JSON-LD can be devolved easily from JSON-based systems.
- **Usable as RDF:** JSON-LD can be mapped to/from RDF and can be used as RDF without having any knowledge about RDF.

From the above, terseness and simplicity are the main reasons that JSON-LD was chosen over RDF/XML. JSON-LD also allows for referencing external files to provide context. This means contextual information can be requested on demand and makes JSON-LD better suited to situations with high response times or low bandwidth usage requirements. More information can be found in http://json-ld.org/.

The data model underlying JSON-LD is a labelled, directed graph. There are a few important keywords, such as @context, @id, @value, and @type. These keywords are the core part of JSON-LD. Four basic concepts should be considered:

- **Context:** A context in JSON-LD allows using shortcut terms to make the JSON-LD file shorter and easier to read (as well as increasing its resemblance with pure JSON). The context maps terms to IRIs. A context can also be externalized and reused for multiple JSON-LD files by referencing its URI.
- **IRIs:** Internationalized resource identifiers (IRIs) are used to identify nodes and properties in linked data. In JSON-LD, two kinds of IRIs are used: absolute IRIs and relative IRIs. JSON-LD also allows defining a common prefix for relative IRIs using the keyword @vocab.
- **Node identifiers:** Node identifiers (using the keyword @id) reference nodes externally. As a result of using @id, any RDF triples produced for this node would use the given IRI as their subject. If an application follows this IRI, it should be able to find some more information about

the node. If no node identifier is specified, the RDF mapping will use blank nodes.

- **Specifying the type:** It is possible to specify the type of a distinct node with the keyword @type. When mapping to RDF, this creates a new triple with the node as the subject, a property rdf:type and the given type as the object (given as an IRI).

JSON-LD example:

```
[{"@id":"http://www.example.org/Sensor_A","http://www.example.org/measures":[{"@val
ue":"21.8C"}]}]
```

SPARQL:
SPARQL (SPARQL protocol and RDF query language, https://www.w3.org/TR/sparql11-query/) is the most popular query language to retrieve and manipulate data stored in RDF and became an official W3C recommendation in 2008. Depending on the purpose, SPARQL distinguishes the following for query variations:

- **SELECT query:** Extraction of (raw) information from the data.
- **CONSTRUCT query:** Extraction of information and transformation into RDF.
- **ASK query:** Extraction of information resulting a true/false answer.
- **DESCRIBE query:** Extraction of RDF graph that describes the resources found.

Given that RDF forms a directed, labelled graph for representing information, the most basic construct of a SPARQL query is a so-called *basic graph pattern*. Such a pattern is very similar to an RDF triple with the exception that the subject, predicate, or object may be a variable. A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph. Using the same identifier for variables also allow combining multiple graph patterns. Besides aforementioned graph patterns, the SPARQL 1.1 standard also supports the sorting (ORDER BY), and the limitation of result sets (LIMIT, OFFSET), the elimination of duplicates (DISTINCT), the formulation of conditions over the value of variables (FILTER), and the possibility to declare a constraint as OPTIONAL. The SPARQL 1.1 standard significantly extended the expressiveness of SPARQL. In more detail, the new features include:

- Grouping (GROUP BY) and conditions on groups (HAVING).
- Aggregates (CONT, SUM, MIN, MAX, AVG, etc.).
- Subqueries to embed SPARQL queries directly within other queries.
- Negation to, e.g., check for the absence of data triples.
- Project expression, e.g., to use numerical result values in the SELECT clause within mathematical formulas and assign new variable names to the result.
- Update statements to add, change, or delete statements.
- Variable assignments to bind expressions to variables in a graph pattern.
- New built-in functions and operators, including string functions (e.g., CONCAT, CONTAINS, etc.), string digest functions (e.g., MD5, SHA1, etc.), numeric functions (e.g., ABS, ROUND, etc.), or date/time functions (e.g., NOW, DAY, HOURS, etc.).

As mentioned previously, RDF graph data is represented as triples, i.e., "subject", "predicate", and "object". A very basic SPARQL, which brings back 100 triples from the RDF graph, can be written as follows.

SPARQL example:

```
SELECT * WHERE {?s ?p ?o} LIMIT 100
```