Horizon 2020 – The EU Framework Programme for Research and Innovation
Project Co-funded by the European Commission
Contract number: 761145
Call identifier: NMBP-22-2017
Project Start Date: 1st January 2018

MANUSQUARE

MANUfacturing eco**S**ystem of **QUA**lified **R**esources **E**xchange

D2.4
Semantic Infrastructure

Dissemination Level	Public
Partners	INNOVA
Authors	Marko Vujasinovic (INNOVA), Alessio Gugliotta (INNOVA)
Planned date of delivery	M24 – 31/12/2019
Date of issue	31/12/2019
Document version	V1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 761145

DOCUMENT HISTORY

Version	Issue date	Content and changes	Author
0.1	06 June 2019	ToC and Introduction, Sections structured.	INNOVA
0.2	11 June 2019	Overall overview of Restful API with usage examples. Restful API Documentation shared with Consortium.	INNOVA
0.3	18 Nov 2019	Executive summary added, introduction revised	INNOVA
0.4	28 Nov 2019	All sections drafted. Proofreading by authors	INNOVA
0.5	29 Nov 2019	Proofreading completed. Version for internal review	INNOVA
0.9	2 Dec 2019	Minor changes. Shared with internal reviewers	INNOVA
0.9.1	5 Dec 2019	1st internal review. Comments received and addressed	SINTEF, INNOVA
0.9.2	5 Dec 2019	2nd internal review. Comments received and addressed. Sent to quality assurance review.	HOLONIX, INNOVA
0.9.9	17.12.2019.	Quality assurance / Project Coordinator review	SUPSI
1.0	17.12.2019.	Final version	INNOVA

Role	Partner	Person
Reviewer 1	SINTEF	Audun Vennesland
Reviewer 2	HOLONIX	Serena Albertario
Quality assurance	SUPSI	Elias Montini

TABLE OF CONTENTS

Document history 2
Table of contents 3
List of figures 5
List of tables 5
List of examples/listings 5
List of abbreviations 6
1 Executive summary 7
2 Introduction 8
2.1 Aim and scope of Task 2.4 9
2.2 Semantic support in MANUSQUARE 10
2.3 Current status of Task 2.4 11
2.4 Document outline 11
3 RDF Technology 12
3.1 RDF 12
3.2 RDFS 12
3.3 OWL 13
3.4 SPARQL 14
3.5 SPIN (SPARQL Inference Notation) 14
3.6 RDF Triplestores 14
4 MANUSQUARE Semantic Infrastructure 15
4.1 Requirements Analysis 15
4.1.1 Requirements from Project-RFQ Creation tool perspective 15
4.1.2 Requirements from Challenge/Idea Creation tool perspective 16
4.1.3 Requirements from Matchmaking tool perspective 16
4.1.4 Requirements from Supplier (User) Profiling tool perspective 17
4.1.5 Requirements from Reputation Manager tool perspective 17
4.2 Design & Architecture 18
4.2.1 Core Layer 18
4.2.2 Ontology Authoring Layer 21
4.2.3 Services (RESTful API) Layer 21
4.3 Restful API Details 22
4.3.1 Access Control 22
4.3.2 Request Controllers 22
4.3.3 Data Controller 22
4.3.4 Ontology Controller 27

4.3.5 Query Requests Controller 31
4.3.6 Tag-Acquisition Controller 31
4.3.7 Inference Rule Controller 32
4.4 Creation of rules by templates and controlled natural language expressions 34
4.5 Rule management in Ontology Editor 35
5 Docker container deployment 36
6 Conclusion 37

LIST OF FIGURES

Figure 1: Composition of the unused potential 8
Figure 2: High level schema of MANUSQUARE architecture and WP2 role within it 9
Figure 3 RDF graph example 12
Figure 4 RDF + RDFS graph example 13
Figure 5 High-level architecture of MANUSQUARE Semantic Infrastructure 18
Figure 6 Graph of Supplier profile 19
Figure 7 RDFS subClassOf inference 20
Figure 8 Swagger Documentation Screen of API 21
Figure 9 Swagger: Data CRUD Requests Controller 23
Figure 10 Swagger POST /repository/manusquare/informal-to-semantic 30
Figure 11 Input to informal-to-semantic conversion 30
Figure 12 Tag-Acquisition Controller API 32
Figure 13 Rule Editor in Ontology Authoring tool 35

LIST OF TABLES

Table 1 Triplestores 14
Table 2 Requirements from Project/RFQ perspective 15
Table 3 Requirements from Idea Creation perspective 16
Table 4 Requirements from Matchmaking perspective 16
Table 5 Requirements from User Profile perspective 17
Table 6 Requirements from Reputation Manager perspective 17
Table 7 API Request controllers 22
Table 8 Ontology Controller GET operations 27
Table 9 Inference Rule Controller Operations 32

LIST OF EXAMPLES/LISTINGS

Listing 1 RDF description example 12
Listing 2 RDFS example 13
Listing 3 SPARQL Query example 14
Listing 4 RDF description of Supplier Profile 19
Listing 5 SPARQL query to search suppliers that produce more than 1000 pieces of CathodeTubes 20
Listing 6 SPIN inference rule example 20
Listing 7 Configuration of access control and security token 22
Listing 8 Supplier Profile in JSON-LD 24
Listing 9 Manufacturing Resource Description in JSON-LD 25
Listing 10 Project Request in JSON-LD format 26
Listing 11 Idea Description in JSON-ID (simple example) 27
Listing 12 Concept retrieval result 28
Listing 13 Property suggestion result 29
Listing 14 Property Value suggestion result 29
Listing 15 SPARQL example 31
Listing 16 SPARQL Geospatial example 31
Listing 17 Tag insertion payload 32
Listing 18 SPIN rule “servesIndustry” 33
Listing 19 Domain-specific Rule Templates 34
Listing 20 docker-compose.yml 36

LIST OF ABBREVIATIONS

RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
OWL	Ontology Web Language
SPIN	SPARQL Inferencing Notation
SPARQL	SPARQL Protocol and RDF Query Language
MODE	MANUSQUARE Ontology development methodology
JSON	JavaScript Object Notation
API	Application programming interface
REST	Representational state transfer
W3C	World Wide Web Consortium
UUID	Universally Unique Identifier
GUI	Graphical User Interface
URI	Uniform Resource Identifier
IRI	Internationalized Resource Identifier
MVP	Minimum Viable Product
CRUD	Create, read, update, delete
WP	Work Package

1 EXECUTIVE SUMMARY

This document presents results of the implementation and deployment of MANUSQUARE Semantic Infrastructure, developed in Task 2.4. The Semantic Infrastructure involves several subcomponents including a semantic repository (RDF triplestore database), RESTful API to interact with the Semantic Infrastructure in order to manage the ecosystem's semantic data and knowledge, and ontology editor, which has been developed in Task 2.2.

Semantic Infrastructure, at the first place, provides CRUD (create-read-update-delete) operations for the first-order citizens of the platform including **supplier profiles**, their **process-chains** and **process descriptions** (revealing production and engineering resources, their capability and capacity descriptions), **challenge/idea descriptions** (coming from innovation management toolset), and **project** (customer needs) **descriptions.** In addition to the CRUD capabilities, the Semantic Infrastructure provides multiple services that support: (1) **semantic inferencing** on assertions deployed by domain-specific ontologies and descriptions of manufacturing capabilities, to infer relationships between domain concepts and data, (2) **semantic querying** over the data stored in a semantic repository, (3) **inference rules managemen**t, (4) **semi-automatic support** to convert semi-structured descriptions of manufacturing resources into their semantic descriptions that are aligned with MANUSQUARE Core and domain-specific ontologies.

Internally, semantic Infrastructure has adopted RDF (Resource Description Framework) as a model for capturing semantically-relevant data managed by the infrastructure, while core- and domain-specific concepts and their relationships are captured using RDF Schema (RDFS). All the services of Semantic Infrastructure are exposed by standard RESTful API and documented with Swagger[1], with a JSON representation for HTTP request/response bodies.

The document starts from a brief overview of RDF(S) model and baseline technologies, then proceeds to describe an overall architecture of MANUSQUARE's Semantic Infrastructure. Then, it presents RESTful API with usage examples and explains how to run and deploy the infrastructure as a Docker container. Finally, the last section provides concluding remarks.

This written deliverable is accompanied with a working software component/service that, at Month 24, can be accessed in our development server[2] or run as Docker container (see Deployment section).

[1] https://swagger.io/

[2]Links: Swagger documentation http://116.203.187.118/semantic-registry/swagger-ui.html and MODE http://116.203.187.118/mode

2 INTRODUCTION

The MANUSQUARE project creates an novel ecosystem that acts as a digital marketplace bringing the available production capability and unused capacity, as well as other tangible and intangible production assets including know-how, technology, and by-products (waste), closer to the production demand to obtain the higher rate of matches between the supply and demand (between manufacturers and customers), as shown in Figure 1.

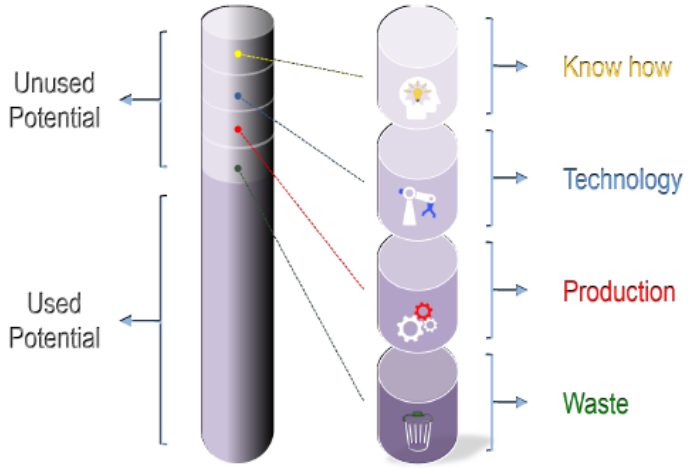


Figure 1: Composition of the unused potential

The MANUSQUARE ecosystem, based on the enhanced matchmaking between production supply and demand, provides two main advantages. First, the reintroduction and optimization in the loop of unused production capacity and production/business potential that would otherwise be lost. Second, the rapid and efficient creation of local and distributed value networks for innovative providers of product services.

In short, a manufacturing company that uses MANUSQUARE may have a role of supplier (seller) and/or customer (buyer). In the case of the former, the company uses MANUSQUARE to create its respective supplier profile that advertises what production capabilities, capacities, know-how, products, and by-products supplier may offer. In the case of the latter, the company uses the platform to engage with the MANUSQUARE ecosystem to fulfil a manufacturing-related need, such as to find a certain production capability. The platform, using its internal services, looks for the optimal matching on a wide number of possible suppliers from the MANUSQUARE register by employing the **semantic search and matchmaking on semantic descriptions in the platform**, and additionally by multifaceted criteria evaluation that may include quality and reliability aspects of suppliers, the best-fit costs and time for a business transaction. Interestingly, MANUSQUARE may enable discovery also for resources other than production capability, e.g. available production hours or tangible assets, with the aim to identify and exploit unexpected synergies between participants, not only within the same industry, but also within diverse industries, different value networks, for beneficial exploitation of all the competences.

Therefore, the MANUSQUARE project and technical platform aim to: (1) make European unused manufacturing capacity emerge towards its reintegration in the loop and the creation of local efficient value networks, (2) support innovative SMEs and start-ups in finding the optimal suppliers to transform their business ideas into new product-services, (3) seamlessly involve actors all along the entire value network including consumers for cross-fertilization of product-service solutions and underlying technologies, and (4) coordinate the whole ecosystem towards a better use of resources and a more sustainable European manufacturing.

To achieve the objectives, MANUSQUARE offers a set of integrated advanced services including a customer request vs. supplier capabilities matchmaking, by-products matching and reuse, users reputation management, idea management, sustainability assessment, and request-for-quantitation management.

Underneath these services, the platform has a block-chain technology as a secured ledger for supply chain transactions, and, as said, has a **semantic infrastructure to establish formal and unambiguous description of** MANUSQUARE **resources** and **to enable** resource **discovery and knowledge sharing** based on their formalized semantics.

2.1 Aim and scope of Task 2.4

Task 2.4 is one of the tasks of MANUSQUARE's Work Package 2 (Ecosystem ontological representation). WP2 develops and provides: (1) a semantic infrastructure (as highlighted in figure 2); (2) tool-supported development of ontological representation of manufacturing concepts and their taxonomies, (3) application-specific models and inference rules management.

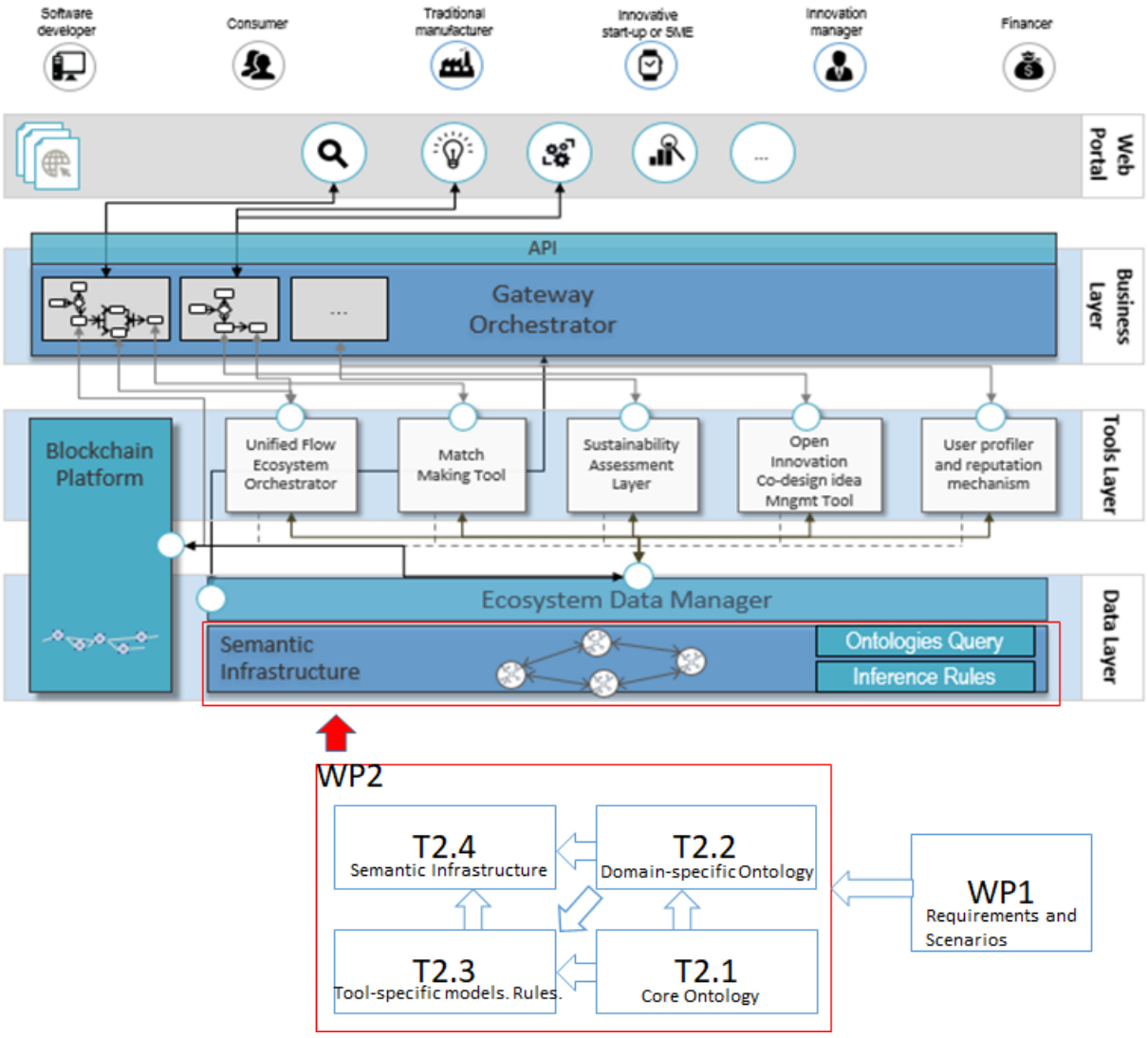


Figure 2: High level schema of MANUSQUARE architecture and WP2 role within it

Semantic Infrastructure of Task 2.4 serves as integration point of all technical elements from WP2. It develops, integrates and maintains all the technical deliverables from other tasks in WP2.

To remind, Task 2.1 provided a MANUSQUARE Core Ontology (Ecosystem Core Data model as reported in Deliverable D2.1.) The Core Ontology, or Code Concepts model, captures all core and generic concepts that are shared, within MANUSQUARE ecosystem, across multiple manufacturing sectors and applications, and that represent five main elements: (1) manufacturing capability and capacity, (2) innovation idea, (3) sustainability assessment, (4) request for quotation, and (5) user reputation. The Core Ontology, thus, forms a common upper vocabulary for descriptions of core manufacturing concepts - Process, Process Type, Equipment, Equipment Type, Item, Component, Capability, Supplier, and others. Task 2.3 further enhanced the core model of Task 2.1 with models containing the propositions required for

supporting the functioning of the service-providing applications, which are being developed in WP4. Semantic Infrastructure has deployed the latest version of MANUSQUARE Core Ontology into a semantic database and uses core concepts to classify data and core relationships to establish links between data.

Then, Task 2.2 provided a vertical extension of MANUSQUARE Core Ontology into sector-specific ontologies, by capturing sector-specific taxonomies of process types, equipment types, capability types, product types, material types, certification types, and flexible attribute types. For example, it captures a taxonomy of equipment types in the machining sector (e.g. Milling Machine, Drilling Machine, etc.). The conceptual scope of the domain-specific ontologies is narrowed to the scope of two demonstration scenario: (1) machining sector scenario of new product development use-case; (2) textile and cosmetics sector scenario of by-product usage for development of new product. In addition to the development of these two sector-specific ontologies, Task 2.2 provided a Web-application based ontology authoring tool to facilitate further development, expansion and evolution of sector-specific ontologies.

In addition to all that**,** Task 2.3 has defined types of useful inference rules that can allow to find and to infer additional facts and relationships by reasoning with the rules on the data present in the Semantic Infrastructure.

And finally, the Semantic infrastructure (Task 2.4), which is described in this deliverable, deploys and integrates all the semantic artefacts so far published by the project and on top of them builds the following capabilities: creation-updating-deletion-read operations for supplier profiles, production resource and capabilities descriptions, idea semantic tagging, project descriptions, querying over ontologies, inferences based on domain-specific rules, semantic query support for semantic matchmaking and discovery of MANUSQUARE resources. The Semantic Infrastructure, thus, stores, describes and maintains the knowledge that is necessary to the other platform tools to implement added-valued, automatic processes.

2.2 **Semantic support in MANUSQUARE**

There are many advantages of a semantic-based approach. Ontologies and semantics give unambiguous meaning to the concepts and data. Ontology as a shared and machine-interpretable definition of classes in the domain and relations among them, together with a set of individual instances of classes constitutes a knowledge base. Reasoning and inference engines use semantics of the ontological language constructs that capture concepts, their instances and their relationships, in order to classify instances, infer non-asserted relationships, detect inconsistent statements and to validate conceptual models. Using semantic descriptions of resources and inference rules, it is possible to computationally infer uncomplete, missing facts or new facts about the resources. Domain-specific inference rules can be expressed declaratively using a rule language and domain-specific vocabulary, rather than procedurally, which gives a great advantage over the traditional approaches.

A matchmaking of MANUSQUARE resources is the service (see D4.1) that largely exploits the domain-specific ontologies and captured semantics. The matchmaking compares the data/parameters that describe the needed resources, as entered by a customer, with description of available resources, as entered by suppliers, to detect and rank most suitable suppliers for customer needs.

The Matchmaking algorithm's precision and recall depends on an overall approach taken, wheatear it is textual search-based or semantics-based. Traditional manufacturing sourcing platforms[3] deploy a text-based search algorithms that perform literal matches of the search words or variants of them with words contained in the textual descriptions of production capabilities or capacities. A text-based search on informal textual descriptions is a very limited approach if the users want results that precisely semantically match the search request. Then, inference rule creation and execution over textual descriptions is almost impossible task as textual descriptions are very much unstructured and usually rely on completely different vocabularies.

[3] Examples of traditional platforms considered here are online manufacturing sourcing platforms, such as ThomasNet, GlobalSpec, where descriptions of suppliers are unstructured and informal, provided using a free-text without controlled and formalized vocabularies.

In contrast to text-based search algorithms, MANUSQUARE's matchmaking tool takes the semantic approach, which provides a higher rate of search precision and recall than the text-based search. The semantic approach exploits semantic, formal, unambiguous and structured description of production capabilities/capacities and utilizes a domain knowledge with semantic relationship between concepts to compute semantic similarity or semantic distance between the search request and supplier profiles.

With a semantic approach, inference rules contribute to the added-valued, automatic processes in the platform. An inference reasoner derives additional facts from the descriptions resources, driven by the rules, and turns these additional facts into useful knowledge. The inference rules may make explicit previously not stated (unknown) facts about resources and their relationships. There are many examples of this. For example, an inference subsystem may infer that one supplier knows another supplier, by applying an inference rule that specifies the following rule "if supplier A has successfully completed a business process with supplier B, then supplier A knows supplier B, and vice-versa". Obviously, there is no need to assert that a supplier 'knows' another supplier, but 'knows' can be automatically inferred by the platform. Or, if an RFQ requires a hole aspect ratio (a hole depth-to-diameter ratio) more than 10, then the qualified supplier should provide Deep Hole Drilling for creating the hole. Obviously, a customer does not need to know what process operation is required to produce the hole, but the operation can be inferred using domain-specific rules. In summary, by declaring and executing domain-specific inference rules, new facts about resources are created and, in turn, chances to create new production opportunities get higher.

2.3 Current status of Task 2.4

The expected outcome of Task 2.4 is the development and deployment of Semantic Infrastructure, and the integration with the platform. All these activities have been achieved, with the current detailed status as follows:

- Eclipse RDF4J[4] is adopted as a baseline technology for RDF database; installed, configured and integrated with Semantic Infrastructure. Before RDF4J, the Semantic Infrastructure used Ontotext's GraphDB[5], however due to changes of their licence models, the infrastructure has been migrated to RDF4J database.
- Completed development of RESTful API to interact with MANUSQUARE Semantic Infrastructure
    a. CRUD (create-read-update-delete) operations for stakeholder (supplier) profiles, manufacturing process-chains and process descriptions, challenge/idea descriptions, and project (customer needs) descriptions.
    b. Semantic querying over the data stored in a semantic repository,
    c. Inference rules management,
    d. Semi-automatic support to convert semi-structured descriptions of manufacturing resources into their semantic descriptions
    e. Operations for exploring concepts captured in domain-ontologies / taxonomies and property/value smart suggestions service
- Domain-specific ontologies are built and further enhanced, and deployed into the Semantic Infrastructure
- An updated version of the ontology authoring tool has been released. Ontologies are maintained and evolved as being requested from project participants.
- A Docker image of the Semantic Infrastructure is produced and integrated within the platform to support a quick release of the first MVP version.

2.4 Document outline

This deliverable is organized into five sections. The first section provided an executive summary. Section 2 briefly introduces the MANUSQUARE project and the manner in which Semantic Infrastructure relates to other components within the project. Section 3 provides an introduction of RDF baseline technologies. Section 4 places an emphasis of

[4] https://rdf4j.org/
[5] https://www.ontotext.com/products/graphdb/

MANUSQUARE Semantic-Infrastructure development and RESTful API. Section 5 provides instructions about how to run Semantic Infrastructure as a Docker container, while finally Section 6 provides conclusions and some future steps.

3 RDF TECHNOLOGY

There are different semantic languages available, with a higher or lower lever of the adoption from the community, with different levels of technology maturity and toolset support. The most recent and relevant applications of semantic languages, standards and baseline technologies are coming from W3C Semantic Web.[6] W3C Semantic Web standards define RDF, RDFS, OWL, and SPARQL languages, which are briefly introduced in this chapter.

3.1 RDF

Resource Description Framework (RDF) is a format and a data model that a semantic technology uses to describe the resources and to store the resource descriptions or to publish them online. RDF describes the resources in the format of "subject-predicate-object" triples, and thus, forming a graph.

Just as an example to illustrate the RDF language and the subject-predicate-object triple model, the listing below provides a description (and triple data model) of an imaginary resource. In the example, RDF subject is a resource identified with '`http://www.manusquare-project.eu/data/machine/1`', while predicates are '`name`' and '`owner`', with links to '`CNC Machine`' and '`Rapid Manufacturing Ltd`' objects. The objects in this example are the plain/literal values, but the objects can be other RDF resources, which can have their own triples, again with the RDF resources as objects, and so on, making a graph, where subjects and objects are graph nodes, and predicates are graph edges.

An example RDF document, shown in RDF/XML and as a graph:

Listing 1 RDF description example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:msq="http://www.manusquare-project.eu/">
<rdf:Description rdf:about="http://www.manusquare-project.eu/data/machine/1">
  <msq:name>CNC Machine</msq:name >
  <msq:owner>Rapid Manufacturing Ltd</msq:owner >
</rdf:Description>
</rdf:RDF>
```

Subject	Predicate	Object
http://www.manusquare-project.eu/data/machine/1	http://www.manusquare-project.eu/name	"CNC Machine"
http://www.manusquare-project.eu/data/machine/1	http://www.manusquare-project.eu/owner	"Rapid Manufacturing Ltd"

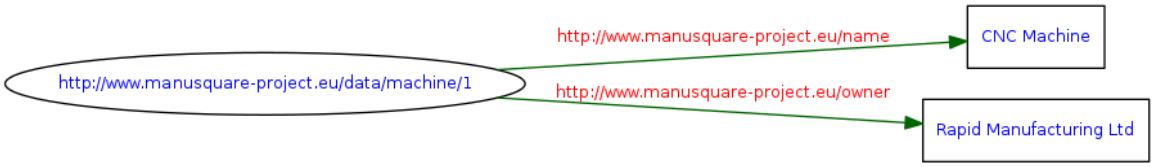


Figure 3 RDF graph example

3.2 RDFS

Resource Description Framework Schema (RDFS) is an extension of RDF that provides language constructs to define the classes and properties of the resources. In particular, RDFS provides the following constructs:

- rdfs:Class to declare/specify a class of the resources.
- rdf:Property is the class to declare the properties.
- rdfs:domain to declare the class of the subject in a triple whose predicate is a rdf:Property.

[6] https://www.w3.org/standards/semanticweb/

- rdfs:range to declare the class or datatype of the object in a triple whose predicate is a rdf:Property.
- rdf:type property to describe that a resource is an instance of a class.
- rdfs:subClassOf allows declaration of hierarchies of classes.

Therefore, by using RDFS, it is possible to structure RDF resources, by describing the classes (types) of the resources, properties of the resources, and by modelling the hierarchical subsumption of the classes and properties. Considering our previous example of RDF description, there can be a class 'Machine' introduced into the description, to be able to link a resource '`http://www.manusquare-project.eu/data/machine/1`' to its type (to classify it), and to precisely specify a domain and range of the relevant properties. With RDFS it easy to define hierarchies of classes, e.g. that 'Machine' is a subclass of 'ManufacturingResource' class.

A RDF Schema Example (in RDF/XML syntax and as a graph):

Listing 2 RDFS example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:msq="http://www.manusquare-project.eu/model#">

<rdfs:Class rdf:about="http://www.manusquare-project.eu/model#ManufacturingResource"/>

<rdfs:Class rdf:about="http://www.manusquare-project.eu/model#Machine">
  <rdfs:subClassOf rdf:resource="http://www.manusquare-project.eu/model#ManufacturingResource"/>
</rdfs:Class>

<rdf:Property rdf:about="http://www.manusquare-project.eu/model#name">
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
  <rdfs:domain rdf:resource="http://www.manusquare-project.eu/model#ManufacturingResource"/>
</rdf:Property>

<rdf:Description rdf:about="http://www.manusquare-project.eu/data/machine/1">
  <rdf:type rdf:resource="http://www.manusquare-project.eu/model#Machine"/>
  <msq:name>CNC Machine</msq:name >
  <msq:owner>Rapid Manufacturing Ltd</msq:owner >
</rdf:Description>

</rdf:RDF>
```

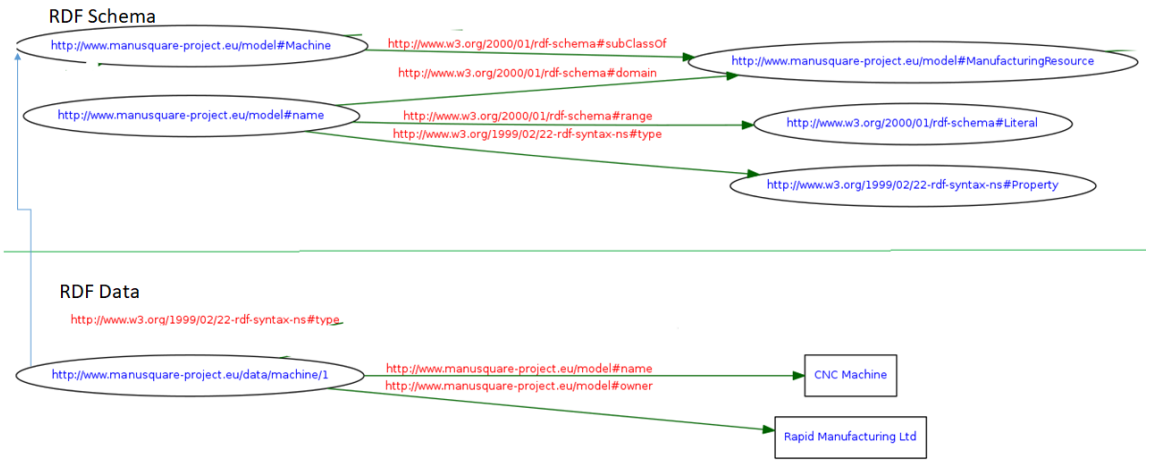


Figure 4 RDF + RDFS graph example

3.3 OWL

Further, Web Ontology Language (**OWL**) extends RDFS and allows for expressing further schema definitions in RDF. OWL overcomes some RDFS limitations in its ability to express rich semantic constructs. For example, RDFS does not allow expressions of property restrictions (value constraints and cardinality constraints) and it has very few constructs to make extensive inferences. Similarly to RDFS, OWL allows description of classes, but using <owl:class> construct, which is actually a subclass of rdfs:Class. But in contrast to RDFS, OWL allows to express equality of individuals (owl:sameAs),

equivalence or disjointness of properties and classes (owl:equivalentClass, owl:equivalentProperty, owl:disjointWith, owl:propertyDisjointWith), transitive properties, inverse properties, functional properties, etc. There are three versions of OWL: OWL Lite, OWL DL, and OWL Full. The main differences are related to the different logics that can be expressed with the language. More complex logics (OWL Full) enable more comprehensive descriptions and automatic inferences, but introduce limitations from the computational point of view. OWL is not actually used in MANUSQUARE as there have not been any requirements to specifically employ OWL semantic and OWL-DL reasoning. Moreover, in this kind of systems where performances and scalability are the key aspects, OWL DL and OWL Full are most likely not the best options due many technical limitation of open sources solutions for large scale OWL reasoning.

3.4 SPARQL

SPARQL is the semantic query and update language designed to query and update RDF databases, and to retrieve and process data stored in RDF format across diverse data sources. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions, supports aggregation, subqueries, negation, creating values by expressions.[7] For example, SPARQL can be used to execute geospatial search for suppliers within certain distance range, as shown in this example:

Listing 3 SPARQL Query example

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX core: <http://manusquare.project.eu/core-manusquare#>

SELECT distinct ?supplier ?location ?distance
WHERE {
 ?supplier geo:asWKT ?location .
 ?supplier rdf:type core:Supplier
 BIND((geof:distance(?location, "POINT(51.502497 -0.108915)"^^geo:wktLiteral, uom:metre)/1000) as
?distance) FILTER (?distance<10)}
```

3.5 SPIN (SPARQL Inference Notation)

SPIN (https://spinrdf.org/) can be used to represent business and inference rules based on SPARQL expressions. SPIN combines concepts from object oriented languages, query languages, and rule-based systems to describe object behaviour on the web of data. SPIN links class definitions with SPARQL queries to capture rules and constraints that formalize the expected behaviour of those classes. Rules are implemented USING SPARQL CONSTRUCT or SPARQL UPDATE requests (INSERT and DELETE) and can be used to calculate the value of a property based on other properties, to derive new property, to classify instance, to isolate a set of rules to be executed under certain conditions to support incremental reasoning, to initialize certain values when a resource is first created, or to drive interactive applications, etc. Beside SPIN means, rules can be implemented on differ ways, e.g. using APACHE Jena[8] or just as plain SPARQL CONSTRUCT queries.

3.6 RDF Triplestores

An RDF triplestore is a database for the storage and retrieval of RDF triples with SPARQL Query and SPARQL Update capabilities. Many triple stores are today available[9], but not all of them are open-source and free license. Most importantly, of those open/free source only few RDF triplestore are still actively maintained by the vendors. We have limited our focus only on those that have had a stable release within last year and that are still being actively supported and maintained either by the community or directly by vendors. Table 1 shows some of the available triplestores.

Table 1 Triplestores

Name	Licence	Reference	Last release

[7] https://www.w3.org/TR/sparql11-query/
[8] https://jena.apache.org
[9] For a quick reference see https://en.wikipedia.org/wiki/Comparison_of_triplestores

RDF4J (Sesame)	Eclipse Public license	rdf4j.org	Oct 26, 2019
GraphDB by Ontotext	Commercial/Proprietary	ontotext.com/products/graphdb	Sept 30, 2019
Stardog	Commercial/Proprietary	stardog.com	Aug 7, 2019
Apache Rya	Apache 2	rya.apache.org	July 7, 2019
Halyard	Apache 2	github.com/Merck/Halyard	June 2, 2019
Apache Jena TDB	Apache 2	jena.apache.org	May 5, 2019
Blazegraph	GNU GPL (v.2)	www.blazegraph.com	Mar 19, 2019
KiWi (Apache Marmotta)	Apache 2	marmotta.apache.org/kiwi	June 6, 2018

The Semantic Infrastructure in MANUSQUARE uses RDF4J as it is very actively maintained by Eclipse foundation, its open source and released under public/free licence, and provides real-time RDFS inference, SPARQL and SPIN implementation.

Besides RDF4J, we have successfully used GraphDB as a triplestore for the Semantic Infrastructure. However, GraphDB has a commercial licence. GraphDB, in contrast to RDF4J, provides capabilities for OWL reasoning. This is to say that in the future, the MANUSQUARE platform adopters, are not limited to use only RDF4J and RDFS reasoning, but can deploy commercial triples stores such is GraphDB or Stardog to apply OWL reasoning if needed.

Another RDF4J-alternative configuration can be to deploy Apache Jena TDB database over Apache Fuseki SPARQL Server[10].

4 MANUSQUARE SEMANTIC INFRASTRUCTURE

MANUSQUARE's Semantic Infrastructure, at the first place, provides CRUD (create-read-update-delete) operations for the first-order citizens of the platform including stakeholder profiles, process-chains and process descriptions (i.e. production and engineering capability and capacity description), idea descriptions and project (user needs) descriptions. In addition to those CRUD capabilities, the Semantic Infrastructure provides a support for semantic inferencing, matchmaking and querying over structured domain ontologies/taxonomies, and over the data stored in a semantic repository. RDF is the representation language for the data (instances), while concepts and their relationships are captured using RDFS/OWL language. Semantic Infrastructure services are exposed via RESTful API, with JSON as a main representation format of request/response bodies.

4.1 Requirements Analysis

The main requirements for the Semantic Infrastructure have been elicited in three stages:

- Workshop on Semantic Infrastructure @ WP4 kick-off meeting. Lugano CH, January 2019
- Tools Integration and (Ecosystem) Data Model Workshop @ GA, Milan, IT, 20-21 May 2019.
- Bilateral discussions with the SUPSI, SINTEF, Holonix on various integration points, during the period of July-November 2019.

The requirements captured this deliverable are mostly final ones, however, as the project goes forwards with the integration of the tools (M26) and industrial pilots, there can be some additional specific requirements and requests for changes that would be needed to get addressed either by additional implementation (as platform support activity) or by system reconfiguration (for example to add new sector-specific ontologies/taxonomies). The updates, if any, will be inserted in Deliverable D4.6 at Month 26.

4.1.1 Requirements from Project-RFQ Creation tool perspective

Table 2 Requirements from Project/RFQ perspective

Tool Description/Required Interaction/Benefit Analysis

[10] https://jena.apache.org/documentation/fuseki2/index.html

Project creation description	A web-based tool (web GUI) will be used by customers to create new projects (e.g. RFQ – requests for quotation). During the projects creation a customer will provide a project name and other details describing the project. Those details may include desired process types, material types etc. process or material related attributes and will be captured as set of project attributes via hasSupplierAttribute and hasProcessAttribute relationships (according to extended version of Ecosystem Data Model).
Interaction with Semantic Inf.	**Req.1** Semantic Infrastructure should provide suggestions (autocomplete for input fields) to assist to the users during the project creation. In addition to that, there should be: • suggestion of most frequently used properties and their values for certain material e.g. for steel or certain process • suggestion of other most frequently used details during the project creation **Req.2**. Project data entry should be stored in a semantic repository. The entry will contain a project name, project type (if any), and list of attributes related to desired features of suppliers and processes. An inference ruleset can be run on the project description to infer missing/uncomplete elements. **Req.3.** Existing projects are becoming a part of accumulated knowledge in the platform to assist for creation of new projects.
Benefits of interaction with Semantic Infrastructure	• Manufacturing/production knowledge acquisition through controlled crowdsourcing activity (via project creation template) • Platform-provided assistance/knowledge for projects creation

4.1.2 Requirements from Challenge/Idea Creation tool perspective

Table 3 Requirements from Idea Creation perspective

Tool Description/Required Interaction/Benefit Analysis	
Idea creation description	Using Idea Manager tool, the users will publish their ideas and will be looking for collaborative idea development and design. An idea will be tagged using semantic tags in order to enable intelligent discovery and search of challenges and related ideas.
Interaction with Semantic Inf.	**Req.4.** Semantic Infrastructure should provide list of tags (concepts from taxonomical formalization) for tagging ideas by product type, material type, etc. Idea (Id + its Tags) should be stored in a semantic database. **Req.5.** Semantic Infrastructure should provide suggestions on how to describe the things/idea in terms of their functionalities, materials, attributes. Knowledge from Semantic Infrastructure (from project descriptions and process descriptions) can be used to provide these suggestions. **Req.6.** User should be able to introduce new tags and push them to Semantic Infrastructure for future use. These new tags should be approved by a platform manager / domain expert and in a proper way added/introduced into the existing taxonomies by relating them with subsumed and parent concepts.
Benefits of interactions with Semantic Infrastructure	• Reach more people through the tags (tag-based search) • Activities on the platform can provide additional information to innovation manager (is there available capacity for production) • Knowledge acquisition through new tags

4.1.3 Requirements from Matchmaking tool perspective

Table 4 Requirements from Matchmaking perspective

Tool Description/Required Interaction/Benefit Analysis	
Matchmaking description	Matchmaking tool, in one of its stages of matchmaking, matches a customer search request with suppliers' profiles in order to discover suppliers that match the request.

Interaction with Semantic Inf.	**Req.7.** Querying database of suppliers by process type, by material type, by human capabilities, attribute type and attribute values, etc. Search should support geospatial filtering based on given distance and current position of customer.
Benefits of interactions with Semantic Infrastructure	• Semantic search with a greater precision and recall than traditional text-based search

4.1.4 Requirements from Supplier (User) Profiling tool perspective

Table 5 Requirements from User Profile perspective

Tool Description/Required Interaction/Benefit Analysis	
Profiling description	User profiling tool is responsible for creation and evolution of user (suppliers) profiles including a company name, location, certifications, etc.
Interaction with Semantic Inf.	**Req.8.** Supplier's profiles and description of supplier's manufacturing resources, know-how and technologies should be stored in a semantic database **Req.9.** Profiling tool may have different query requests related to the supplier profiles (e.g. get competences of supplier, get address, get certifications, etc.), so the Semantic Infrastructure should provide means for querying the profiles. **Req.10.** During the creation of manufacturing resources description, the Semantic Infrastructure should provide suggestions about most used properties and values for particular resource description. Thus, the Semantic Infrastructure should be populated with the needed semantic models to support that assistance, and therefore, there should be a semi-automatic conversion of semi-structured textual descriptions of generic resource types into respective semantic models, to save time and cost during extensive supplier profile creation.
Benefits of interactions with Semantic Infrastructure	n/a

4.1.5 Requirements from Reputation Manager tool perspective

Table 6 Requirements from Reputation Manager perspective

Tool Description/Required Interaction/Benefit Analysis	
Tool description	Reputation tool calculates reputation index for suppliers based on history of transactions of the platform, users' feedback (ratings and reviews) and profile completeness rate.
Interaction with Semantic Inf.	**Req. 11.** Semantic infrastructure should provide means for querying the supplier profiles.
Benefits of SI interactions	n/a

4.2 Design & Architecture

Figure 5 illustrates a high level view of architecture of MANUSQUARE Semantic Infrastructure. The composing layers are described in the following paragraphs.

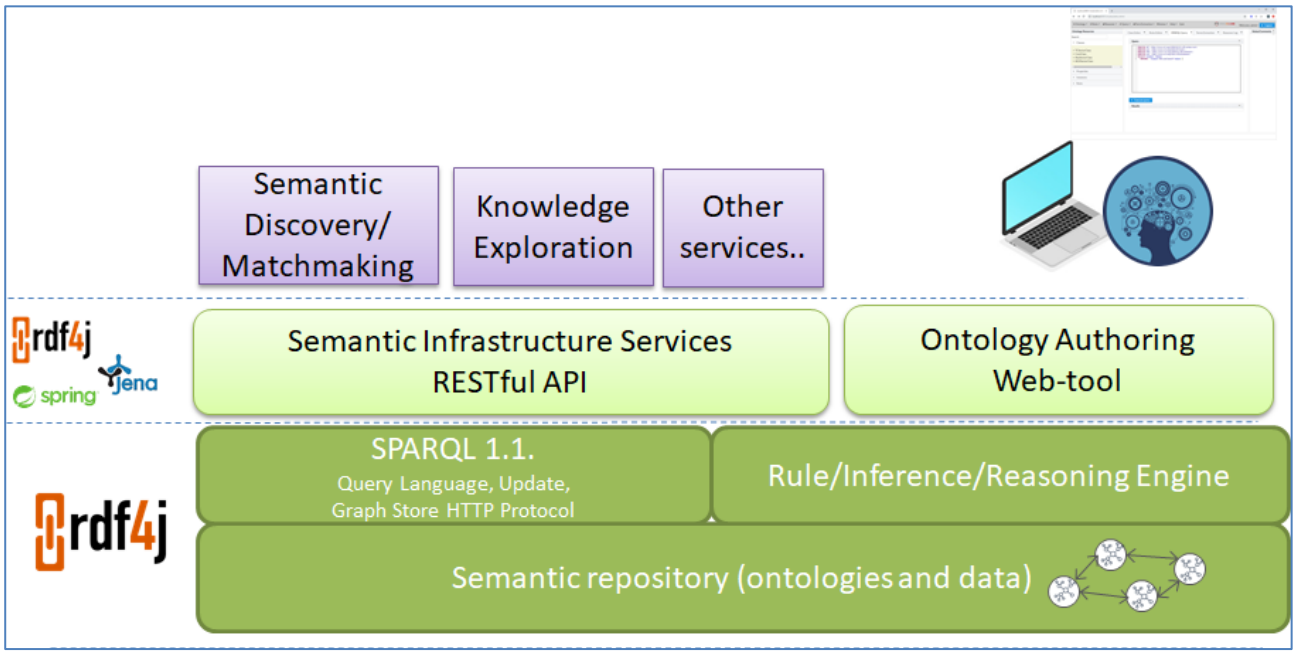


Figure 5 High-level architecture of MANUSQUARE Semantic Infrastructure

4.2.1 Core Layer

The core layer of the infrastructure consists of a semantic repository (RDF triplestore) with querying support as well as inference rules support.

4.2.1.1 Semantic repository

Semantic repository is a persistent storage of domain ontologies and other semantically relevant resources of MANUSQUARE platform (supplier profiles, process descriptions, etc), including also inference rules, in a RDF triple form. Triples in MANUSQUARE platform are organized into a structure of **named graphs.** Each named graph captures only RDF descriptions about particular core-level resource such as **stakeholder** (supplier) **profile**, **process-chains** / **process description** (i.e. production and engineering capability and capacity description), **idea description**, **project** (user needs) **description**, and finally domain-specific inference rule. Each named graph is uniquely identified by its IRI (URI). MANUSQUARE Semantic Infrastructure deploys a certain convention for naming the named graphs in order to clearly organize graphs into four main sets:

- **stakeholders,**
- **processchains,**
- **ideas,**
- **projects**

For example, a description of supplier that has unique identifier Id=1, will be transformed into an RDF resource that has URI <urn:stakeholders:1> and that belongs to the named graph also named with <urn:stakeholders:1>, as graphically illustrated in a figure below.

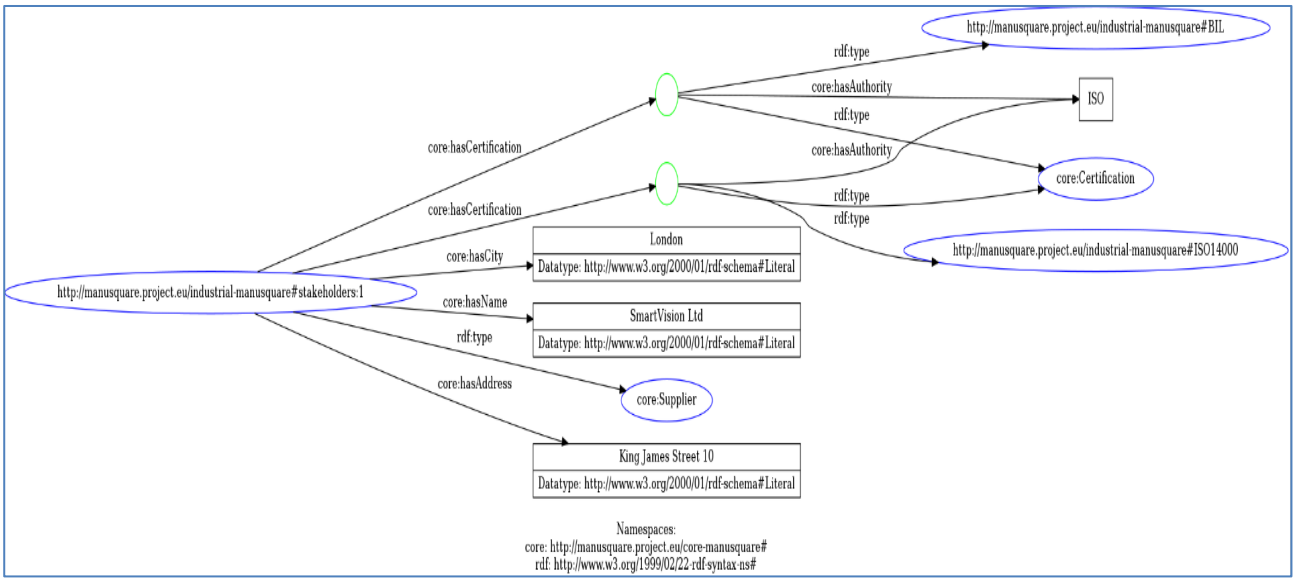


Figure 6 Graph of Supplier profile

Listing 4 RDF description of Supplier Profile

```
@prefix : <http://manusquare.project.eu/industrial-manusquare#> .
@prefix core: <http://manusquare.project.eu/core-manusquare#> .

<http://manusquare.project.eu/industrial-manusquare#stakeholders:1> a core:Supplier ;
    core:hasAddress "King James Street 10"^^rdfs:Literal ;
    core:hasCertification [ a core:Certification,
                urn:BIL ;
            core:hasAuthority "ISO" ],
        [ a core:Certification,
                urn:ISO14000 ;
            core:hasAuthority "ISO" ] ;
    core:hasCity "London"^^rdfs:Literal ;
    core:hasName "SmartVision Ltd"^^rdfs:Literal .
```

By employing the graph naming convention, the repository gets logically structured in four sets with a controlled creation of unique URIs, which in turn, makes the repository more maintainable.

As a technology for implementation of the semantic repository, we have chosen Eclipse RDF4J (https://rdf4j.org/ http://graphdb.ontotext.com/). RDF4J, as introduced in the chapter 3, is semantic RDF graph database. It implements the W3C SPARQL Protocol specification and supports all RDF serialization formats (RDF/XML, N3, Turtle, N-Quads, etc.). Most importantly, it provides RDFS inference in real time, which means that any operation that changes the content of the repository will trigger execution of inference rules. The reasoning strategy applied is one of so called total materialisation, where the inference rules are applied repeatedly to the asserted (explicit) statements until no further inferred (implicit) statements are produced. Hence, after each update to the repository, the inferred closure is computed and made available for further query evaluation or retrieval.

4.2.1.2 Querying

RDF4J implements W3C SPARQL, which is introduced in the chapter 3 as a query language and protocol for RDF. MANUSQUARE Semantic Infrastructure uses SPARQL 1.1 Query Language for querying the semantic store, SPARQL 1.1 Update as an update language for RDF graphs (create, drop, insert, delete), and SPARQL 1.1 Graph Store HTTP Protocol for graphs operation over HTTP put/delete/post/get operations.

Listing 5 SPARQL query to search suppliers that produce more than 1000 pieces of CathodeTubes

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX core: <http://manusquare.project.eu/core-manusquare#>
PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#>
prefix owl:   <http://www.w3.org/2002/07/owl#>

SELECT distinct ?supplier ?output  ?quantity
WHERE { ?processChain core:hasSupplier ?supplier .   ?supplier core:hasName ?name.
        ?processChain core:hasProcess ?process. ?process core:hasOutput ?output .?output rdf:type
ind:CathodeTubes .
?output core:hasQuantity ?quantity
    FILTER(?quantity > 1000)
}
```

The query can be executed in two mode: taking into account also inferred statements or executing the query without inferred statements taken into account.

4.2.1.3 Inferencing

The platform is capable of RDFS inference. As MANUSQUARE does not yet need the complexity of the most expressive supported semantics (owl-ones), we opted for RDFS, which is less complex and thus results in faster inference. So far gathered requirements for the platform can be addressed with an entailment of RDFS semantics of subClassOf and subPropertyOf relationships and classification based on domain and range definitions of RDF properties. SubClassOf is used as a main construct to structure domain-specific taxonomies of MANUSQUARE core concepts, hence, the query answering in MANUSQUARE can give more results based on inferred classifications of instances. This means that types of different resources captured in the semantic repository can be inferred based on subClassOf formalization of MANUSQUARE Core concept taxonomies.

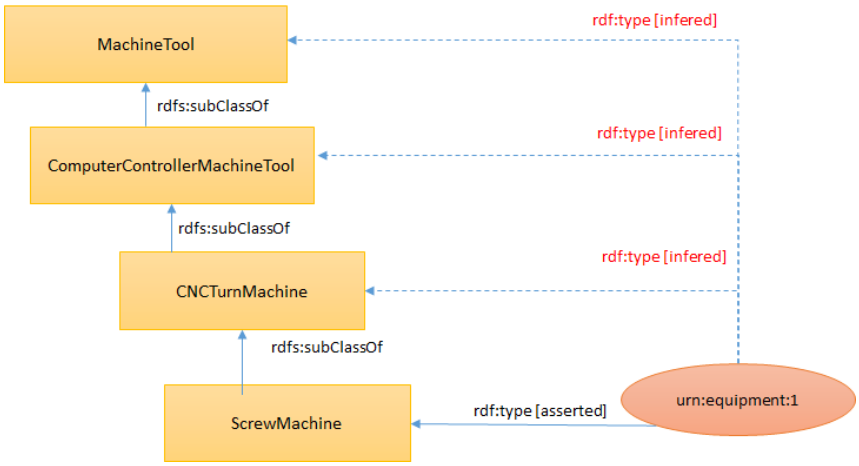


Figure 7 RDFS subClassOf inference

In addition to RDFS inference, as introduced in D2.3, the Semantic Infrastructure supports management of domain-specific SPIN rules. Rules can be added into the semantic repository and then executed to infer new facts. Just as an example, a rule that infers that business party knows other business party, if they have established a Quotation (by RFQ), can be declared as shown in Listing 6.

Listing 6 SPIN inference rule example

```
core:Quotationspin:rule ind:ruleKnows.
        ind:ruleKnows rdf:type sp:Construct .
        ind:ruleKnows sp:text
                '''REFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#>
                PREFIX core: <http://manusquare.project.eu/core-manusquare#>
                construct {?party1 core:knows ?party2 . ?party2  core:knows ?party1}
                where { ?p rdf:type core:Project. ?p core:hasCustomer ?party1. ?p core:hasQuotation
                ?q . ?q core:hasSupplier ?party2 }''' .
```

4.2.2 Ontology Authoring Layer

A web-based tool for domain-experts and platform managers is developed in T2.2 to facilitate development, expansion and evolution of the domain-specific ontologies in MANUSQUARE. It provides an environment for creation and evolution of domain-specific ontologies, SPARQL Query + Update execution, and front-end for creation/management of inference rules (T2.3). The ontology authoring tool is fully described in the D2.2 deliverable, hence its description is out of scope of this deliverable.

4.2.3 Services (RESTful API) Layer

On top of a semantic repository, the MANUSQUARE Semantic Infrastructure provides services to support:

- creation of semantic descriptions of manufacturing supply and demand-centered resources
- semantic matchmaking of manufacturing supply and demand, thru exploration of manufacturing knowledge and semantic relationships established between resources stored in a semantic repository
- operations to evolve and explore ontologies and aggregated knowledge
- domain-specific inference rules management
- acquisition of new concepts and properties

The services are available via a RESTful API, and documented with Swagger, as illustrated in figure below.

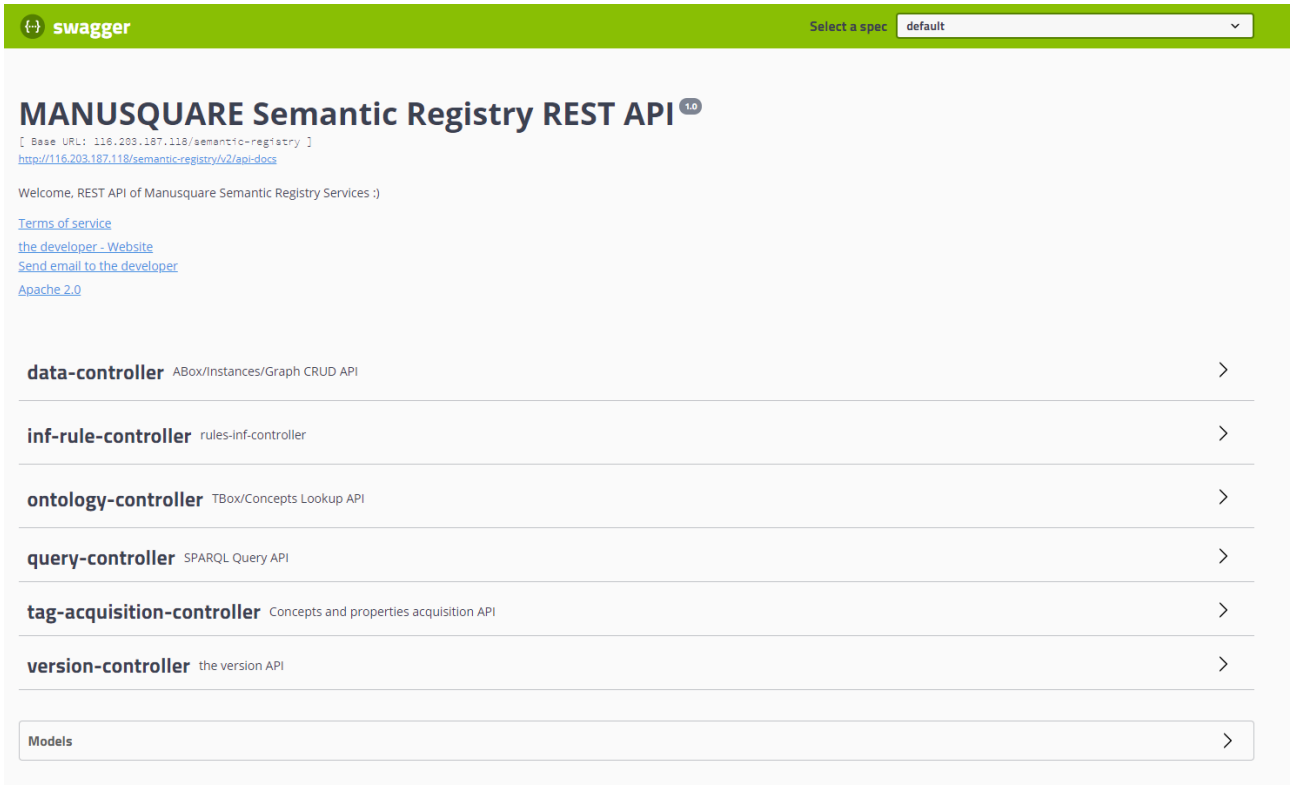


Figure 8 Swagger Documentation Screen of API

The Semantic Infrastructure is implemented using open source technologies including Eclipse RDF4J and Spring (https://spring.io/ for implementation of back-end service layer, Apache Jena API (https://jena.apache.org/) is used for internal handling of RDF, RDFS, OWL data and models, while Primefaces (https://www.primefaces.org/) is used for development of front-end components and controls.

4.3 Restful API Details

4.3.1 Access Control

Security features such as authentication and authorisation are implemented within the service-layer, so that only authorized users can read or write repository content. Access the RESTful API operations is restricted only to authorized users. A token 'c5ec0a8b494a30ed41d4d6fe3107990b' can be used for read-only operations and for execution of SPARQL Select queries. Access control data (security tokens and permissions) are stored inside the repository within the graph named **<urn:users>.** For example, the matchmaking tool has only read permissions and user can be configure like this:

Listing 7 Configuration of access control and security token

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix urn: <http://manusquare.project.eu/industrial-manusquare#> .

urn:user:2 a rdfs:Resource;
  urn:privileges "folksonomy.read", "graph.ideas.read", "graph.processchains.read",
    "graph.projects.read", "graph.stakeholders.read", "sparql.execute" ;
  urn:token "secretTokenHere";
  urn:username "matchmaking" .
```

4.3.2 Request Controllers

A RESTful API is organized into a five controllers, as shown in table below.

Table 7 API Request controllers

Name	Description	Requirements addressed
Data-controller	Operations needed for CRUD (create-read-update-delete) of MANUSQUARE resources that are required to be stored in the semantic repository	Req#2, Req#3., Req#4, Req#8, Req#9
Ontology-controller	Operations for exploration of domain-specific taxonomies such as MaterialTypes, ProcessTypes or EquipmentTypes, to assist to the creation of projects and manufacturing capabilities description, and operation for converting a structured template of resource description into the platform knowledge.	Req#1, Req#4, Req#5, Req#10
Query-controller	Endpoint for sending arbitrary SPARQL queries to the semantic repository.	Req#7., Req#9., Req#11
Tag-acquisition-controller	Operations for insertion/removal of new concepts and properties into/from a temporary folksonomy. The new concepts/properties are at some point transferred into a controlled domain-specific taxonomies (ontologies), by a platform manager (domain expert) using the ontology authoring tool.	Req#6.
Inf-rule controller	Set of operations for CRUD management of domain-specific inference rules.	Req#2

4.3.3 Data Controller

Data in the semantic repository are organized into a structure of named graphs that capture RDF statements about **stakeholder** (supplier) **profiles**, **process-chains** / **process descriptions** (i.e. production and engineering capability and capacity description), **ideas description**, and **projects** (user needs) **descriptions.** Each named graph has to be uniquely identified by its IRI (URI). A convention for formulation of IRIs of the named graphs helps to logically and visually organize graphs into clear sets of stakeholders, processchains, ideas, and projects, and therefore, should be applied during the graphs creation, retrieval and removal. Therefore, the Semantic Infrastructure provides a common API for creation, retrieval and removal of the named graphs (CRUD operation), irrespectively whether the named graph captures the statements that describe **stakeholders**, **processchains**, **ideas**, or **projects.**

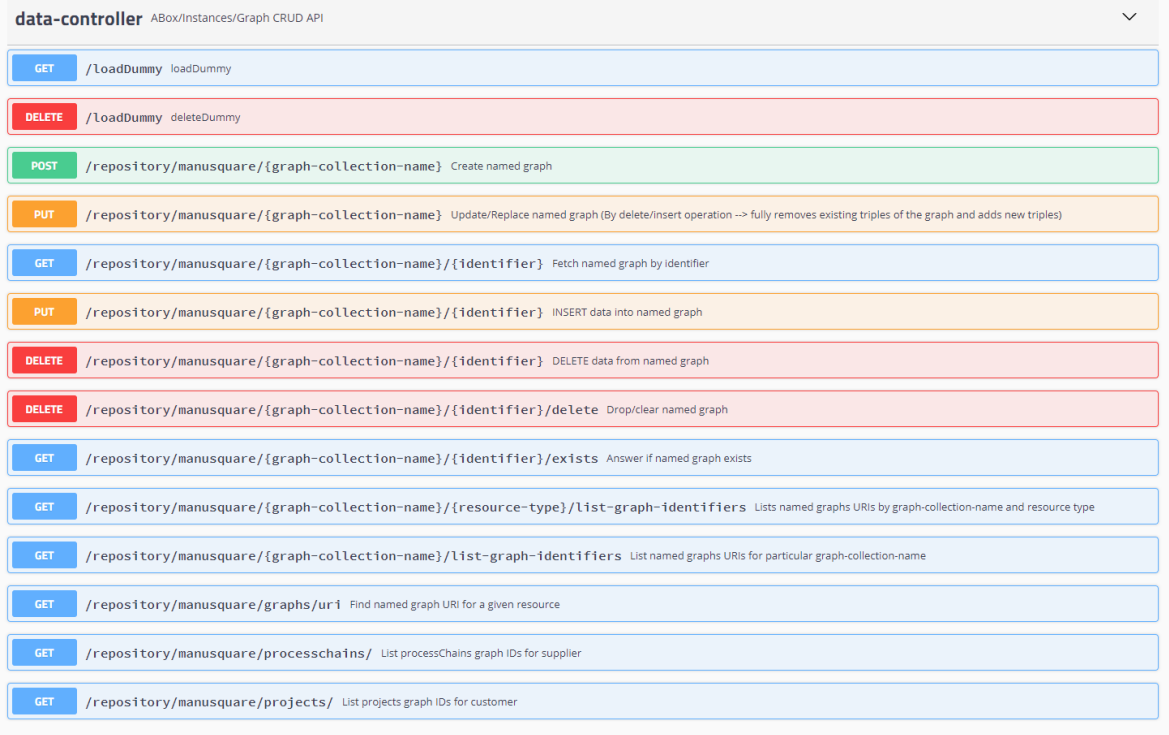
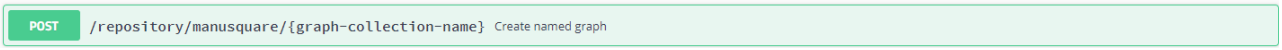


Figure 9 Swagger: Data CRUD Requests Controller

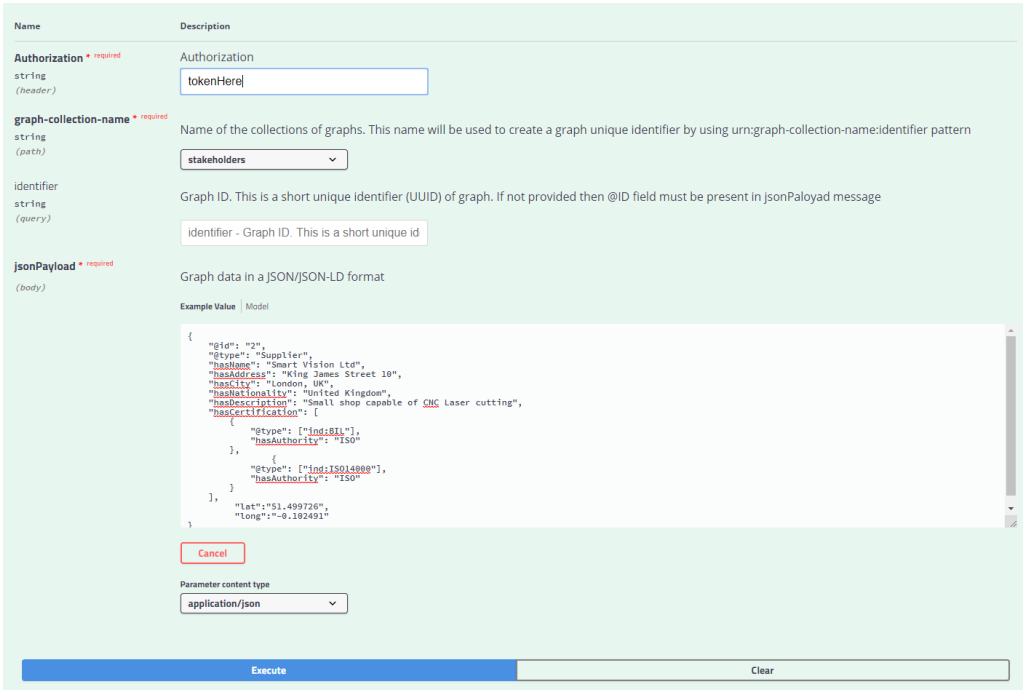
CRUD operations are managed over HTTP POST, GET, PUT, and DELETE request, respectively. In the following subsection, we provide some examples.

4.3.3.1 Insert new supplier profile

To create a named graph representing a supplier profile, use HTTP POST operation



Example:



Graph-collection-name has to be 'stakeholders', while if @id is not provided within the JSON-LD, then id must be provided within parameter identifier. JSON-LD[11] is used as a representation format of a request payload. Please note that @type determines the type of a resource, in this case a Supplier. @id must be a unique identifier (UUID), for example generated by a numeric sequence generator.

Listing 8 Supplier Profile in JSON-LD

```
{
    "@id": "2",
    "@type": "Supplier",
    "hasName": "Smart Vision Ltd",
    "hasAddress": "King James Street 10",
    "hasCity": "London, UK",
    "hasNationality": "United Kingdom",
    "hasDescription": "Small shop capable of CNC Laser cutting",
    "hasCertification": [
        {
            "@type": ["ind:BIL"],
            "hasAuthority": "ISO"
        },
                {
            "@type": ["ind:ISO14000"],
            "hasAuthority": "ISO"
        }
    ],
         "lat":"51.499726",
         "long":"-0.102491"
}
```

4.3.3.2 Delete supplier profile

To delete a named graph representing a supplier profile the following HTTP Delete operation has to be involved:

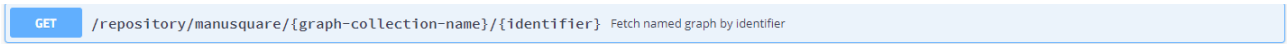
Also in this case, specific graph-collection-name and identifier parameter must be provided. For example, if we want to delete supplier profile created in the previous POST example, a call is:

curl -X DELETE "http://host:port/semantic-registry/repository/manusquare/stakeholders/2/delete" -H "accept: */*" -H "Authorization: secretToken"

Importantly, please note that there is no cascade delete for processChains belonging to a supplier. If a supplier profile has to be deleted all together with processChains belonging to that supplier, then subsequent calls should be made to delete processChains.

4.3.3.3 Read supplier profile

Read operations are managed by GET requests. For example, to retrieve the supplier profile we inserted in the previous example, call a GET operation, with includeInfered set to false to get only asserted statements.



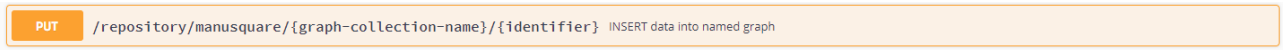
curl -X GET "http://host:port/semantic-registry/repository/manusquare/stakeholders/2?includeInfered=false" -H "accept: application/json" -H "Authorization: secretToken "

[11] JSON for Linking Data (https://json-ld.org/) is a JSON representation for RDF data. It is a fully JSON valid document and very similar to traditional JSON representation of data objects. In MANUSQUARE, there might be a need to transform traditional JSON representation of various MANUSQUARE resources such as Supplier Profile or Process Chain, into their corresponding JSON-LD format, however, that transformation is out-of-scope of the Semantic Infrastructure, and should be addressed elsewhere in the architecture (Ecosystem Data Manager).

4.3.3.4 *Update supplier profile*

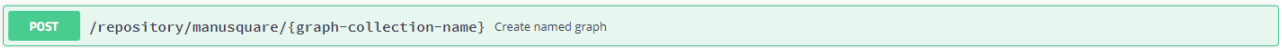
Update operations are managed by PUT requests. For example, to replace the supplier profile we inserted in the previous example with updated data, a call will be via

If the supplier profile only has to be fulfilled with new data, it is also possible to insert that additional data by this operation, which also takes graph-collection-name, identifier parameters and JSON-LD data.



4.3.3.5 *Insert process-chain (production capability / resource) description*

To create a named graph representing a process-chain (description of production capability or machining resource description), the following POST operation has to be used:



Graph-collection-name, in this case has to be '**processchains'**. Each entry must have unique identifier, either provided within JSON-LD as @Id field or by request parameter named 'identifier'.

A request payload example:

```
{
"@id": "urn:processchain:2",
"@type": "ProcessChain",
"hasMeanTime": 1600,
"hasName": "JPM-GeneralProcessChain",
"hasSupplier": [{
"@id": "urn:stakeholders:2"  -- here you must have uuid of supplier profile stored in the Sem Inf.
}],
"hasProcess": [{
        "@id": "ind:process:uuidHere",
        "@type": ["Process", "ind:CNCGrinding"],
        "hasResource":[
                {"@id":"ind:equipment:uuidHere", "@type":"ind:ScrewMachine",
                "hasCapability":[{"@type":["ind:CNCGrinding"]}]}],
        "hasAttribute" :[
                {"@type":"AttributeMaterialType","hasValue":[{"@type":["ind:Steel"]}]},
                {"@type":"ind:xAxis","hasValue":"700","hasUnitOfMeasure": [{"hasName":"mm/min"}]},
                {"@type":"ind:yAxis","hasValue":"400","hasUnitOfMeasure": [{"hasName":"mm/min"}]},
                {"@type":"ind:zAxis","hasValue":"300","hasUnitOfMeasure": [{"hasName":"mm/min"}]},
                {"@type":"ind:FeedRate","hasValue":"10","hasUnitOfMeasure": [{"hasName":"mpm"}]},
                {"@type":"ind:WheelSpeed","hasValue":"20","hasUnitOfMeasure": [{"hasName":"rpm"}]},
                {"@type":"ind:FlowRate","hasValue":"300","hasUnitOfMeasure": [{"hasName":"lpm"}]},
                {"@type":"ind:NanoParticleSize","hasValue":"10","hasUnitOfMeasure":
[{"hasName":"mm"}]}
                ]
        }
    ]
}
```

Listing 9 Manufacturing Resource Description in JSON-LD

4.3.3.6 *Delete process-chain description*

To delete a named graph representing a processChain description the following HTTP Delete operation has to be invoked:

Also in this case, specific graph-collection-name and identifier parameter must be provided, For example, to delete a named graph that we created in the example above, a call is:

curl -X DELETE "http://host:port/semantic-registry/repository/manusquare/processchains/2/delete" -H "accept: */*" -H "Authorization: secretToken"

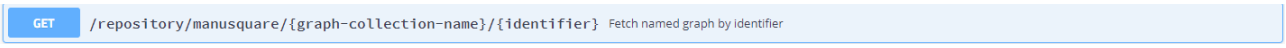
4.3.3.7 Update process-chain description

Update operations are managed by PUT requests. For example, to replace the process-chain description with the updated data, a call needs to be made by to the following PUT operation.

If the process-chain description only has to be fulfilled with new data, e.g. additional attributes related to process, it is also possible to insert that additional data by another PUT operation, which also takes graph-collection-name, identifier parameters and JSON-LD data.

4.3.3.8 Read process-chain description

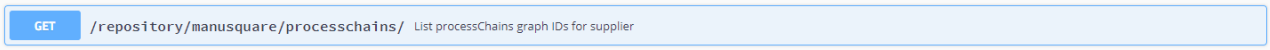
Read operations are managed by GET requests. For example, to retrieve the process-chain description the following GET operation should be used with graph-collection-name set to processchains and with given identifier of process chain.



curl -X GET "http://host:port/semantic-registry/repository/manusquare/processchains/2?includeInfered=false" -H "accept: application/json" -H "Authorization: secretToken "

IncludeInfered should be set to false to reduce result only to asserted statements.

4.3.3.9 List process-chains URIs for particular supplier



curl -X GET "http:// host:port/semantic-registry/repository/manusquare/processchains/?isShortId=true&supplierIdentifier=2" -H "accept: application/json" -H "Authorization: secretToken"
returns list of found URIs: `[ "urn:processchains:2"]`

4.3.3.10 CRUD operations for Project and Idea resources

CRUD operations for Project and Idea resources are also managed over the mentioned HTTP POST, GET, PUT, and DELETE request. Only difference is in the value of graph-collection-name parameter. For CRUD operations of Projects, graph-collection-name must be set to 'projects', while for idea must be set to 'ideas'. The listings below provide simple examples for JSON-LD representation of a project and idea.

Listing 10 Project Request in JSON-LD format

```
{
"@type": "Project",
"hasName": "Example Project",
"hasDescription": "Example Project Description",
"hasDeadLine": "2019-07-22 08:55:00",
"hasSelectionType":"Automatic",
"hasSupplierMaxDistance":"100",
"hasSize":"2",
"hasServicePolicy":"true",
"hasProcessAttribute" :[
{"@type":["Attribute","ind:AspectRatio"],"hasValue":"11"},
{"@type":["Attribute","ind:MinSpindleSpeed"],"hasValue":"8000","hasUnitOfMeasure":
[{"hasName":"rpm"}]},
{"@type":["Attribute","ind:Tolerance"],"hasValue":"1","hasUnitOfMeasure": [{"hasName":"mm"}]},
{"@type":["Attribute"],"hasValue":{"@id":"ind:LaserCuttingMachine"}}
],
"hasSupplierAttribute" :[
{"@type":"Attribute","hasValue":{"@id":"ind:ISO9000"}} ]}
```

Listing 11 Idea Description in JSON-ID (simple example)

```
{
    "@type": "Idea",
    "hasName": "My nice idea",
    "hasDescription": "I want to produce a new bike",
    "hasTag":[
        { "@type": ["ind:CarbonSteel"]},
{"@type": ["ind:MotorcycleBicycleAndPartsManufacturing"]}]
}
```

4.3.4 Ontology Controller

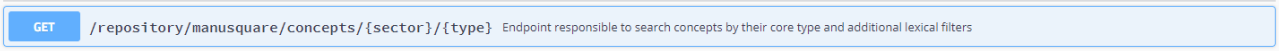
Ontology Controller manages requests for exploration of domain-specific taxonomies such as taxonomy of Material Types, taxonomy of Process Types, taxonomy of Equipment Types, taxonomy of Certification Types, to assist to the creation of projects (RFQs) and to create supplier profiles and their production capabilities descriptions. Currently provided GET operations are listed in the table below.

Table 8 Ontology Controller GET operations

GET operations of Ontology Controller	Description
/repository/manusquare/concepts/{sector}/{type}	Handles requests for searching concepts captured in domain-specific ontology by their core type and additional lexical filters
/repository/manusquare/concepts/coretypes	Provides list of core concept types suitable for concepts lookup using /repository/manusquare/concepts/{type} endpoint
/repository/manusquare/concepts/taxonomy	Provides list of concepts that are subsumed (subclassOf-ed) from a given concept. E.g. to retrieve all Certifications
/repository/manusquare/ontology	Get the whole MANUSQUARE ontology as RDF/XML file
/repository/manusquare/properties/suggest	Operation responsible to provide suggestions of properties for descriptions of concepts/instances
/repository/manusquare/properties/vals/suggest	Operation responsible to provide suggestions of values for properties for descriptions of concepts/instances

In following subsections, examples are provided.

4.3.4.1 Suggestions for autocomplete of inputs for material/process/equipment/ types*

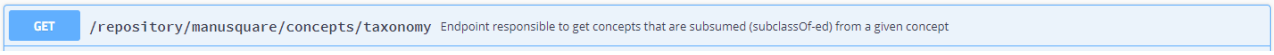


Example: search for all material types whose name contains word ‘steel’, across all domain-specific taxonomies, is specified by a lexical filter that uses operator ‘contains’, over field ‘name’, with value ‘steel’.

```
[{"field" : "name","operator" : "contains","value" : "steel"}]
```

curl -X GET "http://host:port/semantic-registry/repository/manusquare/concepts/any/Material?filters=%5B%7B%22field%22%20%3A%20%22name%22%2C%22operator%22%20%3A%20%22contains%22%2C%22value%22%20%3A%20%22steel%22%7D%5D" -H "accept: application/json" -H "Authorization: secretToken"
Besides the operator ‘contains’, the operator ‘startsWith’ is also supported. Filtering options are not limited to one field, but a search can be performed on multiple fields (for example, on ‘label’ field).

4.3.4.2 Retrieval of concepts of particular type



Example: to get list of all certification types, execute

curl -X GET "http://host:port/semantic-registry/repository/manusquare/concepts/taxonomy?conceptName=**Certification**&indirect=true" -H "accept: application/json" -H "Authorization: secretToken"

Or, to get list of all process types:

curl -X GET "http:// host:port/semantic-registry/repository/manusquare/concepts/taxonomy?conceptName=**ProcessType**&indirect=true" -H "accept: application/json" -H "Authorization: secretToken"

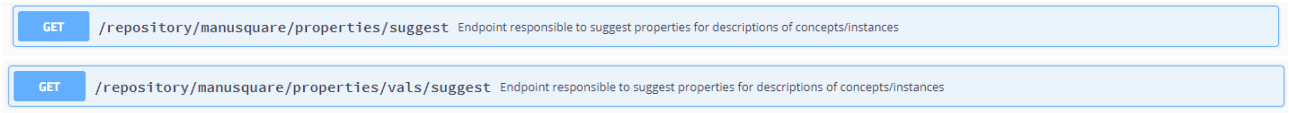
Result is a list of concepts, with their URI (Id) and labels, as illustrated in the listing below.


```
[..
  {
    "name": "CNCMilling",
    "uri": "http://manusquare.project.eu/industrial-manusquare#CNCMilling",
    "description": "",
    "labels": {
      "en": "CNC Milling"
    }
  },
  {
    "name": "ChamferMilling",
    "uri": "http://manusquare.project.eu/industrial-manusquare#ChamferMilling",
    "description": "",
    "labels": {
      "en": "Chamfer Milling"
    }
  },
  {
    "name": "EndMilling",
    "uri": "http://manusquare.project.eu/industrial-manusquare#EndMilling",
    "description": "",
    "labels": {
      "en": "End Milling"
    }
  },
  {
    "name": "FaceMilling",
    "uri": "http://manusquare.project.eu/industrial-manusquare#FaceMilling",
    "description": "",
    "labels": {
      "en": "Face Milling"
    }
  },
  {
    "name": "Milling",
    "uri": "http://manusquare.project.eu/industrial-manusquare#Milling",
    "description": "",
    "labels": {
      "en": "Milling"
    }
  },
  {
    "name": "PrecisionMilling",
    "uri": "http://manusquare.project.eu/industrial-manusquare#PrecisionMilling",
    "description": "",
    "labels": {
      "en": "Precision Milling"
    }
  },
..
]
```


Listing 12 Concept retrieval result

4.3.4.3 Suggestions of properties and value

Semantic Infrastructure can suggest, based on accumulated knowledge in the platform, the most used **properties** and their **values** for creation of descriptions of supplier profiles, for creation of process description, or for equipment descriptions. This is supported by these two GET operations:



Example: get list of properties used to describe a ScrewMachine.

curl -X GET "http://host:port/semantic-registry/repository/manusquare/properties/suggest?conceptName=ScrewMachine&mostlyUsed=true" -H "accept: application/json" -H "Authorization: secretToken"

Response is a list of property URIs/names with a count of their usage:

```
[
  {
    "property": "http://manusquare.project.eu/core-manusquare#hasAttribute",
    "count": 6
  },
  {
    "property": "http://manusquare.project.eu/core-manusquare#hasCapability",
    "count": 1
  }
]
```

Listing 13 Property suggestion result

Example: get list of values for hasAttribute property of a ScrewMachine

curl -X GET "http:// host:port/semantic-registry/repository/manusquare/properties/vals/suggest?conceptName=ScrewMachine&propertyName=hasAttribute" -H "accept: application/json" -H "Authorization: secretToken"

Response is a list of value used with a count of their usage:

```
[
  {
    "value": "http://manusquare.project.eu/industrial-manusquare#TableLength",
    "count": 1
  },
  {
    "value": "http://manusquare.project.eu/industrial-manusquare#TableWidth",
    "count": 1
  },
  {
    "value": "http://manusquare.project.eu/industrial-manusquare#minSpindleSpeed",
    "count": 1
  },
  {
    "value": "http://manusquare.project.eu/industrial-manusquare#xAxis",
    "count": 1
  },
  {
    "value": "http://manusquare.project.eu/industrial-manusquare#yAxis",
    "count": 1
  },
  {
    "value": "http://manusquare.project.eu/industrial-manusquare#zAxis",
    "count": 1
  }
]
```

Listing 14 Property Value suggestion result

4.3.4.4 Semi-automatic conversion of informal descriptions of resources

Ontology Controller provides an operation that helps to convert a semi-structured textual description of generic resource type into their semantic models aligned with MANUSQUARE ontology (Req#10 from Table 5). POST /repository/manusquare/informal-to-semantic takes an Excel file that contains semi-structured description of a resource

type (e.g. Milling Process) and converts it to RDF semantic model by creating an instance of concept (parameter conceptName) with properties and their values if provided.

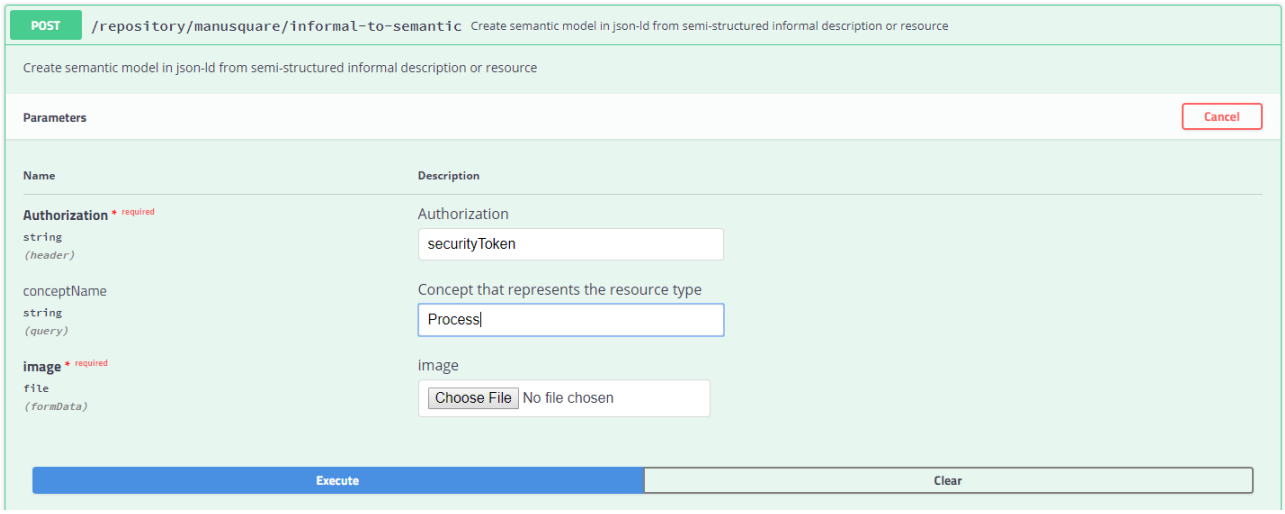


Figure 10 Swagger POST /repository/manusquare/informal-to-semantic

There is an implementation for the conversion of a process template description into the corresponding semantic model. The template is provided through an Excel's spreadsheet that follows a certain structure of rows and columns so that a backend service can process it. Below is an example of such Excel template, where for each process type there is a list of relevant attributes that apply to the process type, and in addition a list of mostly relevant values for the attributes. (Only materials at this time, while values for other attributes are not the part of this specific template, but it can expended in future if needed.)

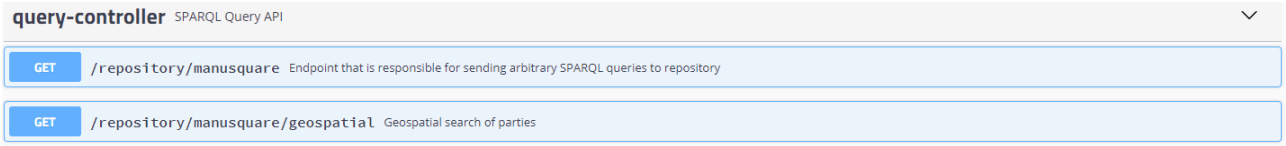
Process	Attribute Keys	Attribute value associated to Material
Milling	Working Area X (mm)	Alloy Steel
	Working Area Y (mm)	Carbon Steel
	Working Area Z (mm)	Cast Iron
	Surface Finishing (µm)	Stainless Steel
	Tolerance (± mm)	Aluminum
	Max Wall Thickness (mm)	Copper
	Axis	Magnesium
	Material	Zinc
		Ceramics
		Composites
		Lead
		Nickel
		Tin
		Titanium
		Elastomer
		Thermoplastics
		Thermosets

Figure 11 Input to informal-to-semantic conversion

A conversion algorithm works on this way. For each term (from spreadsheet illustrated in Figure 11), placed either under Process, Attribute Keys or Materials row, the algorithm looks up for the most suitable concept to capture the term by applying a lexical similarly computation between the term and available concepts in the target taxonomy (ProcessType, AttributeType, or MaterialType taxonomy). The lexical similarity score (0-1 range) is based on two edit distance based algorithms (Jaro-Winkler, Levenshtein) and one token-based (Jaccard) and assumption that a result with the highest similarity score among these three algorithms and if greater than 0.7 is a suitable result. If the suitable concept is found, the term is replaced with an instance of the found concept. That instance is then added into the model according to the target structure, which in this case is, ProcessChain-Process-Attribute-Value structure. Otherwise, if no match found, the term just becomes a new proposed concept that will need to be added in a proper position under the existing taxonomy. The final output is JSON document that contains converted semantic description of given resource(s), and list of unmatched terms (new concepts).

4.3.5 Query Requests Controller

This controller manages requests for execution of arbitrary SPARQL queries upon the semantic repository. It exposes two operations. First GET operation is for execution of any SPARQL Select/Construct query, with or without inferred statements inclusion. Second GET operation is specialized for geospatial search of suppliers.



Here are few example of SPARQL queries that can be executed by the mentioned GET requests.

Find suppliers that have a machine with minSpindleSpeed greater than 7000

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX core: <http://manusquare.project.eu/core-manusquare#>
PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#>
PREFIX : <http://manusquare.project.eu/industrial-manusquare#>
prefix owl:   <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT distinct ?supplier
WHERE { ?processChain core:hasSupplier ?supplier .   ?supplier core:hasName ?name.
       ?processChain core:hasProcess ?process. ?process core:hasResource ?resource .?resource
core:hasAttribute ?attribute .
       ?attribute rdf:type ind:minSpindleSpeed. ?attribute core:hasValue ?value .
    FILTER(xsd:decimal(?value) > 7000)
}
```

Find Suppliers with Grinding capability

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX core: <http://manusquare.project.eu/core-manusquare#>
PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#>
prefix owl:   <http://www.w3.org/2002/07/owl#>

SELECT distinct ?supplier ?capability
WHERE { ?processChain core:hasSupplier ?supplier .   ?supplier core:hasName ?name.
       ?processChain core:hasProcess ?process. ?process core:hasResource ?resource .?resource
core:hasCapability ?capability .
?capability rdf:type ind:Grinding}
```

Listing 15 SPARQL example

Geo-spatial filtering

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX uom: <http://www.opengis.net/def/uom/OGC/1.0/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX core: <http://manusquare.project.eu/core-manusquare#>
SELECT distinct ?supplier ?location ?dist
WHERE {
 ?supplier geo:asWKT ?location .
 ?supplier rdf:type core:Supplier
 BIND((geof:distance(?location, "POINT(51.502497 -0.108915)"^^geo:wktLiteral, uom:metre)/1000) as
?dist)
 FILTER (?dist<10000)
}
```

Listing 16 SPARQL Geospatial example

4.3.6 Tag-Acquisition Controller

There are two sets of GET/POST/DELETE endpoints in this controller.

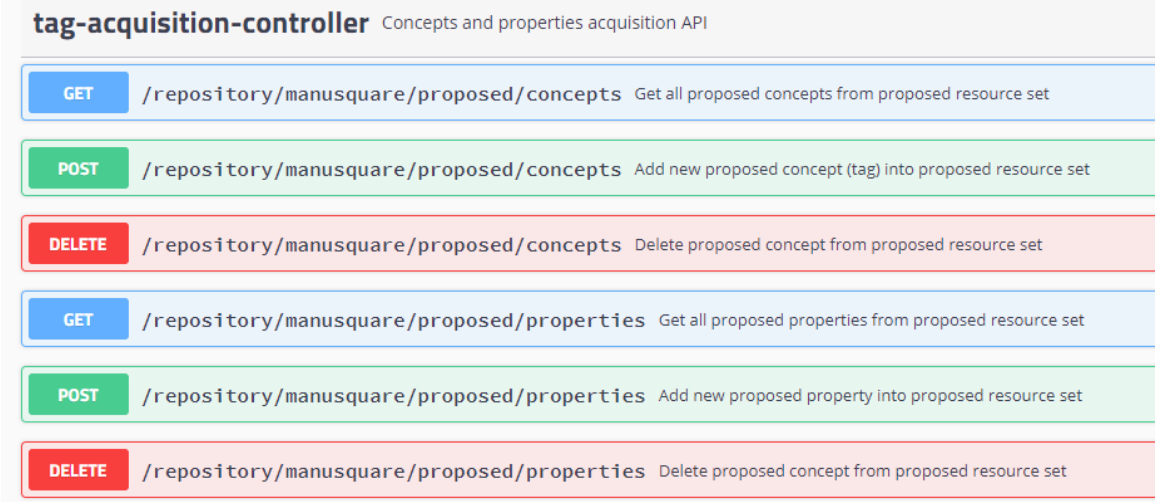
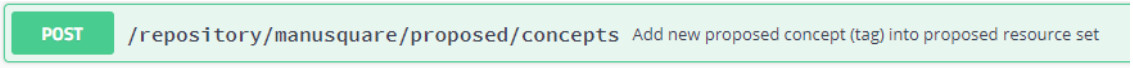


Figure 12 Tag-Acquisition Controller API

The first set is for insertion/removal/retrieval of new suggested concepts, while the other one is for insertion/removal/ retrieval of new suggested properties.

For example, if a user during a MANUSQUARE project creation, wants to assert that the project needs a processing of an Alloy Steel material. But, the Alloy Steel is not contained in a domain-specific taxonomy of material types. However, even in such a case, when the concept is missing, the user should be still given possibility to express the desired need. The platform will not constrain the users in adding new knowledge into the system. New concepts, such as Alloy Steel in this case, should be sent to a tag-acquisition controller that collects new desired entries of concept taxonomies. Afterwards, a proposed concept will be introduced into a domain-specific taxonomy and placed into specific category of concepts. Another example can be a process type Abrasive Machining as a desired capability, but not present as a concept of the Process Types taxonomy. Similarly, new properties can be added into the ontology, if needed.

Example: POST AbrasiveMachining concept into the system using



Request body is

```
[
   {
      "name":"AbrasiveMachining",
      "description":"Abrasive Machining",
      "labels":{
         "en":"Abrasive Machining",
         "it":"Lavorazione Abrasiva",
         "es":"Mecanizado Abrasivo"
      }
   }
]
```

Listing 17 Tag insertion payload

curl -X POST "http://host:port/semantic-registry/repository/manusquare/proposed/concepts" -H "accept: */*" -H "Authorization: secretToken" -H "Content-Type: application/json" -d "[{ \"name\":\"AbrasiveMachining\", \"description\":\"Abrasive Machining\", \"labels\":{ \"en\":\"Abrasive Machining\", \"it\":\"Lavorazione Abrasiva\", \"es\":\"Mecanizado Abrasivo\" } }]"

4.3.7 Inference Rule Controller

Inference rule Controller handles CRUD requests for management of domain-specific inference rules in MANUSQUARE. The table below outlines the available operations, while usage examples are provided in next subsections.

Table 9 Inference Rule Controller Operations

POST /repository/manusquare/load/default	Load default ruleset, if not preloaded
POST /repository/manusquare/spin/add	Add domain-specific inference SPIN rule into the repository
DELETE /repository/manusquare/spin/delete	Delete domain-specific inference SPIN rule for given concept or for given identifier
GET /repository/manusquare/spin/find	Find domain-specific SPIN rules for given concept or for given identifier
GET /repository/manusquare/spin/run/{conceptName}	Forces SPIN execution for all instances of particular class or re-execution of all rules

The domain-specific inference rules are expressed using SPIN (SPARQL Inference Notation) language, which was briefly introduced in the section 3. As said, SPIN allows rule implementation using SPARQL CONSTRUCT or SPARQL UPDATE requests (INSERT/DELETE) and can be used to meet several requirements. For example, to calculate the value of a property based on other properties, to derive new property, to initialize certain values when a resource is first created, or to make additional classification of instance. SPIN rules can be ordered and prioritized to provide incremental reasoning.

Deliverable 2.3 has introduced several types of domain-specific inferences that are seen as valuable for the MANUSQUARE platform. These are:

- **Inferred properties**. Some properties of instances stored in MANUSQUARE' semantic repository are inferred from the asserted statements. Inference is a preferable capability of manufacturing sourcing platforms such as MANUSQUARE, as it may reduce effort for explicitly populating descriptions of suppliers and their manufacturing capabilities and capacities. Some example of this type of inference are:
  - "***knows***" property inference, informally expressed like this: "if supplier A has successfully completed a business process with supplier B, then supplier A knows supplier B, and vice-versa". Obviously, there is no need to assert that supplier 'knows' another supplier, but 'knows' can be automatically inferred by the platform. Listing 6 provides SPIN rule for "knows" inference.
  - "***servesIndustry***" property inference, informally expressed like this: „if supplier A has done business with customer B, and customer B has stated which industry it belongs to, then supplier A serves industry of customer B". Manufacturers provide information about what industries they serve (e.g. Automotive Industry, Office Equipment, Sports Equipment, Aviation Industry, Marine Industry, Kitchen Appliances, Computer Industry, etc). This dynamic part of the profile evolves on a daily basis and inference reduces manual effort to maintain the profile. Inference additional description of a machine with a description coming from another supplier. Listing 18 provides SPIN rule for "knows" inference.

Listing 18 SPIN rule "servesIndustry"

```
@prefix …

core:Quotation spin:rule urn:b1c67f2d-5333-48d6-8b02-41eead54d6fc .
urn:b1c67f2d-5333-48d6-8b02-41eead54d6fc a sp:Construct;
  rdfs:comment "Quotation:ind:b1c67f2d-5333-48d6-8b02-41eead54d6fc";
  rdfs:label "Quotation:ind:b1c67f2d-5333-48d6-8b02-41eead54d6fc";
  sp:text "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX ind:
<http://manusquare.project.eu/industrial-manusquare#> PREFIX core:
<http://manusquare.project.eu/core-manusquare#> construct {?supplier
core:servesIndustry ?industry .} where { ?p rdf:type core:Project. ?p core:hasCustomer
?customer. ?p core:hasQuotation ?q . ?q core:hasSupplier ?supplier . ?customer
core:belongsToIndustry ?industry}" .
```

- **Inferred Capability and Machine Classification.** Selection of appropriate machining capability, process operation, or equipment, if not explicitly stated within an RFQ, can be inferred from a given descriptions and associated inference rules. Or, description of machine tools provided by suppliers can have very few statements and can be uncomplete, due a lack of knowledge or simply, due lack of the time to complete descriptions. Thus, the platform can offer inference rules to classify an equipment, based on its features, attributes, and KPIs. For example:

- If an RFQ requires hole aspect ratio (a hole depth-to-diameter ratio) more than 10, then the qualified supplier should provide Deep Hole Drilling process (DeepHoleDrilling capability) for creating the hole.
- If an RFQ requires the accuracy of 0.0001 or less, then then the qualified supplier must provide a Precision Service capability.
- If an RFQ requires reverse engineering, then the qualified supplier must have CAD capabilities.
- If an instance of machine tool has xAxis, yAxis, and zAxis attributes, is controlled by computer, then a machine tool is classified into 3AxisCNCMachine.
- If a supplier provides cut sheet metals operation with thickness larger than 1 inch, then supplier has a water jet cutting machine

4.3.7.1 Insert new rule

To insert a new inference rule attached to the concept, the POST operation has to be used with two mandatory parameters: conceptName and ruleText.

For example, we can add a rule to concept Item that will classify all instances of Item into Assembly, if there is property hasItems linked to Item instance. conceptName has to be Item, while ruleText can be:

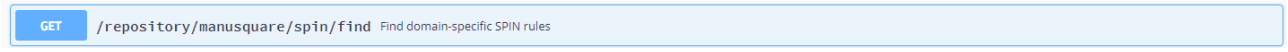
"PREFIX core: <http://manusquare.project.eu/core-manusquare#> PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#> PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> CONSTRUCT { ?input rdf:type core:Assembly . } WHERE { ?this core:hasItem ?input . }"

4.3.7.2 Delete rule

To delete a rule or all rules of particular concept, either rule Id or concept name has to be provided to the DELETE operation.

curl -X DELETE "http://host.port/semantic-registry/repository/manusquare/spin/find?ruleId=9b34f64c-6334-4ed2-8da0-1cdef0fcd0f4" -H "accept: application/json" -H "Authorization: secretToken"

4.3.7.3 Get rule



curl -X GET "http://host:port/semantic-registry/repository/manusquare/spin/find?ruleId=9b34f64c-6334-4ed2-8da0-1cdef0fcd0f4" -H "accept: application/json" -H "Authorization: secretToken "

4.4 Creation of rules by templates and controlled natural language expressions

The Capability and Machine Classification inference examples from the previous subsection, follow certain query and rule patterns, and thus, they can be captured in templates to generalize such query patterns (rule body and rule head) so that rules can be created and reused in a more maintainable way. The Semantic Infrastructure provides support for creation of rules by using the templates. An end-user can by using the Ontology Editor provided by the Semantic Infrastructure, use a controlled natural language to express the rules powered by the template and pattern. The listing below shows two templates and then we show how they are used for creating the rule.

Listing 19 Domain-specific Rule Templates

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#>
PREFIX app: <http://manusquare.project.eu/app-manusquare#>
PREFIX core: <http://manusquare.project.eu/core-manusquare#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

app:template1 a app:RuleTemplate ;
  rdfs:comment 'Desired Capability inference' ;
```

```
  rdfs:label 'Infer ?CapabilityOrService where ?Attribute ?valueExpression;
  app:appliesTo core:Project ;
  app:ruleTemplate '''
        PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#>
        PREFIX : <http://manusquare.project.eu/core-manusquare#>
        CONSTRUCT      {?c      :hasProcessAttribute      ?bnode      .      ?bnode      rdf:type
:ProcessType,:CapabilityType,ind:MachiningService  .  ?bnode  :hasValue  ind:?CapabilityOrService}
where  {  ?c  :hasProcessAttribute  ?attr  .  ?attr  rdf:type  ind:?Attribute  .  ?attr  :hasValue  ?value
BIND(BNODE() AS ?bnode) filter (xsd:decimal(?value) ?valueExpression)}'''  .


app:template2 a app:RuleTemplate ;
    rdfs:comment 'Equipment Classification based on process and attribute' ;
    rdfs:label  'If  supplier  provides  ?Process  with  ?Attribute  ?valueRange,  then  supplier  has
?Machine ' ;
   app:appliesTo core:Process ;
   app:ruleTemplate '''
              PREFIX ind: <http://manusquare.project.eu/industrial-manusquare#>
              PREFIX : <http://manusquare.project.eu/core-manusquare#>
              CONSTRUCT     {?process     :hasResource     ?bnode     .     ?bnode     rdf:type
:Equipment,ind:?EquipmentType . }
where  {?process  rdf:type  :Process,?ProcessType  .  ?process  :hasAttribute  ?attr  .  ?attr  rdf:type
ind:?Attribute  .  ?attr  :hasValue  ?value  BIND(BNODE()  AS  ?bnode)  filter  (xsd:decimal(?value)
?valueExpression)}'''  .
```

By using the template within rdfs:label, a domain expert expresses rules in a control natural language expression, that is further converted into SPIN rule by replacing the placeholders in app:ruleTemplate expression. For example, "Infer **DeepHoleDrilling** where **HoleAspectRatio** greater than 10", or "Infer **PrecisionService** where **Accuracy** less than 0.0001", or "'If supplier provides **CutSheetMetalProcess** with thickness larger than 1, then supplier has **WaterJetCuttingMachine**", are all user-friendly, in a controlled-natural-language-way-made-rules, which are turned into SPIN rules as defined by the template.

4.5 Rule management in Ontology Editor

The rules management is added into the Ontology editor, as illustrated in the Figure below.

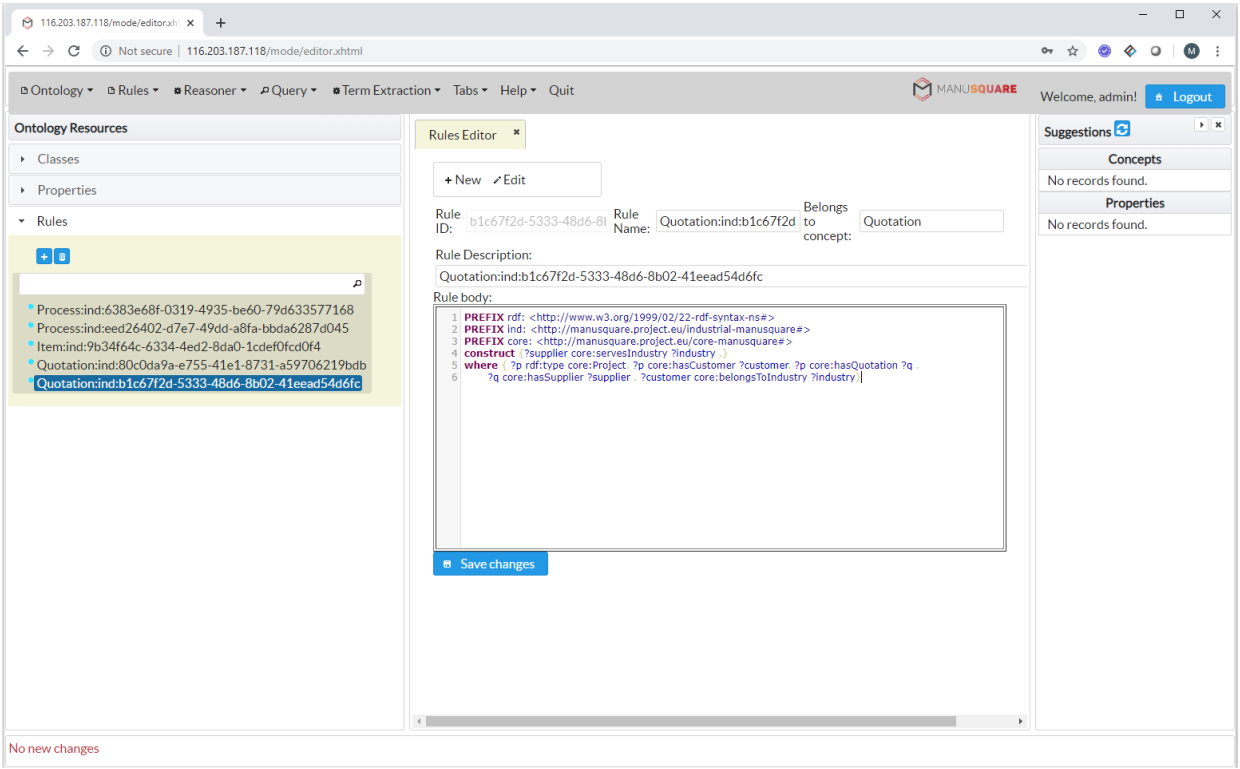
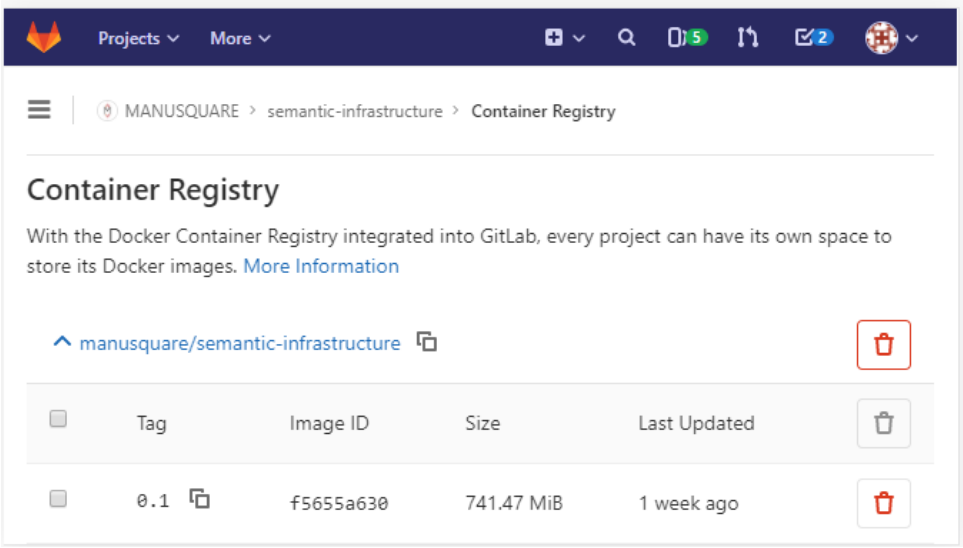


Figure 13 Rule Editor in Ontology Authoring tool

5 DOCKER CONTAINER DEPLOYMENT

Docker (https://www.docker.com/) technology performs operating-system-level virtualization and containerization of applications, and is a very convenient choice to build, distribute, deploy and run an application using containers. The Docker image of the Semantic Infrastructure starts from the tomcat:8.5-jre8-alpine image and deploys with several war files into Tomcat application server. These are semantic-registry.war, mode.war, rdf4j-server.war and rdf4j-workbench.war. In order to preserve logs and data in running Docker container there are two volumes created: VOLUME /var/rdf4j and VOLUME /usr/local/tomcat/log. The volumes can be mounted to a file or directory on the host system, if needed.

A docker image of Semantic Infrastructure are published on the GitLab and can be retrieved from this link https://gitlab.com/manusquare/semantic-infrastructure/container_registry.



Once pulled into a Docker, the Semantic Infrastructure's container can be run by this command: `docker run -d -p 127.0.0.1:8080:8080 -e JAVA_OPTS="-Xms1g -Xmx2g" -e RDF4J_HOST_ADDRESS=http://127.0.0.1:8080 -v data:/var/rdf4j -v logs:/usr/local/tomcat/logs semantic-infrastructure:{tag}*`

After the docker container is placed, access to Semantic Infrastructure Swagger API documentation is available at http://localhost:8080/semantic-registry/swagger-ui.html, while the Ontology Authoring tool Mode is available at http://localhost:8080/mode login with: manusquare/m4nusqu4re

Alternatively, the Semantic Infrastructure can be run through docker-compose that is shown in listing below:

Listing 20 docker-compose.yml

```
version: "3"
services:
  semantic-infrastructure:
    container_name: siminf
    image: semantic-infrastructure:0.2
    restart: on-failure:10
    ports:
      - "8080:8080"
    environment:
#     - RDF4J_HOST_ADDRESS=http://116.203.187.118:8080 - to point to external RD4J server
      - RDF4J_HOST_ADDRESS=http://siminf:8080
      - LOGGING_PATH=/usr/local/tomcat/logs
      - JAVA_OPTS=-Xms1g -Xmx2g
    volumes:
    - data:/var/rdf4j
    - logs:/usr/local/tomcat/logs
volumes:
  data:
  logs:
```

To start docker-compose, use this command `docker-compose up -d`, while to stop it use`docker-compose stop`. If all successfully started, the Semantic Infrastructure will be available on host:8080/semantic-registry and host:8080/mode endpoints.

6 CONCLUSION

This document presented results of the implementation and deployment of MANUSQUARE Semantic Infrastructure of WP2 Task 2.4. The document is accompanied with a working software component that, at Month 24, can be accessed in INNOVA's development server[12] or can be run as a Docker container either on local machine or server machine.

The Semantic Infrastructure provided all needed CRUD (create-read-update-delete) operations, over HTTP POST/PUT,DELETE/GET operations, for the first-order citizens of the platform including stakeholder (supplier) profiles, their process-chains and process descriptions (more precisely, supplier's production and engineering capability and capacity descriptions), challenge/idea descriptions (coming from innovation management toolset), and project (customer needs) descriptions. In addition to that, Semantic Infrastructure provides support for: (1) semantic inferencing on assertions deployed by domain-specific ontologies and descriptions of manufacturing resources (2) semantic querying over the semantic repository, (3) inference rules management, (4) to some extent a semi-automatic support to convert informal descriptions of manufacturing resources into their formal semantic descriptions that are aligned with core- and domain-specific ontologies, (5) web application environment form ontologies management.

At the time of writing this deliverable, the Semantic Infrastructure has been successfully integrated within the platform. It is published as a standalone Docker application which provides services to other components via its RESTful API. It has been integrated with the Ecosystem Data Manager that manages all the data flow between the GUI and tools on one side, and data repositories on another side. Integration with the Matchmaking tool is also successfully accomplished.

Finally, besides the fact that all so far specified requirements have been addressed, implemented and deployed in the current release of the Semantic Infrastructure, there might be additional unforeseen requirements and a need to fine-tune efforts by the end of Integration task (M26) and also during the demonstrations and project pilots. INNOVA plans to support requirements that are crucial for the MVP success and community development.

[12]Links: Swagger documentation http://116.203.187.118/semantic-registry/swagger-ui.html and MODE http://116.203.187.118/mode login with: manusquare/m4nusqu4re