Ecosystem for COllaborative Manufacturing PrOceSses – Intra- and Interfactory Integration and AutomaTION
(Grant Agreement No 723145)

D6.8 Collaborative Manufacturing Services Ontology and Language II

Date: 2019-02-28

Version 1.0

Published by the COMPOSITION Consortium

Dissemination Level: Public

Co-funded by the European Union’s Horizon2020 Framework Programme for Research and Innovation under Grant Agreement No 723145

Document control page

Document file: D6.8 Collaborative manufacturing services ontology and language II-v1
Document version: 1.0
Document owner: CERTH

Work package: WP6–COMPOSITION Collaborative Ecosystem
Task: T6.4 – Collaborative manufacturing services ontology and language
Deliverable type: OTHER

Document status: ☒Approved by the document owner for internal review
☒Approved for submission to the EC

Document history:

Version	Author(s)	Date	Summary of changes made
0.1	Dimosthenis Ioannidis, Alexandros Nizamis (CERTH)	2019-02-05	Revisions on Document Structure and updates related to Architecture
0.2	Alexandros Nizamis, Vakakis Nikolaos (CERTH)	2019-02-11	Updates on Chapters 4, 5 and 6 related to SotA and Ontology
0.3	Mathias Axling (CNET), Giuseppe Pacelli (ISMB)	2019-02-12	Updates on Chapter 4 related to Marketplace and Agents
0.4	Alexandros Nizamis, Vakakis Nikolaos, Alexopoulos Nikolaos (CERTH)	2019-02-19	Updates and input on Chapters 7, 8 and 9 related to Ontology API, Security and Deployment
0.5	Alexandros Nizamis (CERTH)	2019-02-20	Finalization for internal review
0.6	Vakakis Nikolaos, Ntinas Christos (CERTH)	2019-02-27	Updates based on Peer Review Comments
1.0	Alexandros Nizamis (CERTH)	2019-02-28	Final version to be submitted to the European Commission

Internal review history:

Reviewed by	Date	Summary of comments
Alexander Grass (FIT)	2019-02-25	Minor changes and comments
Peter Haigh (TNI-UCC)	2019-02-21	Minor edits and some edits to formatting.

Index:

1. Executive Summary 4

2. Abbreviations and Acronyms 5

3. Introduction 6

3.1 Purpose, Context and Scope of this Deliverable 6

3.2 Content and Structure of this Deliverable 6

4. Collaborative Manufacturing Services Ontology in COMPOSITION Overall Architecture7

4.1 Overview 7

4.2 COMPOSITION Marketplace 7

4.3 Ontology and Agents 8

4.4 Ontology and Rule-based Matchmaker 9

5. State Of The Art Analysis of Ontology Languages, Building Methodologies and Tools10

5.1 Semantic Modelling 10

5.1.1 Definitions 10

5.1.2 Components 10

5.2 Ontology Languages 11

5.2.1 Traditional Ontology Languages 11

5.2.2 Ontology Mark-up Languages 12

5.3 Methodologies for Building Ontologies 13

5.4 Leading Tools for Building Ontologies 14

6. COMPOSITION Collaborative Manufacturing Services Ontology 16

6.1 Imported Ontologies 16

6.1.1 MSDL 16

6.1.2 MASON 17

6.1.3 GoodRelations Language 17

6.2 COMPOSITION Ontology 18

6.2.1 Methodology 18

6.2.2 Ontology Specifications 26

7. COMPOSITION Ontology API 43

7.1 Methodology and Implementation Technologies 43

7.1.1 Introduction to Apache Jena 43

7.1.2 Implementation Details 44

7.2 Supported Interfaces 48

8. COMPOSITION Ontology's Quality Control, Deployment and Security 49

8.1 Quality Control 49

8.1.1 Collaborative Manufacturing Services Ontology 49

8.1.2 Ontology API 49

8.1.3 Scalability of Ontology and Ontology API 51

8.2 Deployment 51

8.3 Ontology and Security Framework 52

9. Conclusions 53

10. List of Figures and Tables 54

10.1 Figures 54

10.2 Tables 54

11. References 56

12. ANNEX I 58

13. ANNEX II 59

1. Executive Summary

The deliverable presents the work has done and the results of the Task 6.4 Collaborative Manufacturing Services Ontology and Language. It aims to describe and analyse the COMPOSITION Ontology, which is delivered as software alongside with this report. In addition, in this report the design of the Ontology API, which is part of the complete semantic framework of the project, is described. The ontology framework design is driven by COMPOSITION project use cases, requirements, WP6 activities related to Marketplace and connections with other WPs.

COMPOSITION Collaborative Ecosystem needs a knowledge base in order to support flexible specification and execution of manufacturing collaboration schemes. The knowledge base should enable the description of supply and demand entities participating in the Ecosystem as well as the description of manufacturing services' capabilities and resources for participating entities. In order to cover these needs a Collaborative Manufacturing Services Ontology is adopted and will be used as a common vocabulary to offer interoperability and representation of both meanings and data.

As the knowledge store keeps information about business entities and their services, the Ecosystem Agents are able to communicate and make transactions based on this common information. An agent who requests a service or a product will be able to find matching agents who support this service or product based on the information of COMPOSITION Marketplace Ontology. Moreover, the Marketplace will be able to match possible solutions or services providers by inferring new knowledge from the Ontology store and the Matchmaking mechanisms from Task 6.5.

This document updates the D6.7 Collaborative manufacturing services ontology and language I. The structure of the D6.7 was adopted also for this document. All the information from previous version, which is still up-to-date or they are related to methodologies and tools that we have built the ontology were kept in this version as well, in order to provide to the reader a complete picture of Task 6.4. In this updated version, information related to Ontology's extension is added. The updates are new classes, properties and individual and they are integrated to first version's descriptions in order to create a coherent document that documents in detail the ontology. Moreover, the design of the Ontology API has been updated. In contrast, with the first version, in the current one the API is part of the Matchmaker package and not a standalone application. The catalogue of the supported web services of the API has been extended as well. Furthermore, both Ontology and Ontology API have been deployed as a Docker[1] container. As introduced in this version, they are also integrated with the Marketplace Agents. The Agents are able to call services from Ontology API and modify the Ontology. Their communication with the Agents is secured as it is based on services offered by the project's Security Framework. The connection of Ontology with the Security Framework is introduced in this report as well.

Besides purpose, context, and scope the first part of this document is devoted to the content and structure of this Deliverable. The next parts describe both general information about Ontologies as well as specific information about COMPOSITON's Ontology. The general information is a state-of-the-art analysis of Ontology languages, methodologies and tools. The COMPOSITION specific parts describe in details the Collaborative Manufacturing Services Ontology and its implementation process. Furthermore, the COMPOSITION Ontology API which has been developed for the purposes of this project is described. Details about the usage of the delivered Ontology and conclusions of Task 6.4 are also provided.

[1] https://www.docker.com/

2. Abbreviations and Acronyms

Table 1: Abbreviations and acronyms are used in this deliverable

Acronym	Meaning
API	Application Programming Interface
BGW	Border Gateway
CVS	Concurrent Versions System
CXL	COMPOSITION eXchange Language
DAML	DARPA Agent Markup Language
DoA	Description of Action
FLogic	Frame Logic
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
KIF	Knowledge Interchange Format
MASON	Manufacturing's Semantics Ontology
MSDL	Manufacturing Service Description Language
OCML	Operational Conceptual Modeling Language
OIL	Ontology Interchange Language/Ontology Inference Layer
ORSD	Ontology Requirements Specification Document
OKBC	Open Knowledge Base Connectivity
OSF	Open Semantic Framework
OWL	Web Ontology Language
PAL	Pedagogic Algorithmic Language
RDF	Resource Description Framework
RDFa	Resource Description Framework in Attributes
RDFS	Resource Description Framework Schema
RDQL	RDF Data Query Language
SMEs	Small and medium-sized enterprises
SPARQL	Simple Protocol and RDF Query Language
URI	Uniform Resource Identifier
VM	Virtual Machine
WP	Working Package
XML	eXtensible Markup Language
XOL	XML-based Ontology Language

3. Introduction

3.1 Purpose, Context and Scope of this Deliverable

The purpose of Task 6.4 Collaborative Manufacturing Services Ontology and Language and its corresponding deliverables is the development of an Ontology framework as a part of COMPOSITION's Agent Marketplace. The scope of this deliverable is to describe the work that has been done for Task 6.4 and to present the Collaborative Manufacturing Services Ontology. It further describes the release of an API, which offers services for the manipulation of the Ontology, and it is part of the complete COMPOSITION semantic framework alongside with the Ontology and the Matchmaker.

The current version of Ontology contains classes, properties and instances related to use cases, the manufacturing domain, the supply-chain domain and software solutions for manufacturing ecosystems. Some extensions in the current version of Ontology are possible and after the end of this task as the task is strongly correlated with the Task 6.5 – Matchmaker which will be active for further 4 months. Prior to reaching M14 and the first version of this deliverable the focus of Task 6.4 was the research in the Ontology field and the creation of a first version of Collaborative Manufacturing Services Ontology based on well-known manufacturing and e-commerce domain ontologies. Furthermore, technologies and APIs related to ontologies were examined and in the context of COMPOSITION, the most suited were used in the software design. In the next period and until the second iteration of the document (M30) further research conducted for the Ontology extension. Furthermore, more individuals were created based on pilots' needs and so a more stable version of the Ontology was created, the Ontology API was extended, and both Ontology and the corresponding API was deployed as Docker images (part of semantic framework) in order to be connected with the rest Marketplace components such as the Marketplace Agents.

3.2 Content and Structure of this Deliverable

In this deliverable the COMPOSITION's Collaborative Manufacturing Services Ontology and Language is presented. The COMPOSITION's Ontology API and its supported services are described too. In order to properly describe the specifications of the Ontology components we decided to include the following basic sections in this report:

Section 4 describes the integration of the Ontology component with the overall COMPOSITION architecture and its interaction with other COMPOSITION components. Special attention is given to interactions with the Marketplace Agents and the Matchmaker.

Section 5 includes a brief state-of-the-art analysis in the field of Ontologies and Semantic Modelling. Ontology languages, methodologies and leading tools for building ontologies are presented.

Section 6 contains two main parts. In the first part the ontologies which are imported at COMPOSITION's Collaborative Manufacturing Services Ontology are analyzed. In the second part the current version of Collaborative Manufacturing Services Ontology is presented and analysed.

Section 7 is about the COMPOSITION's Ontology API. Implementation and current supported interfaces are presented.

Section 8 contains the quality plan, the deployment and the connection with the Security Framework of the project.

Section 9 outlines the conclusions of Task 6.4 and sums up this deliverable's outcomes.

4. Collaborative Manufacturing Services Ontology in COMPOSITION Overall Architecture

This section describes the position of Collaborative Manufacturing Services Ontology and the position of the Ontology API in COMPOSITION project. The main interactions of the previous two components with the rest of the project's components are described too. Also we present a short description of the Marketplace in order to be clearer about the Ontology's location and usage.

4.1 Overview

Task 6.4 Collaborative Manufacturing Services Ontology and Language and its corresponding software deliverables are part of WP6 COMPOSITION Collaborative Ecosystem. The implemented Ontology is a core part of Collaborative Ecosystem/ Marketplace, as it constitutes the ecosystem's knowledge base.

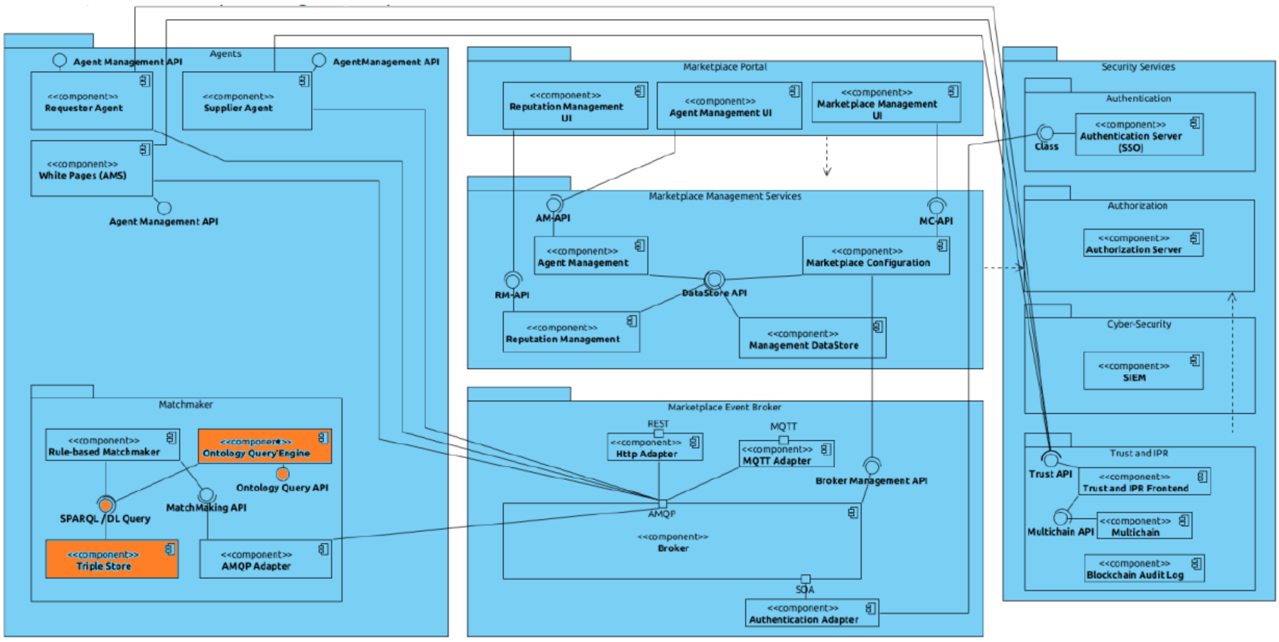

Figure 1: COMPOSITION Marketplace components

As depicted in Figure 1, the Collaborative Manufacturing Services Ontology and the Ontology API belong to the Agents framework. More precisely, they are parts of the Matchmaker package. Collaborative Manufacturing Services Ontology initializes the Triple Store component and the Ontology API is the Ontology Query API interface of the Query Engine. The Rule-based Matchmaker component uses the Ontology, which is stored in the Ontology/Triple Store and after that; it applies rules in order to infer new knowledge from the Ontology. Moreover, the Marketplace's Agents are able to use the Ontology API and Ontology store via Broker and AMQP Adapter components in order to read or store data.

4.2 COMPOSITION Marketplace

Modern manufacturing does not only involve the processes of a single factory, but an intricate network of suppliers, sub-manufacturers and service providers connected in global supply chains. As stated in Strategic Objective 1 (COMPOSITION, 2016), COMPOSITION will provide a digital automation framework for optimizing the value chain; the production processes of the single factory. The goal outlined in Strategic Objective 2 (COMPOSITION, 2016), is to extend the single factory information management system to support a flexible network of connected and interoperable factories in a collaboration ecosystem. Innovative services and practices enabled by this ecosystem could optimize manufacturing and logistics processes and lead to faster production cycles, increased productivity, less waste and more sustainable production. The COMPOSITION Marketplace corresponds to the "Business" IT Layer and "Connected World" Hierarchy Level of the RAMI 4.0 Reference Architecture.

The COMPOSITION collaborative ecosystem will be realized through an interoperable agent-based marketplace where the stakeholders are represented by agents that can exchange information, negotiate deals and find new collaboration opportunities and models. Instead of custom-built, ad-hoc integrations with suppliers or sub-contractors, the goal of the agent-based marketplace is to provide automation of co-ordination, negotiation and data sharing. There will be human intervention and supervision built in, but the degree of autonomy of the agents will be sufficient to find and negotiate with previously unknown parties. The definition of such a marketplace is simply that it is a set of intelligent agents interacting using a common vocabulary through the same shared Broker, using the same shared platform services, i.e. Security Services, Management Services, Matchmaker etc. (Figure 1 COMPOSITION Marketplace components).

Three distinct types of marketplaces have been identified: Open Marketplaces, Closed Marketplaces and Virtual Marketplaces. These provide support for varying degrees of exclusivity in the configuration of a marketplace, which has been identified in the requirements as a major factor in acceptance and adoption of such a system.

An Open Marketplace is open to any stakeholder with valid COMPOSITION credentials; anyone who has acquired valid credentials may enter their offers and requests and collaborate with any other stakeholder. There may be several open marketplaces, primarily organized by the type of supply chain that is supported. A stakeholder may participate in several marketplaces.

A Closed Marketplace is owned - and likely also operated - by one stakeholder and open only to a trusted subset of other COMPOSITION stakeholders. It is a physically separate infrastructure from the Open Marketplace, hosted as a separate platform with its own set of services and components. The Closed Marketplace may be public, allowing join requests by agents in the Open Marketplace, or private, with membership allowed by invitation only.

A Virtual Marketplace is a closed group of agents in the Open Marketplace that have chosen to collaborate exclusively in the context of one or several negotiations. The Virtual Marketplace may exist only for a single negotiation or be persistent over several negotiations, e.g. to support a specific business process or a specially trusted group based on a formalized reputation and trust model.

D9.9 “Sustainable Business Models for IIMS in Manufacturing Industries” describes the evaluation of the COMPOSITION Marketplace from a business perspective. A digital marketplace product (or virtual or online marketplace) is a type of e-commerce site where product or service information is provided by multiple third parties. Transactions are processed by the marketplace operator and then delivered and fulfilled by the participating suppliers or wholesalers. (The classes, properties and instances in the domain of each business model the marketplace platform is applied to are described by the Collaborative Manufacturing Services Ontology.) Business models and value generation for three aspects of the COMPOSITION marketplace were evaluated in D9.9: Waste Management Marketplace, Software Virtual Marketplace and Supply Chain Marketplace. The model showed a positive net cashflow for all actors in all three cases. The final pricing models and revenue streams for the COMPOSITION collaborative ecosystem will be selected and presented in D9.11 “Final Exploitation Strategy and Business Plans”.

4.3 Ontology and Agents

Agents are implemented and operated by different organizations, in general different from the bodies operating the COMPOSITION Marketplace or specifying the Collaborative Manufacturing Services Ontology. Nevertheless, Agent’s core behaviour and internal aspects must necessarily reflect the classes, functions and attributes defined in the common ontology, so to enable interoperable behaviour.

Due to the “open” and potentially evolving nature of the marketplace, suitable techniques must be applied to ensure that the agent’s implementation and the data models linked with the Ontology remain aligned.

To do so, and for agents to have a fully-transparent communication with the Matchmaker and keeping up with the evolving ontologies, a proxy-like service has been implemented in the Agent Management System (AMS). Keeping the complexity of interactions in the AMS allows the definition of a common protocol and data format with the stakeholder agents who no longer need to care about adapting to the evolving ontologies.

Agents contact the AMS in order to request the Matchmaker services through a simple JSON, in order to:

- Request the list of the suitable agents for a certain negotiation, e.g. the agents offering a certain service on the marketplace
- Evaluate the offers that have been received during a negotiation

A more detailed description about agents and their interaction is available in deliverables:

- D6.5: Connectors for Inter-factory Interoperability and Logistics I

- D6.3: COMPOSITION marketplace I

Further updates will be provided in D6.4: COMPOSITION Marketplace II.

4.4 Ontology and Rule-based Matchmaker

COMPOSITION's Rule-based Matchmaker and Collaborative Manufacturing Services Ontology are two extremely connected components. The Matchmaker is strongly correlated with the Collaborative Manufacturing Services Ontology and its functionality depends exclusively on the Ontology store.

Ruled-based Matchmaker's main goal is to match Agents' offers and requests. Matchmaker supports semantic matching in terms of manufacturing capabilities, in order to find the best possible supplier to fulfill a request for a service, raw materials or products involved in the supply chain. Matchmaker considers different decision criteria for supplier selection according to several qualitative and quantitative factors.

In order to be able to perform matching, the Ruled-based Matchmaker infers new knowledge by applying semantic rules in the knowledge stored into the Collaborative Manufacturing Services Ontology. By applying this set of rules, the Matchmaker is able to extract useful conclusions from ontology and connect Agents which are not explicitly connected. The matchmaking process will be analysed in more details at D6.9 and D6.10 deliverables. Also, some details about matchmaking process and Ontology's usage will be mentioned at Section 6 from the current report.

5. State Of The Art Analysis of Ontology Languages, Building Methodologies and Tools

This section is a thorough analysis of Ontology field, languages, building methodologies and tools.

5.1 Semantic Modelling

In a general sense, semantics is the study of meanings of the message behind the words. “Semantic” in the context of data means “from the user’s perspective”. It is the data in context-where the meaning is. Information is also often defined as the data in context. Semantic therefore, while not synonymous with information, carries with it the same sense of data at work, or data in the worker’s hands. The semantic data model is a method of structuring data in order to represent it in a specific logical way. It is a conceptual data model that includes semantic information that adds a basic meaning to the data and the relationships that lie between them. This approach to data modelling and data organization contributes to easy development of application programs and also easy maintenance of data consistency when data is updated.

5.1.1 Definitions

In computer and information science, ontology is a technical term denoting an artefact that is designed to enable the modelling of knowledge. One of the most well-known definitions was presented by Studer and colleagues [Studer et al., 1998]: “An ontology is a formal, explicit specification of a shared conceptualization”. The definition explains the ontology as an approach of an abstract model of some incident in the world with relevant concepts of that incident. Concepts and constrains are defined in an accurate way. The ontology should be machine-readable as well as generally accepted.

Ontology can be viewed as a level of abstraction of data models intended for modelling knowledge about individuals, their properties, and their association to other individuals. Ontologies are typically specified in languages that allow abstraction away from data structures and implementation strategies. In practice, the languages of ontologies are closer in expressive power to first-order logic than languages used to model databases. For this reason, ontologies are said to be at the "semantic" level, whereas database schemas are models of data at the "logical" or "physical" level.

A strong advantage regarding ontologies is that they are independent from lower level data models and used for integrating heterogeneous databases, enabling interoperability among disparate systems, and specifying interfaces to independent, knowledge-based services. In the technology stack of the Semantic Web standards, ontologies are called out as a definitive layer. A multitude of standard languages and a variety of tools have been built for creating and working with ontologies.

5.1.2 Components

Gruber (Gruber, 1993a) proposed modelling ontologies using frames and first order logic. He identified five kinds of components: classes, relations, functions, formal axioms and instances.

Classes represent concepts, which are taken in a broad sense. For instance, in the traveling domain, concepts are: locations (cities, villages, etc.), lodgings (hotels, camping, etc.) and means of transport (planes, trains, cars, ferries, motorbikes and ships). Classes in the ontology are usually organized in taxonomies through which inheritance mechanisms can be applied. We can represent a taxonomy of entertainment places (theater, cinema, concert, etc.) or travel packages (economy travel, business travel, etc.). Classes can represent abstract concepts (intentions, beliefs, feelings, etc.) or specific concepts (people, computers, tables, etc.).

Relations represent a type of connection between concepts of the domain. They are formally defined as any subset of a product of n sets, that is: $R \subset C1 \times C2 \times ... \times Cn$. Ontologies usually contain binary relations. The first argument is known as the domain of the relation, and the second argument is the range. For instance, the binary relation Subclass-Of is used for building the class taxonomy. Examples of classifications are: a Four-Star-Hotel is a subclass of a Hotel, a Hotel is a subclass of Lodging, and a Flight is a subclass of Travel, which is identified by the flight-number.

Functions are a special case of relations in which the n-th element of the relation is unique for the n-1 preceding elements. This is usually expressed as: $F: C1 \times C2 \times ... \times Cn\text{-}1 \Rightarrow Cn$. An example of a function is Pays, which obtains the price of a room after applying a discount. The lambda-body expression on the definition is written in KIF and denotes the value of the function in terms of its arguments.

Formal axioms are a priori assertions always assumed to be true. They are normally used to represent knowledge that cannot be formally defined by the other components. In addition, formal axioms are used to verify the consistency of the ontology itself or the consistency of the knowledge stored in a knowledge base. Formal axioms are very useful to infer new knowledge. An axiom in the traveling domain would be that it is not possible to travel from the USA to Europe by train.

Instances are used to represent elements or individuals in ontology. They form the ground or atomic level of the ontology. An example of instance of the traveling domain concept is the flight that arrives at Seattle on February 8, 2002 and costs 300 (US Dollars, Euros, or any other currency).

5.2 Ontology Languages

Ontology languages are formal languages used to construct ontologies. They allow the encoding of knowledge about specific domains and often include reasoning rules that support the processing of that knowledge. The Selection of an Ontology Language is one of the key decisions to take in the ontology development process. There are many ontology implementation languages and general Knowledge Representation (KR) languages and systems that have been used for implementing ontologies. One must firstly decide what is needed regarding expressiveness and reasoning in order to come to a conclusion about which languages satisfy these requirements.

There are several steps in the implementation of different ontology components in a language taking into account the Knowledge Representation modelling underlying the language. The first step is to describe how concepts are built and then how concept attributes are defined. Usually there are two kinds of attributes distinguished: instance attributes which describe concept instances and can take their values in those instances and class attributes which describe the concept and take their values in it. Next step is the attribute constraint specification and then the creation of concept taxonomies.

Relations are very important components in ontology modelling as they describe the relationships that can be established between concepts, and consequently, between the instances of those concepts. Depending on the language, relations should be given different names. Afterwards, functions are described, in case they can be defined in the language. In many languages, functions are usually defined as special cases of relations.

Upcoming is the definition of formal axioms. Formal axioms can appear embedded in other ontology definitions or as independent definitions in the ontology. Next, instances are included along with comments about how they can be created, how their attribute values can be filled and how a relation that holds between instances can be represented in the language. Finally, other components that can be expressed in the language, such as rules, procedures, ontology mappings, are presented. The remainder of this chapter examines specific languages that are used in ontology modelling.

5.2.1 Traditional Ontology Languages

Ontolingua and KIF

Ontoligua is an ontology language based on KIF (Genesereth and Fikes, 1992; NCITS, 1998) and on the Frame Ontology (Gruber, 1993a). KIF (Knowledge Interchange Format) development was designed to solve the problem of language heterogeneity in knowledge representation, and to allow the interchange of knowledge between diverse information systems. KIF is a prefix notation of first order predicate calculus with some extensions. It permits the definition of objects, functions and relations with functional terms and equality. KIF has declarative semantics and permits the representation of meta-knowledge, reifying functions and relations, and non-monotonic reasoning rules.

LOOM

LOOM (MacGregor, 1991; LOOM tutorial, 1995) was being developed by the Information Science Institute (ISI) of Southern California University. LOOM was not exactly built as a language for implementing ontologies but as an environment for the construction of general-purpose expert systems and other intelligent applications. LOOM is based on the description logics (DL) paradigm and is composed of the “description” and the “assertional” sublanguages.

OKBC

OKBC (Chaudhri et al., 1998) is the acronym for Open Knowledge Base Connectivity. The objective of KBC was to create a frame-based protocol to access knowledge bases stored in different knowledge representation systems.

OCML

OCML (Motta, 1999) stands for Operational Conceptual Modeling Language. One of several pragmatic considerations that were taken into account in its development was its compatibility with Ontolingua. OCML can be considered as a kind of “operational Ontolingua” that provides theorem proving and function evaluation facilities for its constructs.

FLogic

FLogic (Kifer et al., 1995) is the acronym of Frame Logic. FLogic was initially developed as an object oriented approach to first order logic. It was specially used for deductive and object-oriented databases, and was later adapted and used for implementing ontologies. FLogic integrates features from object-oriented programming, frame-based KR languages and first order logic.

5.2.2 Ontology Mark-up Languages

SHOE

SHOE (Luke and Heflin, 2000) stands for Simple HTML Ontology Extension. SHOE was created as an extension of HTML with the aim of incorporating machine-readable semantic knowledge in Web documents. It provides specific tags for representing ontologies. As these tags are not defined in HTML, the information inside them is not shown in standard Web browsers. There is also a slight variant of the SHOE syntax for XML compatibility.

XOL

XOL (Karp et al., 1999) stands for XML-based Ontology exchange Language. The purpose of this language was to provide a format for exchanging ontology definitions among a heterogeneous set of software systems. Therefore, XOL was not intended for developing ontologies, it was created as an intermediate language for transferring ontologies among different database systems, ontology-development tools, and application programs.

RDF and RDF Schema

RDF (Lassila and Swick, 1999) stands for Resource Description Framework. It is being developed by the W3C to create metadata for describing Web resources, and it has been already proposed as a W3C recommendation. The RDF data model is equivalent to the semantic networks formalism and consists of three object types: resources, properties and statements.

The RDF data model does not have mechanisms for defining the relationships between properties and resources. This is the role of the RDF Vocabulary Description language (Brickley and Guha, 2003), also known as RDF Schema or RDFS. RDF(S) is the term commonly used to refer to the combination of RDF and RDFS. Thus, RDF(S) combines semantic networks with frames but it does not provide all the primitives that are usually found in frame-based knowledge representation systems. In fact, neither RDF, nor RDFS, and nor their combination in RDF(S) should be considered as ontology languages per se, but rather as general languages for describing metadata in the Web. RDF(S) is widely used as a representation format in many tools and projects, and there exists a huge amount of resources for RDF(S) handling, such as browsing, editing, validating, querying, storing, etc. In the section about further readings, we provide several URLs where updated information about RDF(S) resources can be found.

OIL

OIL (Horrocks et al., 2000; Fensel et al., 2001) stands for Ontology Interchange Language and Ontology Inference Layer. Like the other languages previously presented, for example, SHOE and RDF(S), OIL was built to express the semantics of Web resources. OIL was superseded by DAML+OIL, however, software is still available to manage and reason with OIL ontologies.

DAML+OIL

DAML+OIL (Horrocks and van Harmelen, 2001) was developed by a joint committee from the USA and the European Union (mainly OIL developers) in the context of the DARPA project DAML (DARPA Agent Markup Language). The main purpose of this language is to allow semantic markup of Web resources.

OWL

OWL (Dean and Schreiber, 2003) is the result of the work of the W3C Web Ontology (WebOnt) Working Group, which was formed in November 2001. This language derives from and supersedes DAML+OIL. It covers most of DAML+OIL features and renames most of its primitives. As the previous languages, OWL is intended for publishing and sharing ontologies in the Web.

OWL 2

The OWL 2 Web Ontology Language or OWL 2 is an ontology language for the Semantic Web with formally defined meaning. OWL 2 is an extension and revision of the OWL Web Ontology Language developed by the W3C Web Ontology Working Group. OWL 2 has similar structure with OWL 1 and offers new features such as richer datatypes, data ranges, keys, property chains, cardinality restrictions etc. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. The ontologies were written on OWL 2 can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents.

SPARQL

Even if it is not an ontology language, SPARQL (E. Prud'hommeaux et al, 2008) is mentioned here because it supports querying the previous languages. SPARQL allows performing queries over RDF data and, since both RDF-S and OWL are based in RDF, also over RDF-S and OWL ontologies. SPARQL can be used to express queries across diverse data sources and its syntax is similar to SQL to facilitate its adoption.

Query in the Semantic Web context means technologies and protocols that can programmatically retrieve information from the Web of Data. RDF provides the foundation for publishing and linking data, allowing many technologies to embed data in documents, such as RDFa, or expose what is stored in databases, or make it available as RDF files.

The SPARQL has been designed to send queries and receive results, e.g. through HTTP or SOAP, within the Semantic Web, which is typically represented using RDF as a data format. This query language is based on (triples) patterns that are similar to RDF triples, and the results of a SPARQL query will be the resources for all triples that match those patterns. Thus, it provides a powerful tool that allows extracting complex information (i.e., existing resource references and their relationships) and present them in different friendly format (i.e. tables).

5.3 Methodologies for Building Ontologies

The goal of this section is to present the foremost methodologies used to build ontologies. The methodologies that will be presented are METHONTOLOGY, On-To-Knowledge, DILIGENT and the most recently developed, NeOn methodology.

METHONTOLOGY

This methodology was developed within the Ontology group at Universidad Politécnica de Madrid. It enables the construction of ontologies at the knowledge level. METHONTOLOGY has its roots in the main activities identified by the software development process (IEEE, 1996) and in knowledge engineering methodologies (Gómez-Pérez et al., 1997; Waterman, 1986). This methodology includes the identification of the ontology development process, a life cycle based on evolving prototypes, and techniques to carry out each activity in the management, development-oriented, and support activities.

On-To-Knowledge

The aim of the On-To-Knowledge project (Staab et al., 2001) is to apply ontologies to electronically available information for improving the quality of knowledge management in large and distributed organizations. A methodology and tools were developed for intelligent access to large volumes of semi-structured and textual information sources in intra-, extra-, and internet-based environments.

The methodology includes a structure for building ontologies to be used by the knowledge management application. Therefore, the On-To-Knowledge methodology for building ontologies proposes to build the ontology taking into account how the ontology will be used in further applications. Consequently, ontologies developed are highly dependent of the application. Another important characteristic is that On-To-Knowledge proposes ontology learning for reducing the efforts made to develop the ontology. The methodology also includes the identification of goals to be achieved by knowledge management tools, and is based on an analysis of usage scenarios (Staab et al., 2001). On-To-Knowledge is considered as a methodology because it has a set of techniques, methods, principles for each of its processes, and because it indicates the relationships between such processes.

DILIGENT

DILIGENT is a methodology, which is intended to support domain experts in a distributed setting to engineer and evolve ontologies. It comprises five main activities: build, local adaptation, analysis, revision and local update. The process starts by having domain experts, users, knowledge engineers, and ontology engineers build an initial ontology. DILIGENT focuses on distributed ontology development involving different stakeholders, who have different purposes and needs and who usually are not at the same location. Moreover, we do not require completeness of the initial shared ontology with respect to the domain. DILIGENT is not constrained to a certain ontology formalism or language. The methodology covers the whole range of possible ontologies, starting with simple taxonomies, vocabularies and topic hierarchies (represented as instances of topic ontology) up to foundational ontologies with many axioms.

NeOn

NeOn aims to advance the state of the art in using ontologies for large-scale semantic applications in the distributed organizations. Particularly, the methodology improves the capability to handle multiple networked ontologies that exist in a particular context, are created collaboratively, and might be highly dynamic and constantly evolving. It is a scenario-based methodology that supports different aspects of the ontology development process, as well as the reuse and dynamic evolution of networked ontologies in distributed environments, where knowledge is introduced by different people (domain experts, ontology practitioners) at different stages of the ontology development process. This methodology has been used to build ontology networks in different domains and areas and by people with diverse background.

5.4 Leading Tools for Building Ontologies

In order to ease the task of building ontologies and implementing them in ontology languages, a lot of tools and building environments were created. There are interfaces that help users in the ontology development process by performing some of the main activities, such as conceptualization, implementation, consistency checking and documentation. An overview of the new generation ontology engineering environments is presented hereafter.

Protégé

Protégé is an open platform oriented to the task of ontology and knowledge-based development. It is an open source, standalone application (also available on-line through Web Protégé), with an extensible architecture. The core of this environment is the ontology editor, and it holds a library of modules that can be plugged, called plug-ins, to add more functions to the environment.

Protégé knowledge model is based on frames and first order logic. The main modelling components of protégé are classes, slots, facets and instances. Classes are organized in class hierarchies where multiple inheritances is permitted and slots can also be organized in slot hierarchies. The knowledge model allows expressing constraints in the PAL language, which is a subset of KIF, and allows expressing metaclasses, which are classes whose instances are also classes. Classes can be concrete or abstract. The former may have direct instances while the latter cannot have them; instances of the class must be defined as instances of any of its subclasses in the class taxonomy.

In terms of interoperability, once an ontology have been created in Protégé, there are many ways to access Protégé ontologies from ontology-based applications. All the ontology terms can be accessed with the Protégé Java API. Hence it is easy for ontology-based applications to access ontologies as well as use other functions provided by different plug-ins.

Open Semantic Framework

Open Semantic Framework (OSF) is an integrated software stack using semantic technologies for knowledge management. It has a layered architecture that combines existing open source software with additional open source components. OSF is designed as an integrated content platform accessible via the Web, which provides needed knowledge management capabilities to enterprises.

The OSF framework is made operational via ontologies that capture the domain or knowledge space, matched with internal ontologies that guide OSF operations and data display. This design approach is known as ODapps, for ontology-driven applications. Ontologies are, in essence, graph structures. Graphs are among the most ubiquitous models of both natural and human-made systems. They can be used to model many types of relations and process dynamics multiple systems. Any problem of practical interest may be represented by a graph. They are especially well suited to capture and manage knowledge domains.

Anzo

Anzo is software based on Semantic Web Technologies for data management and advanced analytics. The Anzo software can be used for data integration, search, analysis, visualization, and interaction. The collection of Anzo modules is also well-suited to building agile, real-time applications that integrate with varied data sources, and allow for easy customization and evolution as business environments change providing significant end-user self-service.

There are three products in the Anzo suite. The first, Anzo Data Collaboration Server is a semantic-standards-compliant environment for connecting systems and storing/accessing data. Second is Anzo on the Web, a Web visualization tool with which non-technical users can create mashed-up views of any data accessible through the Anzo Data Collaboration Server. Anzo on the Web supports semantic lenses that match themselves with data, automatically providing appropriate views to users depending on the type of data they are working with. Last is Anzo for Excel, a plug-in for MS Excel that enables Excel spreadsheets to be mapped to an ontology and the data within the spreadsheets to be stored as RDF in the Anzo Server. All of the Anzo software products leverage semantic standards including RDF, SPARQL, RDFS, OWL, and RDFa.

Vocol

Vocol (L. Halilaj, et al., 2016) is an Integrated Environment for Collaborative Vocabulary Development. Vocol environment tool supports the basic activities such as modeling, population and testing during vocabulary development. The tool is based on agile software and content development methodologies and it is available on the following Github[2] repository: https://github.com/vocol/vocol. Vocol supports ontology development, syntactic and semantic errors' checking, generation of documentation and visualization, a query engine(supports SPARQL) and a version control system.

Mobi

Mobi[3] is a free and open source tool that links data sources to knowledge graphs. The tool offers a web-based collaborative environment for teams to create, share, and evolve data models together. Mobi requires Java 8 and it is built with Apache Karaf[4] and utilizes OWL 2 for authoring ontologies. Mobi offers ontology managing and versioning. It supports SPARQL as query language and handling graph data modelled using the Resource Description Framework. Mobi is also available as a Docker image on Docker Hub.

[2] https://github.com/
[3] https://mobi.inovexcorp.com/
[4] http://karaf.apache.org/

6. COMPOSITION Collaborative Manufacturing Services Ontology

This section consists of two sub-sections. The first one is a brief analysis of the well-known ontologies in manufacturing and e-commerce domains, which selected and imported to COMPOSITION's Collaborative Manufacturing Services Ontology. The second part is a thorough analysis of the design of COMPOSITION's Collaborative Manufacturing Services Ontology. Both the methodology has been followed for Ontology's development and the Ontology's specifications are analysed. Furthermore, the COMPOSITION's Ontology has further extended and re-engineered after the import of existing ontologies in order to meet the requirements of the project. This process is described in this chapter as well.

6.1 Imported Ontologies

The manufacturing domain should be supported as the COMPOSITION Ontology should be able to represent manufacturing services and resources. For this reason, the hereinafter presented ontologies MSDL (Ameri, 2006) and MASON (Lemaignan, 2006), are imported to the COMPOSITION Ontology as they are manufacturing domain specific and they offer a large variety of classes and properties about this domain. On the other hand, the COMPOSITION Ontology should be able to support collaboration mechanism between business entities. It should be able to describe relations and transactions between supply and demand entities which participate in Marketplace. This need lead us to import the GoodRelations Language (GoodRelations, 2011) ontology, which is one of the most well-known and widely used ontologies in e-commerce domain.

6.1.1 MSDL

The Manufacturing Service Description Language or MSDL, is an OWL-based ontology developed for formal representation of manufacturing services. PLM Alliance research group at the University of Michigan started MSDL development and the first version released at 2005. Currently it is maintained and extended under supervision of Farhad Ameri in the INFONEER Research Group at Texas State University.

MSDL provides sufficient expressivity and extensibility for manufacturing knowledge modelling. MSDL is particularly suitable for description of manufacturing capabilities of SMEs. MSDL describes manufacturing capability into different level of abstraction (shop-level, supplier-level, machine-level, process-level, and device-level) and it is designed to enable automated supplier discovery in distributed environments with focus on mechanical machining services.

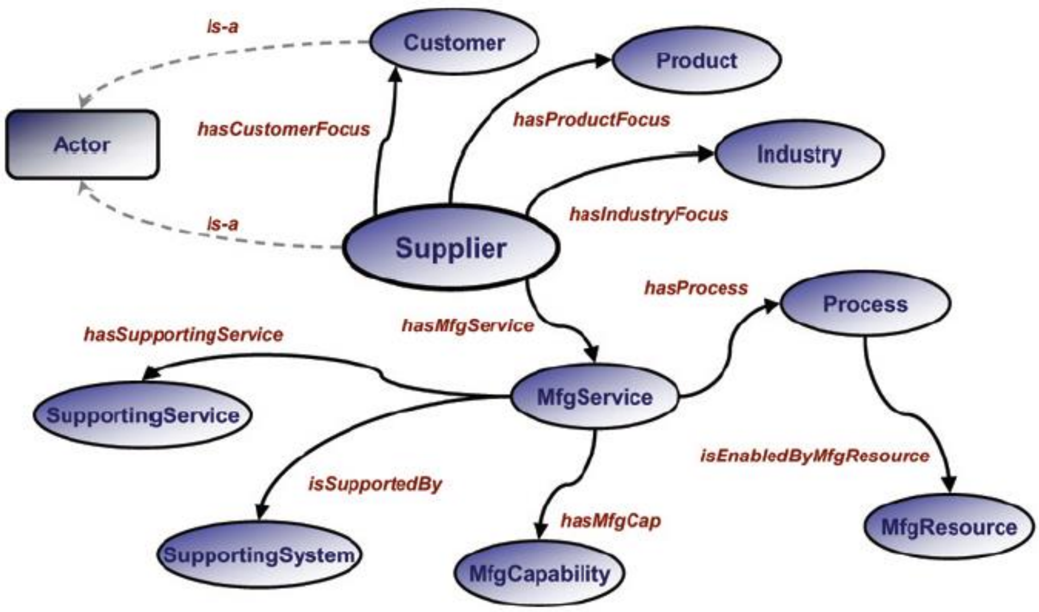

Figure 2: Core Classes of MSDL (Ameri, 2006)

MSDL has two basic parts, MSDL core and MSDL extension. MSDL core is a static part which contains the basic classes for the manufacturing domain description and it is public available as part of many related public reports by the authors. The core classes are presented to Figure 2. MSDL extension is the dynamic part which includes sub-classes and instances built by users. This means that a specific industry is able to build its ontology based on MSDL core part by creating an extension as a dynamic part dedicated to its special domain needs.

6.1.2 MASON

MASON (MAnufacturing's Semantics ONtology) is a manufacturing ontology, aimed to draft a common semantic net in the manufacturing domain. MASON was first proposed by Lemaignan in MASON: A Proposal for an Ontology of Manufacturing Domain. The proposed ontology is written in Web Ontology Language (OWL). The MASON OWL file is public available.

MASON ontology is built over three main concepts:

- *Entities* which aim to provide concepts for specifying an abstract view of a product
- *Operations* relate to processes linked to the manufacturing domain and cover manufacturing, logistic, human and launching operations
- *Resources* represents the whole set of manufacturing linked resources, tools, human resources, and geographic resources like factories and workshops

Figure 3 presents an overview of MASON main classes and sub-classes, and the object properties which connect them:

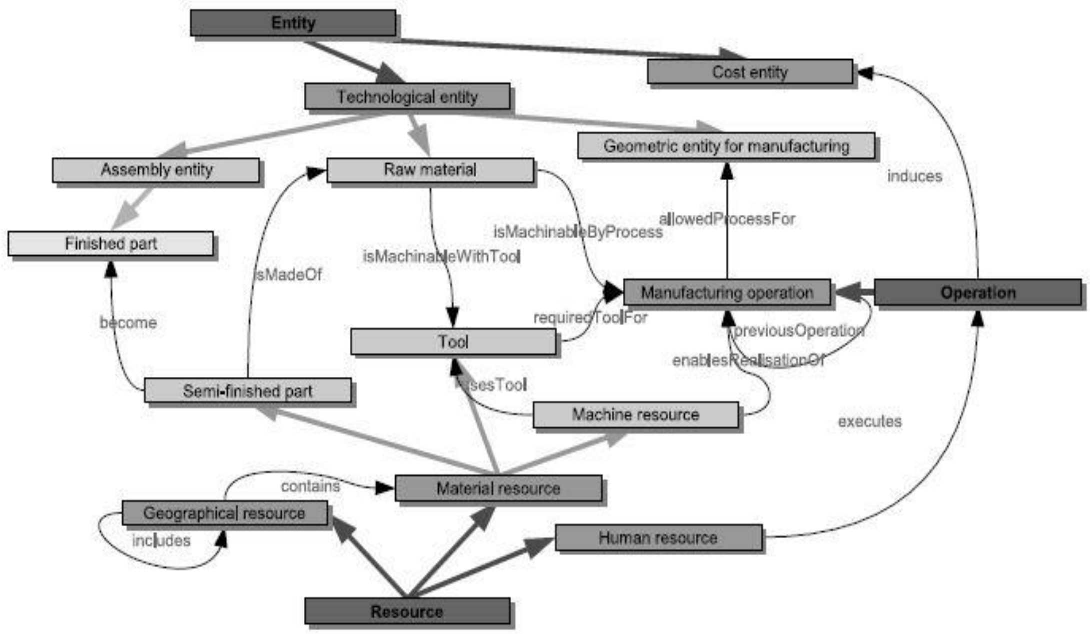

Figure 3: MASON main classes and properties (Lemaignan, 2006)

As depicted in the figure MASON achieves to semantically connect all of its main concepts using object properties. More precisely it is able to connect resources with the operations in a way that it becomes clear, which human resource executes an operation and what materials and machines are required for this execution. Also, it connects operations and resources with the entities they produce. An entity is connected with raw materials, tools, and manufacturing processes which induces costs to this entity.

6.1.3 GoodRelations Language

GoodRelations Language by Martin Hepp is a standardized ontology or vocabulary for products, company data, prices and stores. Nowadays it is one of the most popular ontologies in e-commerce. It can be embedded at web pages and can be processed by many users. In this way increases the visibility of companies' services and products in search engines and other relevant applications.

GoodRelations Language goal is to define data structures for e-commerce that are:

- *Industry-neutral* in a way to be suitable for many kind of services and goods
- *Syntax-neutral.* This means that it should support a large variety of popular syntax such as RDF/XML, RDFa and JSON
- *Valid across the different stages of the value chain.* It has to be valid from raw materials to after-sales supporting services

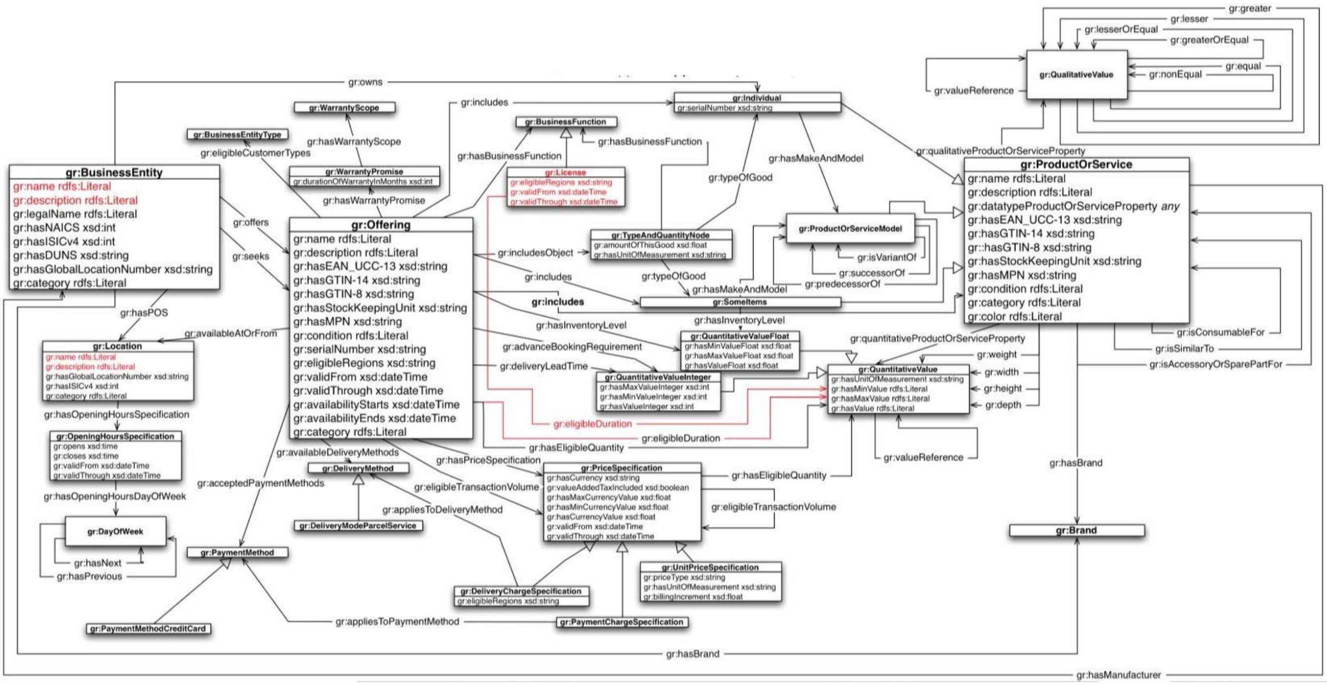

Figure 4: GoodRelations Language main classes and properties[5]

The above figure illustrates the main classes of GoodRelations Language in a graph format produced by Protégé tool. The most important of these classes that lead GoodRelations Language to reach its goals are:

- *BusinessEntity:* For a company or individual representation
- *Offering:* For an offer to sell, or repair something, or to express interest for something
- *ProductOrService:* For the description of a product or a service
- *Location:* For the description of a store location from which an offer is available

By combining these basic classes with the other classes and properties it allows, GoodRelations Language to offer a wide vocabulary, which is suitable to describe almost any kind of e-commerce transactions.

6.2 COMPOSITION Ontology

6.2.1 Methodology

As mentioned in previous sections, Collaborative Manufacturing Services Ontology and Language should be able to describe both the manufacturing and e-commerce domain. In order to achieve this, well-known ontologies from each of these domains will be imported and re-engineered.

From the presented methodologies in section 5.3 – Methodologies for Building Ontologies of this report, the NeOn methodology is selected as the most appropriate one, to cover the needs of the COMPOSITION Ontology's design process. Methodologies such as DILIGENT, METHODOLOGY and On-to-Knowledge are

[5] http://www.heppnetz.de/ontologies/goodrelations/v1.html#uml

highly respected and were been used for years from researchers and developers in ontologies design but in cases of a single ontology development from specifications to implementation. In the case of COMPOSITION we have to combine two different domains, to associate some of intra-factory elements with the Marketplace and to create a domain which will be capable to express and match offers and requests based on manufacturing services and capabilities. Therefore, a methodology, which supports existing knowledge re-usage, re-engineering and offers guidelines in order to build new ontologies, is more related to COMPOSITION targets. This led us to choose the NeOn Methodology over the other methodologies which do not support this kind of design guidelines. More details about the NeOn Methodology are provided in the following sub-section.

Regarding the tools which were presented at the section 5.4 - Leading Tools for Building Ontologies, as a part of our literature review, we have selected Protégé as our main tool for Ontology's implementation. It supports OWL 2.0 and RDF and offers a friendly user interface environment. Protégé is an open-source standalone application compatible with the COMPOSITION project's needs for open and free tools. Protégé supports reasoners[6] which infer logical consequences from a set of axioms and a wide variety of plugins which offers functionalities related to ontology querying, graphical representation and documentation. Some pictures of Protégé environment are available on ANNEX section.

6.2.1.1 NeOn Methodology

The selected methodology for the construction of the Collaborative Manufacturing Services Ontology is the NeOn methodology as already mentioned. The NeOn Methodology (M. C. Suárez-Figueroa, 2010) proposes a variety of different pathways to develop ontologies. These pathways are classified by nine proposed scenarios which manage to cover the most commonly needs occurred during ontology design phase.

The aforementioned nine scenarios for ontology and ontology networks building are the following:

Scenario 1: From Specification to Implementation is about ontology development from scratch without any previous knowledge reuse.

Scenario 2: Reusing and re-engineering non-ontological resources unfolds those cases where non-ontological resources were analysed and used in order to build the new ontology

Scenario 3: Reusing ontological resources covers the case of reusing ready ontological resources.

Scenario 4: Reusing and re-engineering ontological resources refers not only in ontological resources reuse. These resources been engineered again.

Scenario 5: Reusing and merging ontological resources cover the case in which the developers choose more than one of ontological resource to use.

Scenario 6: Reusing, merging, and re-engineering ontological resources covers the case that developers not only choose and merge ontological resources but they also re-engineer them.

Scenario 7: Reusing ontology design patterns. Here, developers access repositories in order to reuse design patterns.

Scenario 8: Restructuring ontological resources is related to cases developers restructure the ontological resources to be integrated in the building ontology network.

Scenario 9: Localizing ontological resources, here the ontology developers adapt ontology to other languages and create a multilingual ontology.

Except the above scenarios the following three valuable components are also provided by NeOn methodology:

- The NeOn Glossary of processes and activities. This glossary identifies and defines the processes and activities involved in ontology's construction. It tries to address the lack of a standard in Ontology Engineering.
- Two ontology network life cycle models. These models specify how to organize the processes and activities based on NeOn Glossary into phases.
- A set of methodological guidelines for the processes and activities included in the NeOn Glossary are provided.

[6] https://en.wikipedia.org/wiki/Semantic_reasoner

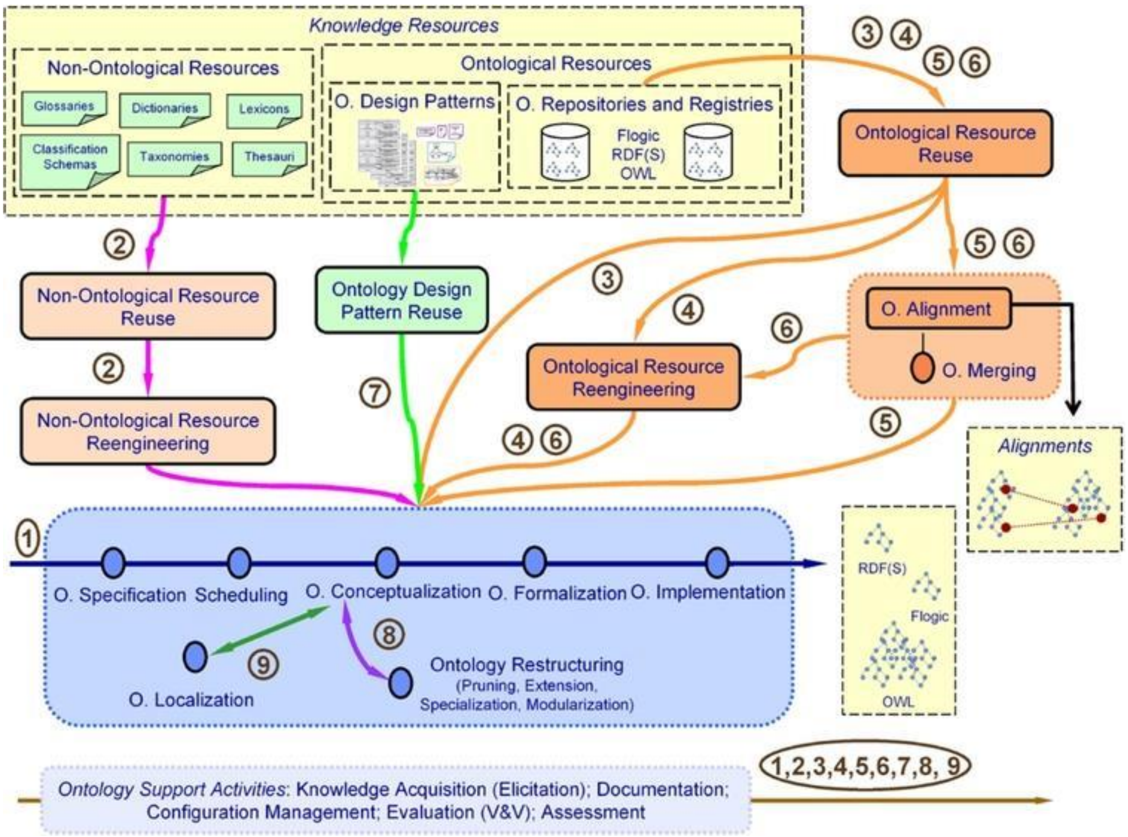

Figure 5: Set of nine scenarios for building ontologies and ontology networks (M.C. Sua´rez-Figueroa, 2012)

For the COMPOSITION Ontology's design the three ontologies should be imported, combined and re-engineered in order to eliminate duplicate information and create a new coherent ontology. As depicted in previous figures and from the aforementioned brief analysis of nine scenarios for building ontologies, *Scenario 6 Reusing, merging, and re-engineering ontological resources* is the one which is completely related to COMPOSITION Ontology's purposes and specifications.

In more details, in Scenario 6 the ontology developers should apply the following steps during the building phase:

1. Select the best possible ontological resources to reuse based on their needs
2. Decide how to reuse the selected ontological resources
3. Perform:
    a. Ontology aligning activity which targets in obtaining a set of alignments among the selected resources
    b. Ontology merging activity which merge the resources using the previous alignments in order to avoid possible overlapping
4. Carry out the ontological resource re-engineering process. Here the resources should be modified in order to be fully compatible with the design's purposes.
5. Development of ontologies
    a. Specify the requirements that the ontology should fulfil (use Ontology Requirements Specification Document - ORSD)
    b. Ontology implementation activity. Here developers start from structure description and semi-computable models and finally implements a computable model using an ontology language.

6.2.1.2 Collaborative Manufacturing Services Ontology and Methodology

This sub-section analyses and describes the design and implementation process of COMPOSITION's Collaborative Manufacturing Services Ontology. The above described NeOn methodology has been adopted and followed. So, the ontology's building phase is described in alignment with NeOn methodology's proposed building steps.

Selection of imported ontological resources

The first step was the selection of the best possible ontological resources to reuse based on COMPOSITION project needs. As mentioned before, Collaborative Manufacturing Services Ontology should be able to represent manufacturing services and resources. Based on literature review and project needs, MASON and MSDL ontologies were selected as the most compatible for COMPOSITION's purposes. They offer sufficient expressivity and extensibility for manufacturing knowledge modelling and they draft a common semantic net in manufacturing domain.

MSDL also provides classes and properties for supply chain description. But after evaluation MSDL considered as unsuitable to cover all of COMPOSITION Collaborative Ecosystem's requirements. Thus, the use of GoodRelations Language aims to cover the requirements of the Collaborative Ecosystem. GoodRelations Language is one of the most well-known and widely used ontologies in e-commerce domain and offers a large variety of classes and properties in order to describe relations and transactions between supply and demand entities. MSDL, MASON and GoodRelations ontologies are presented in more details at section 6.1 of this report.

Decide how to reuse the selected ontological resources

As the COMPOSITION Collaborative Ecosystem aims to be a system capable of hosting a wide range of companies specified in different sub-domains, it was decided the core versions of selected ontologies to be imported to the COMPOSITION Ontology. The proposed ontology intends to be a common vocabulary for the description of supply and demand entities related to the manufacturing domain. This approach aims to make the proposed ontology capable for the description of all Ecosystem participants instead of being an ontology dedicated to one manufacturer or supplier. Thus, the imported core versions of the ontologies are evaluated as the most suitable versions as they offer abstract classes for manufacturing and e-commerce domains description.

More precisely, MASON ontology was selected exclusively for the manufacturing domain description and GoodRelations Language for the description of supply or demand entities and their transactions. On the other hand the use of MSDL is not so strict. Classes and properties of this ontology used for both of domains. Also MSDL offers the central idea of how to connect e-commerce with manufacturing domain by its structure examination.

Ontology aligning and merging activities

After the selection of ontological resources and the decision of the way they will be reused for COMPOSITION purposes two main overlappings have been detected:

- MSDL and MASON have overlapping and duplicate structures within manufacturing domain
- MSDL and GoodRelations Language have overlapping and duplicate structures in e-commerce domain

In the following tables the overlapping is presented in class level. Also the selected class in the final merged version is indicated. In the most of the cases MASON or GoodRelations was selected over MSDL as they are specific in only one domain and they offer better expressivity for these domains.

Table 2: MSDL and MASON overlapping classes' alignment

MSDL Ontology	MASON Ontology	COMPOSITION Ontology
Process represents a manufacturing process which is offered by a Service	*Operation* covers manufacturing, logistic, human and launching operations-processes	*Operation.* COMPOSITION Ontology followed the MASON approach. This class describes processes related to manufacturing but it also provides some operations/processes that support this domain. Moreover, this class provides more relations (properties) between operations/processes and connected resources than MSDL does.
MfgResource represents machine-tools and geographic resources)	*Resource* represents linked resources, like machine-tools, tools, human resources, and geographic resources like plants and workshops)	*Resource* class from MASON was adopted by COMPOSITION Ontology because it offers more resources' descriptions such as human resources. Also it describes more machine-tools.
Material class covers the materials related to manufacturing processes	*Raw Material* covers the list of materials which are machined by tools and they are related to operations/processes	*Raw Material* is the selected class. As Operation and Resource classes are selected from MASON ontology the Raw Material class seems to be the best choice as it is strongly connected with them. Moreover it provides a larger list of materials in comparison with Material class from MSDL
Geometric Shape covers the shape of the parts which are accepted from machining processes	*Geometric Entity* represents the shape of entities can be processed by operations and tools	*Geometric Entity* is the selected class. As Operation and Resource classes are selected from MASON ontology the Geometric Entity class seems to be the best choice as it is strongly connected with them.

Table 3: MSDL and GoodRelations Language overlapping classes' alignment

MSDL Ontology	GoodRelations Language	COMPOSITION Ontology
Service class defines a service that a stakeholder supports/offer s. This services is connected with manufacturing process and resources such as materials and tools	*ProductOrService* class represents a product or a service which is included in an offer or in a request	*Service* from MSDL was adopted by COMPOSITION Ontology. We need to connect a Service with manufacturing processes and resources to an offer/request. MSDL provides these connections as object properties. Actually, processes and resources will be derived from MASON ontology although properties from MSDL can be applied here as they describe similar concepts

MSDL Ontology	GoodRelations Language	COMPOSITION Ontology
Supplier class represents an agent who offers a manufacturing service	*BusinessEntity* class describes an agent who makes or seeks an offer	Keep *BusinessEntity* class as part of COMPOSTITION Ontology because it is connected with offers and requests. These are two very important concepts for Marketplace and they are missing from MSDL core version
Customer class represents an agent who seeks a manufacturing service	*BusinessEntity* class describes an agent who makes or seeks an offer	Keep *BusinessEntity* class as part of COMPOSTITION Ontology because it is connected with offers and requests. These are two very important concepts for Marketplace and they are missing from MSDL core version
RFQ is not MSDL-core class but an extension. Although, the case to use this class was examined in order to decide if it is a better way to represent an offer for a service	*Offer* describes an announcement for the services which a Business Entity provides or the services this Business Entity is looking for	*Offer* class from GoodRelations is finally adopted by COMPOSITION Ontology because it provides a large set of properties and connections to other classes and it is able to describe better the offer as this class was derived for an e-commerce specific ontology.
Advertisement is not MSDL-core class but an extension. Although, the case to use this class was examined in order to decide if it is a better way to represent a request for a service	*Offer* describes an announcement for the services which a Business Entity provides or the services this Business Entity is looking for	*Offer* class from GoodRelations is finally adopted by COMPOSITION Ontology because it provides a large set of properties and connections to other classes and it is able to describe better the request as this class was derived for an e-commerce specific ontology. It is the same class that described above. It is called Offer and it is distinguished is it is actually an offer or a request by object properties.(A Business Entity offers or seeks for an Offer)

Ontological resources' re-engineering process

As soon as aligning and merging activities have been completed, the ontological resources should be modified in order to be fully connected to each other and be compatible with the design's purposes. Many classes from imported ontological resources have been rejected during the previous process in which the overlapping parts have been erased. This process left some classes unconnected and the ontology inconsistent.

In order to create a coherent ontology version which is aligned with COMPOSITION project's requirements the ontological resources, need to be re-engineered. Object properties should be changed as they should be able to cover and connect new concepts after merging activities. The classes represent the domain or the range of some properties is possible to have been replaced by classes of an overlapping resource so these properties should be deleted or they should point now in a new domain or range. Moreover, new classes, new sub-classes and new properties should be added to cover COMPOSITION Ecosystem requirements. The basic goals of the re-engineering process were the following:

- Connect a Service(MSDL) with corresponding Operations(MASON)
- Connect a Service(MSDL) with an Offer(GoodRelations)
- Connect a Service(MSDL) with a Business Entity(GoodRelations)

- Create a Generic Terms catalogue which enables the use of same terms for similar concepts
- Associate concepts with Generic Terms
- Create concepts helpful to COMPOSITION Matchmaker

Extension of ontology and its concepts

The extension of the ontology follows the above-described processes of ontology alignment, merging and re-engineering. The imported ontologies covers the largest part of the required concepts for the projects pilot cases and in general cases related to manufacturing marketplaces. However, in order to cover all the use cases requirements and the needs of a complete and real manufacturing ecosystem, many concepts related to waste management and software for supporting the manufacturing domain should be added. The main goals of this extension process were the following:

- Extend Services to be able to support waste management concepts as they are part of COMPOSITION project
- Extend Operations and resources in order to be able to support waste management concepts
- Extend Services and Operations to be able to support concepts related to software and consulting solutions connected to manufacturing domain

Development of ontology

The last stage of design and implementation process was the development of the ontology. First the requirements were specified based on Ecosystem's needs and *D2.2 Initial requirements specification, D2.5 Lessons Learned and updated requirements report I* and *D2.6 Lessons Learned and updated requirements report II.* Then the ontology was implemented using Protégé tool.

The requirements of the Collaborative Manufacturing Services Ontology modelled to the Table 4, Ontology Requirements Specification Document (ORSD), as it proposed by NeOn methodology.

After the definition of basic requirements of Collaborative Manufacturing Services Ontology the ontology was implemented using Protégé tool:

- A new empty ontology OWL file was created using Protégé
- MSDL, MASON, GoodRelations imported using Protégé
- Based on work in aligning and merging activities the overlapping concepts were deleted using the interface of the tool
- Based on project's requirements new classes, sub-classes and properties were added. Also others were modified
- The stable Ontology version was continuously updated in order to cover project needs
- After the development of the final OWL file, it was deployed as part of the Matchmaker package which consists the complete semantic framework of the COMPOSITION Project. Then the Marketplace Agents who add more individuals automatically extend the Collaborative Manufacturing Services Ontology.

Table 4: ORSD of COMPOSITION Collaborative Manufacturing Services Ontology

	ONTOLOGY REQUIREMENTS SPECIFICATION DOCUMENT
1	**Purpose**
	The purpose of creating the Collaborative Manufacturing Services Ontology is to be used as a knowledge base able to support flexible specification and execution of manufacturing collaboration schemes
2	**Scope**
	The scope of the Collaborative Manufacturing Services Ontology is to enable both the description of supply/demand entities participating in the Ecosystem and the description of manufacturing services' capabilities and resources for entities participating in the Ecosystem
3	**Implementation Language**
	The Collaborative Manufacturing Services Ontology will be implemented in the OWL 2 language using the Protégé tool
4	**Intended End-Users**
	User 1: Marketplace Agents • Supplier Agent • Requester Agent User 2: • Matchmaker
5	**Intended Uses**
	Use 1: Keep information and data about agents. An agent represents a business entity at the Marketplace. Data about a business entity, its resources and services are stored to the ontology. Use 2: Provide data about agents and their resources and services Use 3: Describe offers and requests during transactions and bidding processes in the Marketplace Use 4: Used by Matchmaker. Matchmaker infers new knowledge by applying semantic rules to ontology
6	**Ontology Requirements**
	a. **Non- Functional Requirements**
	1. Ontology should be a knowledge base for the Ecosystem 2. Ontology should describe manufacturing domain 3. Ontology should describe supply/demand entities 4. Ontology should be able to support concepts from waste management domain 5. Ontology should be able to describe concepts related to software solutions for the manufacturing domain 6. Ontology should be updated by agents and generally be available to them – Ontology individuals should be created automatically by the agents that participate in the Ecosystem 7. Ontology should be correlated with Matchmaker 8. Ontology should be implemented in ontology language 9. Ontology should be compatible with Marketplace's definition
	b. **Functional Requirements**
	1. How a business entity will be described into the Ecosystem? The Ontology should contain and describe concepts of the e-commerce domain in correlation with services, operations, resources, of manufacturing domain. 2. How a supplier or requester will be able to express their offers or demands? The COMPOSITION Ontology should have concepts for the description of offers and requests. Also it should connect these information with the corresponding business entity 3. How an agent can update knowledge base's information? The Ontology should be able to be queried from agents with SPARQL queries. This requirement is also connected with Ontology API 4. Should the ontology help Matchmaker to infer knew knowledge? COMPOSITION Ontology should offer classes or properties that will be helpful to Matchmaker. These concepts will be filled by Matchmaker's rules and will provide the new knowledge 5. Should the ontology represent all the knowledge from IIMS to Marketplace? Ontology should offer concepts and relations only for data necessary to the Marketplace. There is no need to hold data from sensors for example.

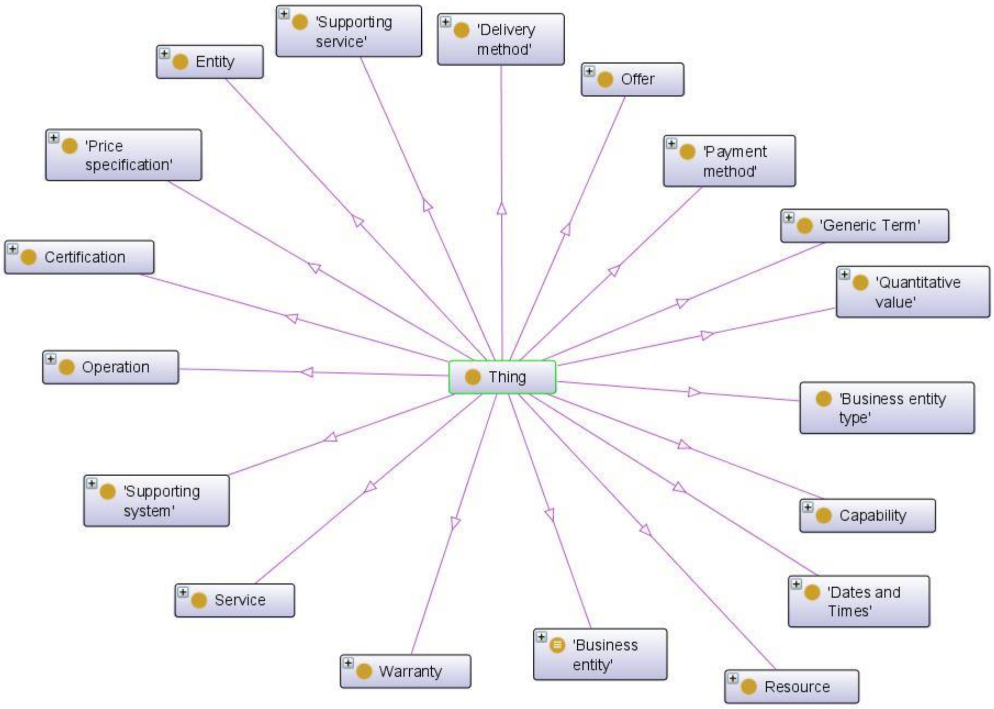

Figure 6: COMPOSITION Collaborative Manufacturing Services Ontology's Class Overview

6.2.2 Ontology Specifications

In this sub-section the specifications of the new created ontology are described. The main classes of the aforementioned ontology are presented. Moreover, some basic object and data properties are presented. Full documentation of the COMPOSITION Collaborative Manufacturing Ontology is provided alongside with this report. The documentation was exported using OWLDoc plugin of Protégé tool.

For each class we define:

- *Class name*: the name of the class which is described
- *Description*: a short description for this class
- *Class hierarchy*: we provide a graph with the sub-classes(if any exists) of mentioned class
- *Object properties*: we provide a table with main object properties of the class
- *Data properties*: we provide a table with main data properties of the class

Business entity class

The "Business entity" class and its sub-classes represent an Ecosystem Agent who has a service (e.g. manufacturing service) and provides or seeks an offer. Every agent who is associated with the Marketplace has this type. The figure below presents sub-classes of "Business entity" class. The following tables present basic object and data properties, respectively.

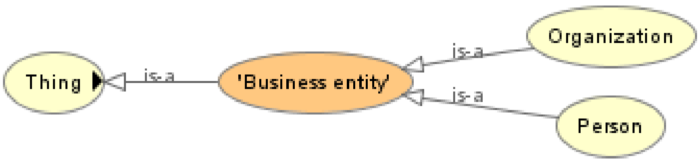

Figure 7: "Business entity" class and sub-classes

Table 5: Object Properties of "Business entity" class

Object Property	Description	Range
offers	Refers to the offers provided by a business entity	Offer
seeksOffer	Refers to the offers requested by a business entity	Offer
hasService	Refers to the services provided by a business entity	Service
matchesWith	Refers to a business entity which is matched with another business entity for a specific term	Business entity
hasPOS	Refers to the position of a business entity	Location
requestFulfilledBy	Refers to request of a business entity fulfilled by a specific offer	Offer

Table 6: Data Properties of "Business entity" class

Data Property	Description	Type
legalName	The legal name of a business entity	Literal
hasID	The agent(business entity) ID within the Marketplace	String
hasRating	The business entity's rating within the Marketplace	Integer
description	A short textual description of an entity.	Literal
hasGlobalLocationNumber	The Global Location Number is a thirteen-digit number used to identify parties and physical locations.	String
hasName	Equivalent to the title of an entity. Not the legal name	Literal
taxID	The Tax ID of a business entity. It is usually assigned by the country of residence	String
valueAddID	The Value-added Tax ID of a business entity	String

Business entity type class

The "Business entity type" class represents the legal form, the size and the position of a business entity in value chain. It is used to specify eligible customers for an offer. There are no sub-classes for this class. Also there are no object properties. We create only individuals of this class which consist the range of the object property, named eligibleCustomerTypes from class "Offer".

Table 7: Data Properties of "Business entity type" class

Data Property	Description	Type
description	A short textual description of an entity.	Literal
hasName	Equivalent to the title of an entity	Literal

Capability class

The "Capability" class and its sub-classes represent the capability of a service. It describes manufacturing capabilities based on machining capabilities, waste management capabilities related to supported tonnages of the services and the software solutions capabilities related to supporting functionalities such as security etc.

Figure 8: "Capability" class and sub-classes

Table 8: Data Properties of "Capability" class

Data Property	Description	Type
hasUnit	The unit of measurement of a capability value	String
hasWeight	The weight of stock	Float
description	A short textual description of an entity	Literal
hasName	Equivalent to the title of an entity	Literal

Dates and Times class

The “Dates and Times” class represents the days that a business entity has opening hours. Also it can represent the day of delivery or the day of availability of a service. This class also supports the description of opening hours of a business entity. So it has two sub-classes: Days of the week and Opening hours specification. The main properties of these sub-classes are presented in the following tables.

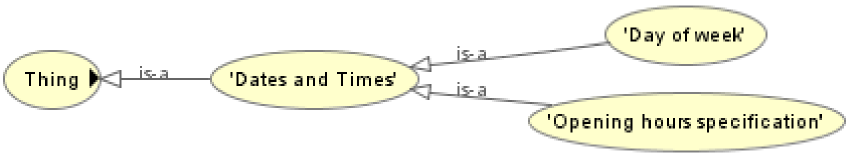

Figure 9: "Dates and Times" class and sub-classes

Table 9: Object Properties of "Dates and Times" class

Object Property	Description	Range
hasNext	Refers to next day of the week	Day of the week
hasPrevious	Refers to previous day of the week	Day of the week
hasOpeningHoursDayOfWeek	Specifies the day of the week to which opening hours is related	Day of the week

Table 10: Data Properties of "Dates and Times" class

Data Property	Description	Type
closes	The closing hour of a specific location of business entity on a given day of the week	Time
opens	The opening hour of a specific location of business entity on a given day of the week	Time
description	A short textual description of an entity.	Literal
hasName	Equivalent to the title of an entity	Literal

Delivery method class

The "Delivery method" class and its sub-class define the available delivery options for a service or product.

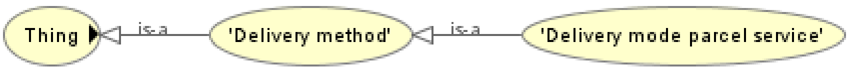

Figure 10: "Delivery method" class and sub-classes

"Delivery method" instances are used only as the range for other object properties.

Table 11: Data Properties of "Delivery method" class

Data Property	Description	Type
description	A short textual description of an entity.	Literal
hasName	Equivalent to the title of an entity	Literal

Entity class

The "Entity" class and its sub-classes represent an entity as a result of a manufacturing process and describe its geometric flaw and entity, assembly entity and raw material. The sub-classes are presented in the next figure.

Figure 11: "Entity" class and sub-classes

The next tables contain some of the basic object properties of "Entity" class and its sub-classes:

Table 12: Object Properties of "Entity" class

Object Property	Description	Range
hasPrice	Refers to the price of an entity	Unit price specification
hasShape	Refers to the shape of a geometric flaw	Shape
isMachinableWithTool	Refers to the tool which process a raw material	Tool
isMachinableByProcess	Refers to the operation in which an entity is processed	Operation
isMadeOf	Refers to the material that a part is made of	Raw material
hasCertification	Refers to the certification of an entity or part or material	Certification

Table 13: Data Properties of "Entity" class

Data Property	Description	Type
hasVolume	A finished part has volume	float
hasRugosity	The rugosity of a geometric flaw	float
description	A short textual description of the entity.	Literal
hasName	Equivalent to the title of an entity.	Literal

Generic Term class

The “Generic Term” class and its sub-classes define common operations, materials and tools. This will enable the use of same terms for similar concepts. The vendor-specific concepts will be mapped with corresponding terms of the common “Generic term” class’ instances.

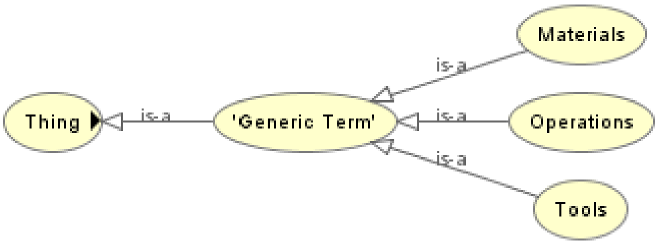

Figure 12: "Generic term" class and sub-classes

The “Generic term” class is not the domain of any object property. It is used as a common dictionary and it is the range of the properties that map other operations, materials and tools to the concepts of this dictionary. Moreover, there are no data properties correlated with this class.

This class is a core concept of the Matchmaker component’s functionality. Every business entity use its own terms to describe one of its offered services. But every one of these vendor specific terms will be mapped with a common generic term. In this way, on the one hand every business entity will be able to participate in the Marketplace and advertise its services, products etc. with its own terms. On the other hand, the Matchmaker will be able to match similar concepts in order to set the Marketplace capable to relate offers and requests among stakeholders or to find possible solutions for some Marketplace participants. The following figure describes in a very simple and abstract way, how the vendor specific operations for scrap metal management of three different business entities is mapped to the same concept.

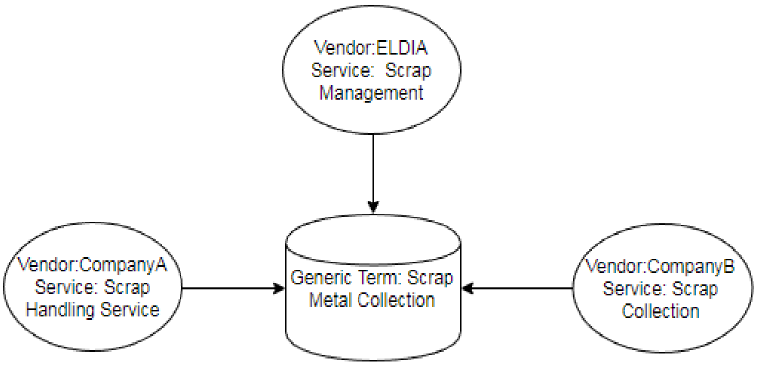

Figure 13: Mapping of vendor specific concepts

Offer class

The “Offer” class represents a public announcement of a business entity that provides or seeks a certain service or product. This is a key class for the description of offers and requests of business entities which are involved into COMPOSITION Ecosystem. The “Offer” class has not any sub-classes. Its basic object properties are presented at the table below.

Table 14: Object Properties of "Offer" class

Object Property	Description	Range
includes	Refers to the service or product which is provided by an offer	Service
acceptedPaymentMethods	Refers to the available payment methods for a certain offer	Payment method

Object Property	Description	Range
addOn	Points to other offers which are linked with a basic offer	Offer
availableAtOrFrom	Refers to the location where the offered service or product is available	Geographic resource
availableDeliveryMethods	Refers to the available delivery methods of a certain offer	Delivery method
advancedBookingRequirements	Refers to the min and max amount of time that is required between accepting the Offer and the actual usage of the resource or service.	Quantitative value
eligibleDuration	Refers to the minimal and maximal duration for which the given Offer is valid	Quantitative value
eligibleCustomerTypes	Refers to the eligible types of customers for a certain offer	Business entity type
eligibleTransactionVolume	Indicates the minimum purchasing volume	Price specification
hasPriceSpecifification	Links an offer to price specifications	Price specification
hasWarrantyPromise	Links an offer with a warranty promise for a product or service by business entity	Warranty promise
deliveryLeadTime	Refers to the delivery time of the offered service	Quantitative value
eligibleQuantity	Specifies the quantities for which an offer is valid	Quantitative value
offerProvidedBy	Points to the business entity which provides or seeks an offer	Business entity
hasInventoryLevel	Specifies the current approximate inventory level of the products that included in an Offer	Quantitative value

Except the object properties some of main data properties of class “Offer” are also presented in the following table.

Table 15: Data Properties of "Offer" class

Data Property	Description	Type
validFrom	The beginning of the validity of an offer	dateTime
validThrough	The end of the validity of an offer	dateTime
eligibleRegions	The geo-political regions where an offer is available	string
availabilityStarts	Specifies the beginning of the availability of the Service included in the Offer	dateTime
availabilityEnds	Specifies the end of the availability of the Service included in the Offer	dateTime
hasOfferID	The identity number of an offer inside the Marketplace	string
description	A short textual description of an entity.	Literal
hasName	Equivalent to the title of an entity or resource.	Literal
serialNumber	Refers to alphanumeric number. This property can be attached to an Offer in cases where the included products are not modelled in more details	String

Data Property	Description	Type
quantityFulfilment	The property was designed to be filled and used from the Matchmaker for matching processes	Boolean
productID	The given ID of a product contained in an Offer or Service	String

Operation class

The “Operation” class and its sub-classes represent the processes of a service. Especially the manufacturing processes. But supporting operations related to human or launching processes are represented as well. Moreover, this class offers the representation of waste management processes and software solutions, which are strongly related with the COMPOSITION project.The following tables present basic object and data properties, respectively.

Table 16: Object Properties of "Operation" class

Object Property	Description	Range
induces	Refers to the price cost that induces the execution of an operation	Price specification
isExecutedBy	Refers to the human resource that executes an operation	Human resource
mappedToCommonTerm	A specific operation is mapped to a generic term	Generic term
allowedProcessFor	Refers to material which is valid for a manufacturing operation	Raw material
handlingMaterial	Refers to materials that can be handled by waste management operations	Raw material
requiresTruck	Refers to the truck resources that are required to a supply-chain operation	Resource
requiresTool	Refers to the tool that is required to a manufacturing operation in order to execute a process related to a raw material	Tool
requiresMachine	Refers to the machine resource that is required to a manufacturing operation in order to execute a process	Machine resource
previousOperation	Points to a previous operation	Operation

Table 17: Data Properties of "Operation" class

Data Property	Description	Type
hasDuration	The duration of an operation	possitiveInteger
hasDelay	The delay of an operation	possitiveInteger
isContinuous	Describe if an operation is a continuous process	Boolean
description	A short textual description of an entity	Literal
hasName	Equivalent to the title of an entity	Literal
modifiesGeometry	Indicates if an operation modifies the geometry of a material or part	Boolean

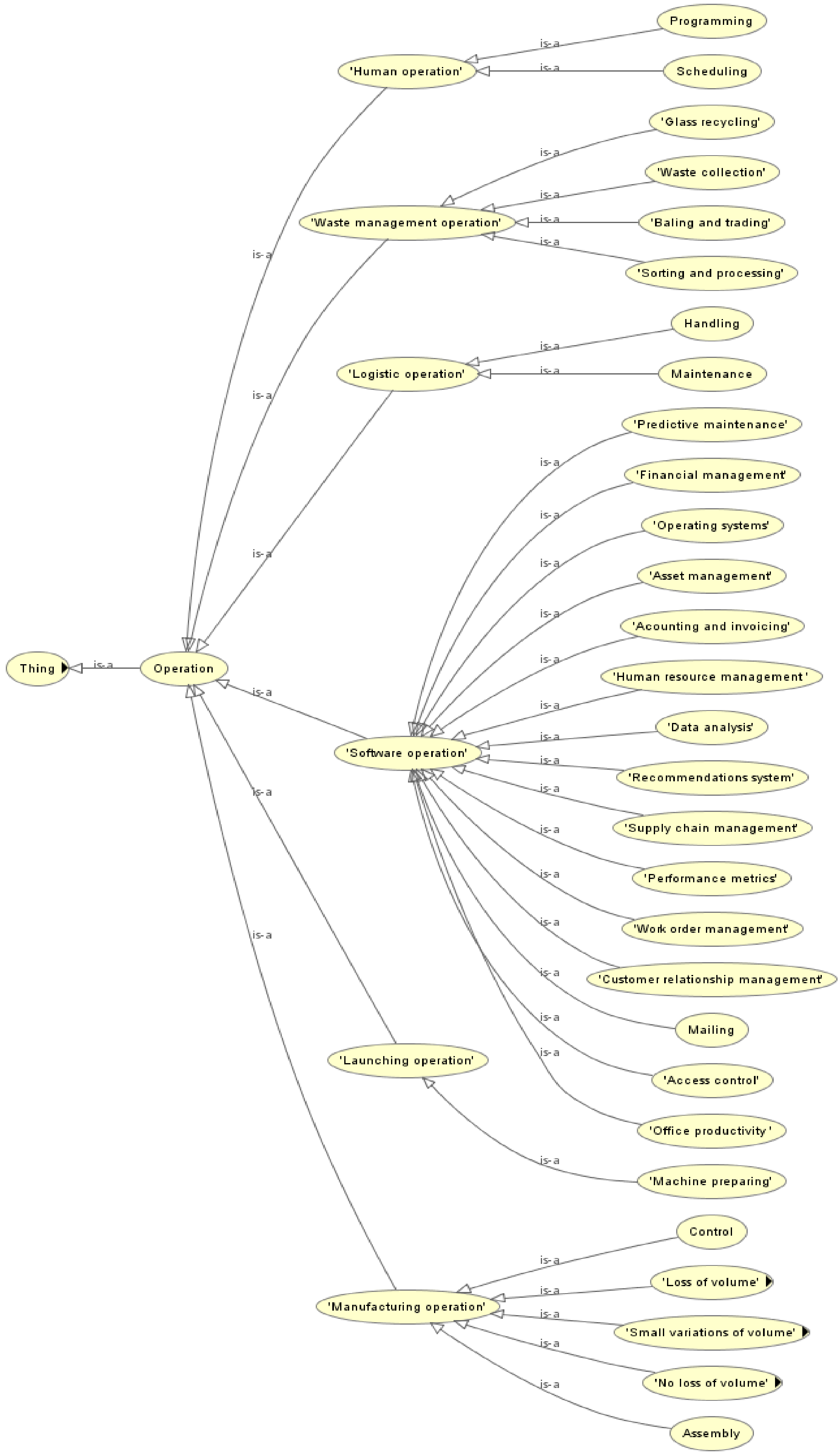

Figure 14: "Operation" class and sub-classes

Payment method class

The “Payment method” class describes the available procedures for transferring the requested amount for a purchase. It contains only a sub-class which is related to credit cards as a payment method.

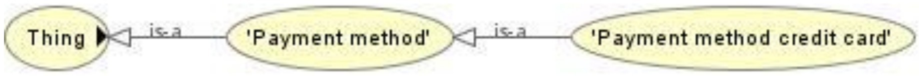

Figure 15: "Payment method" class and sub-classes

The individuals of this class and its sub-class are well-known payments methods that are commonly used in transactions such as cash, bank transfer, VISA, PayPal etc. The only purpose of this class is to create this kind of individuals and they will be used as the range of properties of other classes such as “Offer” and “Price specification”. As a result there was no need to construct object properties with domain the “Payment method” class.

Table 18: Data Properties of "Payment method" class

Data Property	Description	Type
description	A short textual description of the class	Literal
hasName	Equivalent to the title of an entity	Literal

Price specification class

The “Price specification” class and its sub-classes specify the price of a unit, additional delivery costs and additional costs related to a payment method. The figure below presents sub-classes of “Price specification” class. The following tables present basic object and data properties, respectively.

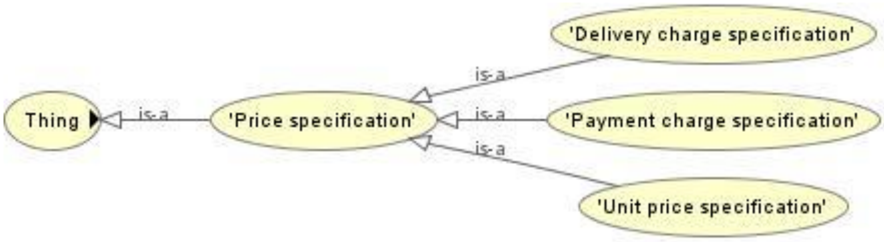

Figure 16: "Price specification" class and sub-classes

Table 19: Object Properties of "Price specification" class

Object Property	Description	Range
appliesToPaymentMethod	Refers to the available payment methods	Payment method
appliesToDeliveryMethod	Refers to the delivery method which induces this cost	Delivery method
isInducedBy	Refers to the operation which adds costs by its execution	Operation

Table 20: Data Properties of "Price specification" class

Data Property	Description	Type
hasCurrency	The currency related to a price (e.g. EUR)	string
hasCurrencyValue	The amount of money for a price or payment charge	float
hasMaxCurrencyValue	The upper bound of the amount of money for a price or payment charge	float
hasMinCurrencyValue	The lower bound of the amount of money for a price or payment charge	float

Data Property	Description	Type
valueAddedTaxIncluded	Specifies if the value-added-tax is included in the price	boolean
description	A short textual description of the class	Literal
hasName	Equivalent to the title of an entity	Literal

Quantitative value class

The “Quantitative value” class and its sub-classes are used as numerical intervals that represent the range of a certain property. Their individuals are mainly used as the range of other classes’ object properties related to quantity measurements. So, we did not adopt any object properties which have this class and its sub-classes as domain. The sub-classes and main data properties related to “Quantitative value” class are presented below.

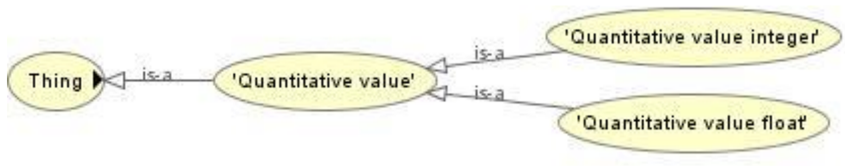

Figure 17: "Quantitative value" class and sub-classes

Table 21: Data Properties of "Quantitative value" class

Data Property	Description	Type
hasValue	The property is a single point value	Literal
hasMinValue	The property captures the lower limit of a value	Literal
hasMaxValue	The property captures the upper limit of a value	Literal
hasValueFloat	A quantitative property is a single point float value	float
hasMinValueFloat	The property captures the lower limit of a float value	float
hasMaxValueFloat	The property captures the upper limit of a float value	float
hasValueInteger	A quantitative property is a single point integer value	int
hasMinValueInteger	The property captures the lower limit of an integer value	int
hasMaxValueInteger	The property captures the upper limit of an integer value	int
hasUnitOfMeasurement	The unit of measurement of a quantitative value	string

Resource class

The “Resource” class and its sub-classes represent the total set of linked resources of a business entity. They are able to describe resources such as buildings and sites, human resources, truck resources, machines and tools. The figure below presents sub-classes of “Resource” class.

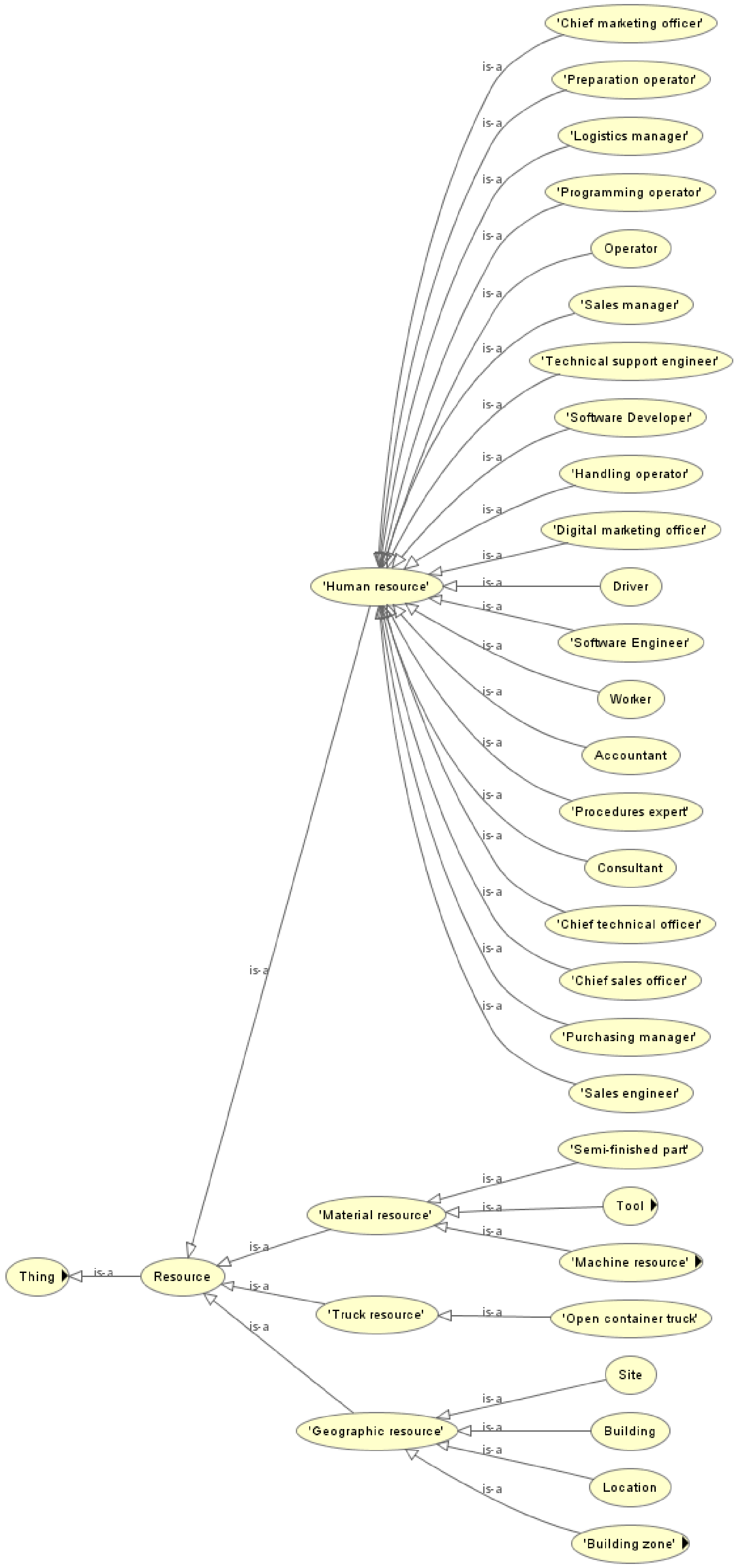

Figure 18: "Resource" class and sub-classes

The following tables present some of basic object properties of “Resource” class and its sub-classes, respectively.

Table 22: Object Properties of "Resource" class

Object Property	Description	Range
contains	Refers to the material resource which is included in a geographical resource	Material resource
includes	Refers to a geographical resource which is included in another geographical resource	Geographical resource
enablesRealisationOf	Refers to an operation which requires a machine resource	Machine resource
execute	Refers to an operation which is executed by a human resource	Human resource
becomes	Refers to transformation of one resource in another after machining processes	Material Resource
isMadeOf	Refers to the material that a resources is made of	Raw Material
usesTool	Refers to the tool that is used by a machine resource	Tool
requiredToolFor	Refers to the tool that is required to a manufacturing operation in order to execute a process	Manufacturing operation
toolUsableOn	Refers to a raw material in which a tool is used	Raw material
toolMappedToCommonTerm	A specific tool is mapped to a tool which is described in generic terms	Tools

Table 23: Data Properties of "Resource" class

Data Property	Description	Type
resourceName	The name of a resource	string
resourceID	The ID of a resource	string
description	Short description of a resource	Literal
operatingRate	The operating rate for a machine resource	float

Certification Class

The “Certification” class conceptualize the certification of an entity success business entities, materials, machines, processes etc. The certification can be for example an ISO[7]. The certifications are important concepts on negotiation and selection processes in a Marketplace. For COMPOSITION purposes, they are used as names of specific certifications.

Table 24: Data Properties of "Certification" class

Data Property	Description	Type
description	A short textual description of the class	Literal
hasName	Equivalent to the title of an entity	Literal

[7] https://www.iso.org/home.html

Service class

The “Service” class and its sub-classes conceptualize all operations and processes related to a product in an abstract level. A service includes operations which are related with resources. It is the general concept of what service or product offers a business entity. The next figure presents the sub-classes of “Service” class.

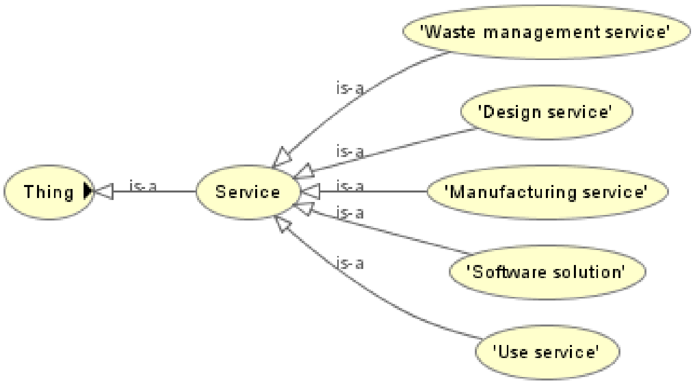

Figure 19: "Service" class and sub-classes

As this class describes processes in a more abstract level is not the domain in any data property. It is connected with processes and their own data properties. The basic object properties of class “Service” are the following:

Table 25: Object Properties of "Service" class

Object Property	Description	Range
hasManufacturer	Links a service or product to the business entity that produces it	Business entity
hasCapability	Refers to the capability of an offered service	Capability
hasSupportingService	Links a service with a supporting service	Supporting service
isSupportedBy	Refers to a system that supports a service	Supporting system
hasOperation	Refers to the process/operation which is actually executed in this service	Operation
seeksOperation	Refers to the process/operation which is actually executed in this service and is requested by another business entity’s service	Operation

Table 26: Data Properties of "Service" class

Data Property	Description	Type
dataTypeServiceProperty	Refers to the data type service property	Literal
name	The name of a service	Literal
description	Short description of an entity	Literal
productID	The given ID of a product contained in an Offer or Service	String

Supporting service class

The “Supporting service” class and its sub-classes represent services which are not basic services but are related to the basic one and support them. They are actually from a different domain than the main services of a business entity, but they are valuable for a company’s activities and processes. As described before for “Service” class, it describes processes in a more abstract level and it is not the domain in any data property. It is connected with processes and their own data properties. It is the same for the “Supporting service” class.

The sub-classes and main object and data properties related to “Supporting service” class are presented below.

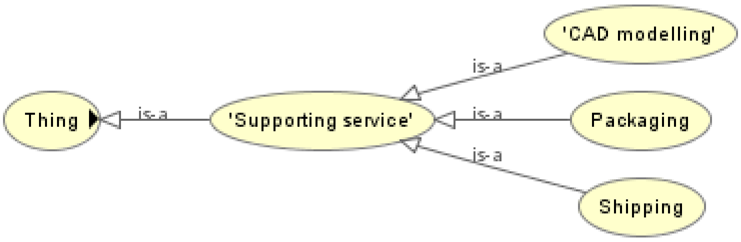

Figure 20: "Supporting service" class and sub-classes

Table 27: Object Properties of "Supporting service" class

Object Property	Description	Range
supports	Links a supporting service to the main service it supports	Service
hasRelatedOperation	Links a supporting service with a human or logistic operation	Human operation and Logistic operation
isSupportedBy	Refers to a system supports supporting service	Supporting system

Table 28: Data Properties of "Supporting service" class

Data Property	Description	Type
description	A short textual description of the class	Literal
hasName	Equivalent to the title of an entity	Literal

Supporting system class

The “Supporting system” class and its sub-classes represent some systems which support a business entity’s services. The figure below presents sub-classes of “Supporting system” class. The following tables present basic object and data properties, respectively.

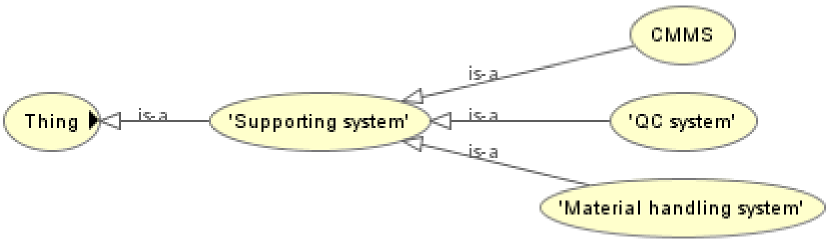

Figure 21: "Supporting system" class and sub-classes

Table 29: Object Properties of "Supporting system" class

Object Property	Description	Range
supportService	Links a supporting system to the service it supports	Service and Supporting service
usedBy	Refers to the human resource that uses a supporting system	Human resource
isLocatedIn	Links a supporting system to a geographical resource where the system is contained	Geographical resource

Table 30: Data Properties of "Supporting system" class

Data Property	Description	Type
systemName	The name of a system	String
systemID	The ID of a system	String
description	Short description of a system	Literal
hasName	Equivalent to the title of an entity	Literal

Warranty class

The "Warranty" class and its sub-classes represent the duration and the scope of free services that will be provided to a customer in case of a possible malfunction or problem. The figure below presents sub-classes of the "Warranty" class. The following tables present basic object and data properties, respectively.

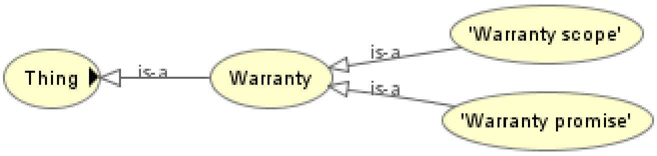

Figure 22: "Warranty" class and sub-classes

Table 31: Object Properties of "Warranty" class

Object Property	Description	Range
hasWarrantyScope	Refers to warranty scope of a warranty promise	Warranty scope
warrantyPromiseOf	Refers to the offer which is related with a warranty promise	Offer

Table 32: Data Properties of "Warranty" class

Data Property	Description	Type
durationOfWarrantyInMonths	Specifies the duration of a warranty promise in months	int
description	Description of a warranty which comes alongside with an offer	Literal
hasName	Equivalent to the title of an entity	Literal

In the next figure, an example of the representation of a Polishing procedure from KLEEMANN factory based on terms of Collaborative Manufacturing Services Ontology is presented.

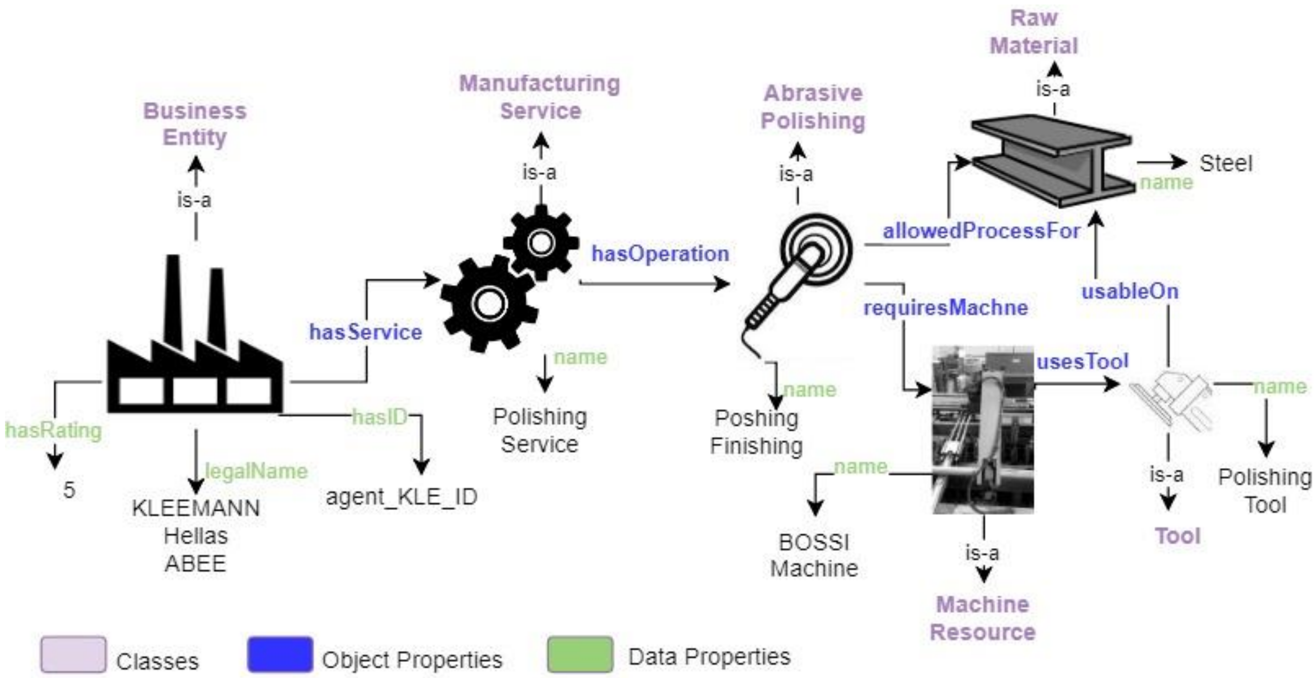

Figure 23: Modelling of KLEEMANN Polishing Procedure using Collaborative Manufacturing Services Ontology

7. COMPOSITION Ontology API

As described in the executive summary and introductory sections besides the Collaborative Manufacturing Services Ontology, an API for the manipulation of the ontological resources has been implemented and is presented in this report. This API provides a basic set of interfaces/services and in this chapter we consider as a complete API both the interfaces and the query engine in the back-end. The Marketplace components such as the Agents are able to access and extend the Ontology using this API. On the contrary, of the first version of this report, the Ontology API is not a standalone application anymore, but it is part of the Matchmaker component, which consists the complete semantic framework of the project. In this section, some key components of Ontology API's implementation and its supported interfaces are presented.

7.1 Methodology and Implementation Technologies

The COMPOSITION Ontology API has been developed in Java and it is offered through RESTful web services. Its development was built upon Apache Jena API. In advance of the description of the Ontology API's implementation, we will offer a brief analysis of Apache Jena which is the key component of COMPOSITION Ontology API and offers all the necessary functionality to create, connect and modify an ontology store.

7.1.1 Introduction to Apache Jena

Apache Jena is an open source Semantic Web framework for Java that has been extensively used in a wide variety of semantic web applications and demonstrators. The main component of this framework is an API that provides data extraction from RDF graphs as well as writing to them. The graphs are defined as an abstract model. A model can collect data from files, databases, URLs or a combination of these. Jena provides a programmatic environment for RDF, RDFS and OWL, SPARQL, GRDDL, and includes a rule-based inference engine. The figure below represents Jena framework's architecture. Subsequently, the different parts that compose Jena's architecture are presented together with the interaction between them.

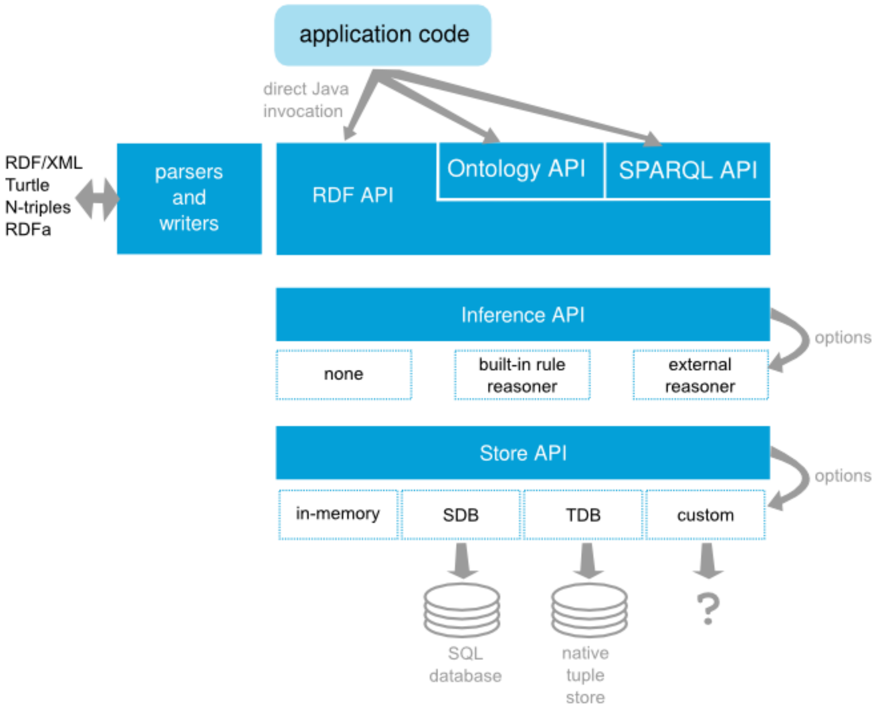

Figure 24: Apache Jena's framework architecture (Apache Jena, 2017)

The RDF API - the core RDF API in Jena

RDF can be better comprehended if it is represented in the form of node and arc diagrams, namely in RDF graphs. Each relationship points only to one direction. Part of the RDF graphs is resources. A resource is some entity. It could be a web resource or it could be a concrete physical thing. It could also be an abstract idea. Resources are named by a Uniform Resource Identifier (URI). Resources have attributes called properties and lastly, properties have data values called literals.

Jena is a Java API which can be used to create and manipulate RDF graphs. The interfaces representing resources, properties and literals are called Resource, Property and Literal respectively. In Jena, a graph is called a model and is represented by the Model interface.

The basic concepts of RDF containers in Jena are three:

- graph, a mathematical view of the directed relations between nodes in a connected structure
- Model, a rich Java API with many convenience methods for Java application developers
- Graph, a simpler Java API intended for extending Jena's functionality.

The most important of these concepts is Model, thus, it is going to be further analyzed. Each arc in an RDF Model is called a *statement*. Each statement asserts a fact about a resource. A statement is called a triple since it contains three distinct parts: the *subject*, which is the resource from which the arc leaves, the *predicate*, which is the property that labels the arc and the *object*, which is the resource or literal pointed to by the arc. The Statement interface provides accessor methods to the subject predicate and object of a statement.

Ontology API

Jena allows a programmer to specify, in an open, meaningful way the concepts and relationships that collectively characterize some domain. The advantage of ontology is that it is an explicit, first-class description; it can be published and reused for different purposes.

There is a multitude of different ontology languages available for modeling ontology information on the semantic web. They range from the most expressive, OWL to the weakest, RDFS. Jena Ontology API aims to provide a coherent programming interface for ontology application development. The Ontology API is independent of the language used: the Java class names are not specific to the underlying language.

In order for distinction between various representations to be clear, each of the ontology languages has a profile, which lists the permitted constructs and the names of the classes and properties. The profile is bound to an ontology model, which is an extended version of Jena's Model class. The base Model allows access to the statements in a collection of RDF data. Jena ontology interface provides support for the kinds of constructs expected to be in ontology: classes (in a class hierarchy), properties (in a property hierarchy) and individuals.

SPARQL API

SPARQL is a query language and a protocol for accessing RDF designed. As a query language, SPARQL is "data-oriented" in that it only queries the information held in the models and does not infer in the query language itself. Jena model creates triples on-demand in order to give the impression that they already exist, including OWL reasoning. SPARQL takes the description of the application demands, in the form of a query, and returns that information, in the form of a set of bindings or an RDF graph.

Interference API

The Jena inference subsystem is designed to allow a range of inference engines or reasoners to be plugged into Jena. Such engines are used to derive additional RDF assertions which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner.

Store API

Two individual parts of the Store API are TDB and SDB, as shown in Figure 5.

TDB is a component of Jena for RDF storage and query. It is a fast persistent triple store that stores directly to disk and supports the full range of Jena APIs. TDB can be used as a high performance RDF store on a single machine. A TDB store can be accessed and managed with the provided command line scripts and via the Jena API. When accessed using transactions, a TDB dataset is protected against corruption, unexpected process terminations and system crashes.

SDB uses an SQL database for the storage and query of RDF data. Many databases are supported, both Open Source and proprietary. An SDB store can be accessed and managed with the provided command line scripts and via the Jena API. Use of SDB for new applications is not recommended. This component is "maintenance only". However, TDB is faster, more scalable and better supported than SDB.

7.1.2 Implementation Details

The Ontology API is designed for the purposes of the COMPOSITION project. It is the component which enables the access of Marketplace Agents into the knowledge base. As described at section 4 Agents components should be able to connect with Collaborative Manufacturing Services Ontology which is the knowledge base of COMPOSITION Ecosystem. So, this component is implemented to cover these needs and to offer the expected functionality.

7.1.2.1 Requirements

Based on COMPOSITION project use cases and requirements, and the COMPOSITION system's proposed architecture the following main requirements were set for Ontology API implementation:

- The API should be connected with COMPOSITION's Collaborative Manufacturing Services Ontology
- The API should be offer the following services
    - Add instances to ontology
    - Update instances to ontology
    - Read instances from ontology
    - Remove instances from ontology
- The API should be able to connect with other COMPOSITION components in order to offers the previous services
- The connection should be based on communication protocols and formats accepted from COMPOSITION system's architecture
- It should be well designed and be compatible with project's quality control
- It should be designed in a way to be easily extended and maintained
- It should cover the security requirements of the project and be compliant with the Semantic Framework from WP4

7.1.2.2 Technologies and Tools

The technologies which are used for Ontology API's development are described in this sub-section. Their selection is indicated by the two basic factors:

- Address the requirements were described above
- Use open and free technologies and tools as the project mention to do in DoA

The main selected technologies are the following:

Java was selected as the implementation language. It is a general purpose, object oriented programming language. Java is one of the most popular programming languages in use, especially for client server web applications.

Web Services as defined by World Wide Web Consortium is a system designed to support interoperable Machine to Machine interaction over a network. Web services are server applications which can process and exchange data. They are selected as a perfect match to represent the required services.

REST or Representational State Transfer was selected as the architectural style of web services. REST offers better performance, modifiability and scalability to enable web services to work better on the Web. The REST architecture style is a client/server architecture where clients and servers exchange representations of resources by using a standardized interface and protocol. Resources are accessed using Uniform Resource Identifiers (URIs) which are the typical links on the Web.

HTTP stands for HyperText Transfer Protocol and was the selected protocol to be used by the RESTful API. This application protocol is used to link pages of hypertext and it is a way to transfer files. HTTP is the foundation of data communication for the Web.

JSON or JavaScript Object Notation was the selected syntax format for exchanging messages. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers. Also, it is easy for machines to parse and generate this format. These properties make JSON an ideal format for data-exchange.

Apache Jena was selected as the Java framework API to support COMPOSITION's Ontology API. As mentioned before, it is a free and open source tool which supports OWL and RDF languages and offers querying and storing capabilities. All this, consist Jena framework as the perfect tool for our implementation.

SPARQL was selected as the query language. It is a semantic query language able to manipulate and retrieve data stored in RDF format. It is standardized of the World Wide Web Consortium, and is recognized as one of the key technologies of the semantic web.

Eclipse[8] IDE is a well-known Java Integrated Development Environment. It is the most widely used Java IDE and contains a basic workspace and large variety of plug-ins. The Eclipse IDE for Java EE Developers was the selected package. It offers tools for Java EE and Web applications development and includes many features such as Eclipse Git Team Provider, Maven Integration for Eclipse etc.

Apache Tomcat[9] was the selected web server environment. It is an open-source Java Servlet Container developed by the Apache Software Foundation. It provides an HTTP web server environment in which Java code can run.

7.1.2.3 Implementation

The implementation of the COMPOSITION Ontology API was based in the previous mentioned technologies and tools. The target of the implementation was the development of software which will be able to fulfil the previous page's requirements.

The implementation's architecture was defined in order to be able to support the following processes:

- The OWL files from Collaborative Manufacturing Services Ontology should be stored in a permanent, scalable and high performance store
- A COMPOSITION Agent sends its request via HTTP and JSON format for message description. The JSON format is similar to Communication eXchange Language (CXL) of Agents.
- The Ontology API uses Jena API to access the permanent store
- Then the JSON message is translated in a SPARQL query by the Query Engine that back-ends the API
- The SPARQL query is applied to the ontology store
- The Ontology API sends back to the component an HTTP response in JSON format in a format similar to CXL

The next figure presents a high-level overview of the architecture design of the complete semantic framework contains the Ontology, the Ontology API and the Semantic Matchmaker:

[8] https://www.eclipse.org/ide/
[9] http://tomcat.apache.org/

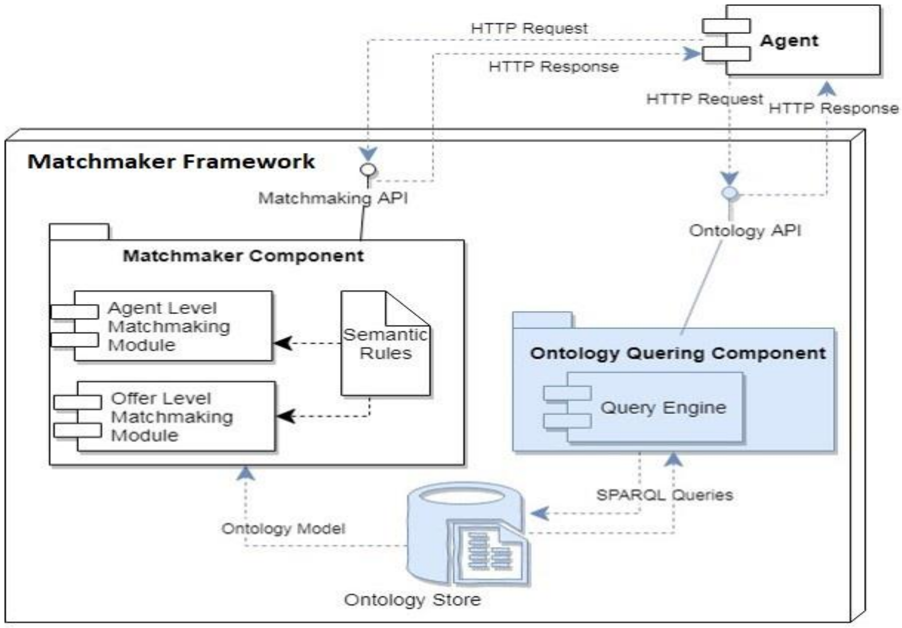

Figure 25: COMPOSITION Semantic Framework high-level architecture overview

Updates on Architecture Design and Lessons Learnt

As depicted in the previous figure, the Ontology Querying Component and its exposed API are parts of the Matchmaker block and they are not a standalone application as in the early stages of the project and the first version of this document. Firstly, they were considered as two different components. The Matchmaker planned to call Ontology API services in order to access some of the ontological resources. However, the need to share the same resources fast and effectively, for both querying and inference, indicates the design of a complete Semantic Framework. This framework offers, in a common way, to the COMPOSITION Marketplace storing, querying and reasoning capabilities.

Key Steps and Designing Approaches during the Development Phase

- Eclipse was selected as the IDE for building the semantic framework
- It was created as Maven project in order to configure effortlessly package project's dependencies such as Jena library, Jersey library etc.
- The first development activity was the handling of the OWL files which have been created at Protégé tool. The OWL files which consist the Collaborative Manufacturing Services Ontology were stored in memory as OntModel using Jena API functionalities
- The next step was the creation of the Ontology Store. The OntModel stored in a permanent store. Two cases were examined based on Jena API. The first was the usage of SDB store which is a SQL database store. The second was the usage of TDB component for storing. The second approach was selected. As native triple store the TDB is faster, more scalable and better supported than SDB store. The SDB store is backed by SQL, so queries from SPARQL have to "turn" into SQL queries. This adds complexity and it is not as efficient as a native triple store. The benefits of using a permanent triple store are that all the queries are applied in the Model which is stored in the tuple space. Therefore, every creation or deletion of individuals takes place at this Model. This means that the original OWL files are not modified.
- A set of SPARQL queries was created based on terms of Collaborative Manufacturing Ontology and Language. All the queries are located in a common directory as .sparql files and they were not created as Strings inside the source code in order to be easier to modify and extend them.
- The necessary functionalities for the Querying Engine has been designed in order to be able to map JSON messages from the Agents to SPARQL queries. Then, the SPARQL API form Jena was used in order to enable the querying of the Ontology Store.
- Last, web services/interfaces were created. Every service as soon as it receives a request, calls the Ontology Querying Engine to handle the request. The engine maps the services input to SPAQL

queries. Then, it is connected to the permanent store and applies the SPARQL queries. Then the queries' result is send back to web services to handle the response.

7.2 Supported Interfaces

The supported web interfaces of the Ontology API are listed below:

POST	/setMarketplaceCompany	Add a new company to Marketplace
POST	/setMarketplaceService	Add a new service to Marketplace
POST	/setServiceOperation	Add a new operation to a service
POST	/setRawMaterial	Add a new raw material
POST	/setMaterialResource	Add a new resource such as machine or tool
GET	/initializeDBWithOntology	Initialize the store with Collaborative Manufacturing Services Ontology
GET	/getMarketplaceCompanies	Get the companies which participate at the Marketplace
GET	/getMarketplaceServices	Get all the services which are described at the Marketplace
GET	/getCompanyDetails	Get all company details such as name, system rating etc.
GET	/getServicesByCompany	Get all the services offered from a specific company
GET	/getOperationsByService	Get all the operations supportred from a specific service
DELETE	/deleteCompany	Delete a company from Marketplace
DELETE	/deleteService	Delete a Marketplace service
DELETE	/deleteOperation	Delete an operation
DELETE	/deleteRawMaterial	Delete a material
DELETE	/deleteMaterialResource	Delete a resource

Figure 26: Ontology API Interfaces

8. COMPOSITION Ontology's Quality Control, Deployment and Security

8.1 Quality Control

A quality control plan has been followed during the development processes of both Collaborative Manufacturing Services Ontology and the corresponding API. This plan alongside with the methodology was followed are factors that indicate the quality of the implemented components.

8.1.1 Collaborative Manufacturing Services Ontology

The steps that followed for building the Collaborative Manufacturing Services Ontology reflect some of its quality and they are mentioned below:

- A thorough analysis of ontology languages and tools has been presented in Section 5
- Selection of OWL 2.0 as ontology language and Protégé as the implementation tool after the evaluation of previous mentioned analysis
- Selection was done after an analysis and based on project's needs, use cases and requirements the domain that the ontology should describe
- Import well-known and widely used ontologies of domains of manufacturing and e-commerce. This ensures quality and enriches ontology with the demanded classes, properties and structures for these domains' description
- A thorough analysis of ontology building methodologies and the building of Collaborative Manufacturing Services Ontology following NeOn methodology (Sections 5 and 6)
- Evaluation of the developed ontology using the open source tool, OntOlogy Pitfall Scanner [10] (OOPS) to check for crucial errors. This tool analyses the RDF code and offers warnings for a large variety of possible pitfalls. The produced warnings were manually inspected in order to determine which of them correspond to actual bugs that require fix, and which are just false alarms (i.e. false positives) After a first evaluation we focused on the critical pitfalls that could affect the ontology's consistency, reasoning, and applicability. Also possible important pitfalls about missing domain or range in properties, untyped properties and classes are handled too. On the other hand, a good number of the produced alerts were false positives, and thus they did not require any corrective action. They most important of them were related in possible wrong equivalent classes. They are not considered as real threats as the tool tried to check the equality of some classes of the new ontology with the original classes of imported ontologies as it found them online using their URIs. However these classes had been re-engineered in the current ontology and the comparison with the original ones has no meaning as they were never used.

Besides the previous steps that indicate the quality of the implemented ontology, the ontology tested in both pilot cases that it is involved, KLE-4 and KLE-7, and it was defined that contains the required concepts that enable the representation of the agents in the Marketplace for this cases. Moreover, it contains all the necessary means for the Matchmaker. By using this Ontology and its concepts, the Matchmaker was able to perform matching and offers' evaluation for both cases. Of course, the Matchmaker task is still in progress and may lead to further updates on the Ontology.

8.1.2 Ontology API

During the implementation phase of COMPOSITION Ontology APIs, the quality control was focused on general software quality criteria, the overall COMPOSITION system architecture's compatibility, and the deliverables D1.1 Project Quality Control Plan I and Project Quality Control plan II of COMPOSITION project. More precisely the quality plan consists of the following factors:

- Identification of the Ontology API requirements
- Analysis of existing technologies and adoption of the best suitable with the COMPOSTITION system's architecture. Use of REST web services and JSON format for messages exchange as both technologies have defined as supported by COMPOSITION architecture at D2.3-The COMPOSITION

[10] http://oops.linkeddata.es/

architecture specification I and D2.4-The COMPOSITION architecture specification II. These will ensure Ontology API's compatibility with other project's components.

- Use of software tools which were proposed at D1.1 Project Quality Control Plan I and D1.2 Project Quality Control Plan II in order to support quality of software:
    - Use of Eclipse IDE as the development environment
    - Use of Git for control versioning (actually EGit plugin from Eclipse IDE)
    - Use of Maven as build tool for dependency management and build of source code
- Test procedures were applied. For software quality assurance both static and dynamic analysis techniques applied:

Static analysis

In static analysis the PMD[11] tool was used. It is an open source tool which offers source code analysis. It is able to detect possible bugs, empty statements, unused variables and methods, duplicate code, classes with high cyclomatic complexity, etc., by offering built-in sets of rules. The tool categorizes the possible problems as violations distributed in 5 categories based on priority: block, critical, urgent, important and warning

During Ontology API, which is part of the Matchmaker package, development process the code was checked for the rules sets which described at the Annex II. About 300 rules were used in different cycles of the development process, the analysis results were evaluated during these faces, and the most important were handled. At the current version of code there are no block, critical, important and warning violations. There are only few urgent violations which are related to excessively long variable names, multi occurrences of some string literals, variables with short names, etc. These violations are considered as false positives and there was no further action in order to fix them.

Dynamic analysis

In dynamic analysis, tests in runtime have been executed. For the project purposes, Unit tests, Integration tests and System tests have been executed.

Automated tests have been built in a Test source code package which was created by Maven. The TestCase class from JUnit was extended and member functions were added. Every function represents a test of a supported web service. The tests are able to be executed without deploying the component in a server and using an external HTTP client. Eclipse Jetty server which provides a Web server and javax servlet container was used. So, the test cases deployed and executed using Jetty. This provided us fast execution and testing of the source code without the need to deploy the project to an external server in order to test every change in the code.

As we mentioned before, a test for every supported web service in Figure 26 has been created. Then we call every function which contained a test and check if we got the expected output at Eclipse's console. The tests were called separately or in combination. For example we had called a test to check if we can get all the companies. After we called a service to delete a company and then called again a service to get all companies, in order to decide if the deletion was executed properly. We executed plenty of these scenarios and combinations.

After the development of the Matchmaker package contains the Ontology API, it was deployed in Apache Tomcat container. Then all the available web services and the previous test cases were executed using Postman[12] Rest Client.

Furthermore, integration units test with the Marketplace Agents has been executed. After the deployment of the Matchmaker framework, the deployed Agents calls all the services exposed by Ontology API in order to test the connectivity and the correct operation. Complete system tests are also executed as Security Framework is also adopted in the communication of the Ontology API and Agents.

[11] https://pmd.github.io/
[12] https://www.getpostman.com/

8.1.3 Scalability of Ontology and Ontology API

The COMPOSITION Matchmaker package and the included Ontology and its API have been designed in order to offer high performance and support large Marketplaces with numerous of participants and services.

As the Matchmaker framework is packaged and deployed in an Apache Tomcat server, the maximum number of connections that this component can access and process depends on Tomcat web server configuration. Based on official Apache Tomcat 8 Configuration [13] the server is able to support over than 8000 connections.

Furthermore, a RDF-triple store is used as the data store of the Marketplace. Based on the COMPOSITION project's pilot partners and use cases there was no need for a big data store for the Marketplace. However, in order to create a Marketplace that can be used beyond the project, triple-store was used. TDB was the triple store, as native triple store is fast, and supports the storage of millions of individuals. Using TDB every change at the ontology takes place at an ontology model stored in the file system leaving the original ontology immutable. This means that the original version of the ontology can be used in order to initialize new Marketplaces.

The performance of the Matchmaker package and its included components was tested for the COMPOSITION use cases such as UC KLE-4 and the online bidding process. The package services response in a reasonable time(less than 5 seconds). However, in order to examine the performance of all sub-components in large Marketplaces, automated JUnit tests were created and applied. Over 20.000 companies and services created and added to the Marketplace Ontology Store. Then some queries were applied and the responses were still in reasonable time (near 5 seconds). Only in the case that the instances were created simultaneously the required response were some minutes. But this is not consider as a serious problem as the Marketplaces was initialized ones and after that every new instance is added as soon as a new company arrives at the Marketplace or offers a new service etc.

8.2 Deployment

The semantic framework of COMPOSITION was deployed as a Docker image in alignment with the rest of the project's components based on the Deployment View of D2.4 The COMPOSITION architecture specification II.

Docker is an open-source project aiming at automating the deployment of applications as portable, self-sufficient containers that can run virtually anywhere, on any kind of server. It can be considered as a lightweight alternative to full machine virtualization provided by hypervisors. While in the traditional hypervisor approaches each virtual machine (VM) needs its own operating system, in Docker applications operate inside a container that resides on a single host operating system that can serve many different containers at the same time.

The Matchmaker package's Docker image contains all the sub-components as it is described at Figure 1 of chapter 4. So in this image the Rule-based Matchmaker, the Querying Engine, the Ontology Store and their corresponding APIs are containing.

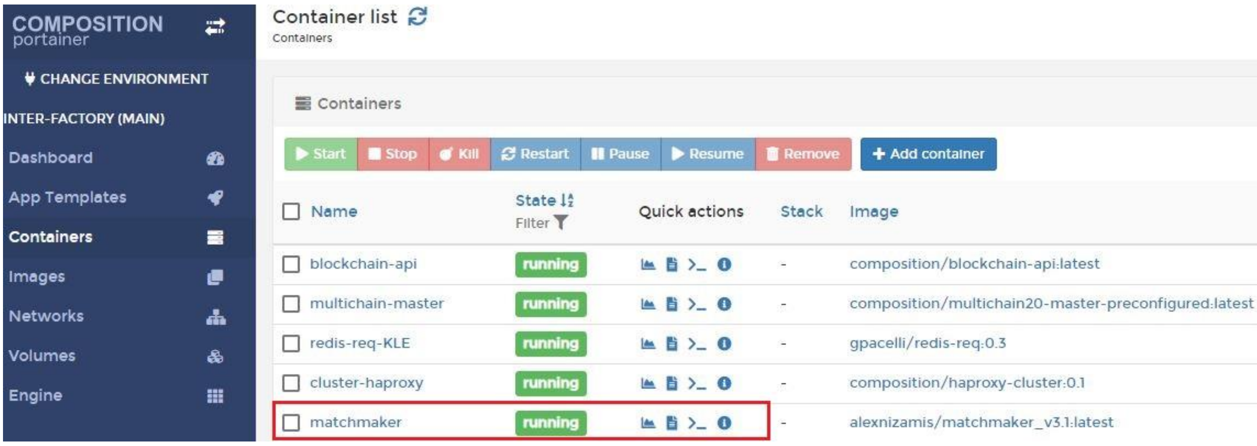

Figure 27: Deployment of Matchmaker Package on COMPOSTITION Production Server

In order to create the Matchmaker's Docker image and the corresponding container, the official Docker image for Apache Tomcat was used. Tomcat was selected as the web server environment as it is web server

[13] http://tomcat.apache.org/tomcat-8.5-doc/config/http.html

environment in which Matchmaker package's Java code can run. So, for the creation of the aforementioned Docker image the Web Application Resource file from the Matchmaker was added to the Tomcat's image. The corresponding Docker container of the Matchmaker image was deployed at the COMPOSITION production server and more precisely, on inter-factory Portainer[14], which offers management of Docker environments.

8.3 Ontology and Security Framework

COMPOSITION Collaborative Manufacturing Services Ontology and its exposed APIs should be secured and compatible with the requirements of the project's Security Framework from WP4.

Generally, all COMPOSITION components, which expose RESTful APIs over the internet, must enforce authentication using OpenID Connect. The LinkSmart® Border Gateway (BGW)[15] can secure these APIs such as the Ontology API from Matchmaker package by providing an overlay on top of all RESTful APIs, passing only authenticated and authorized requests to them.

A Basic Auth authentication will be used in order to secure the Matchmaker API's(including Ontology API) end points. For the COMPOSITION purposes:

- User provides username/password in the REST request
- BGW intercepts the request and negotiates with an OpenID Connect server for a token
- If authenticated, BGW forwards the request to API and caches the token for upcoming requests until it expires

Furthermore, COMPOSITION Security Framework also supports authorization services. BGW is able to enforce policy based authorization based on request path and HTTP methods. The policies are profile attributes assigned to users and groups as part of their accounts in the OpenID Connect server. For a component, such as a Marketplace Agent, that wants to have access on Ontology it should ask to be able to access the following component, method and resource:

- GET: https://inter.composition-ecosystem.eu/matchmaker/#
- POST: https://inter.composition-ecosystem.eu/matchmaker/#

The above links indicates to Keycloak[16] framework that a component is authorized to call both GET and POST services, which is under the Matchmaker package.

[14] https://www.portainer.io/
[15] https://docs.linksmart.eu/display/BGW
[16] https://www.keycloak.org/

9. Conclusions

In conclusion, this deliverable describes the total effort spent from M5 to M30 and represents the outcomes of Task 6.4-Collaborative Manufacturing Services Ontology and Language of WP6. More specifically, this report documents the delivered COMPOSITION Ontology and its corresponded API.

Collaborative Manufacturing Services Ontology has been implemented and presented after a thorough analysis of Ontology languages, methodologies and tools. Moreover, ontologies from the domains of manufacturing and e-commerce were studied and MASON, MSDL and GoodRelations Language were selected to be imported to COMPOSITION Ontology. By using these ontologies and by following NeOn methodology a new ontology was created in OWL language using the Protégé tool. The implemented Collaborative Manufacturing Services Ontology is able to describe both the supply/demand entities and the manufacturing domain's services and resources. The Ontology has further extended with concepts and means form waste management domain and software solutions domain for manufacturing.

An Ontology API has also been implemented. After consideration of the project's requirements and architecture, and after an analysis of available technologies and tools, the Ontology API is developed in Java and it is offered through RESTful web services. It provides a set of services which offers retrieving and storing functionalities from and to ontology store as well to the Marketplace agents.

The results presented in this deliverable mainly affect WP6 and its components such as the Agents and the Matchmaker. In particular, the Matchmaker's functionality is completely depended from Collaborative Manufacturing Services Ontology as the Matchmaker performs matching by applying rules to the ontology. In addition, the agents use as knowledge base the Ontology and they are able to read and store data to the Ontology Store. This deliverable is also connected with WP3 and its modelling tasks as the ontology illustrates some intra-factory information such as manufacturing operations and resources, to the Marketplace. This task is also connected, as the most of the project, with WP4 and the Security Framework.

Finally, as it is perceived, the results of Task 6.4 are presented in this deliverable. The main outcome of this task, which also consists its unique selling point, is the Collaborative Manufacturing Services Ontology that is able to describe most of the concepts in a real-world Manufacturing Marketplace. The proposed ontology offers the concepts for the description and modelling of manufacturing domain connected with supply-chain, waste management and software solutions domains in order to support a complete ecosystem focused on manufacturing.

10. List of Figures and Tables

10.1 Figures

Figure 1: COMPOSITION Marketplace components ... 7
Figure 2: Core Classes of MSDL (Ameri, 2006) ... 16
Figure 3: MASON main classes and properties (Lemaignan, 2006) ... 17
Figure 4: GoodRelations Language main classes and properties ... 18
Figure 5: Set of nine scenarios for building ontologies and ontology networks (M.C. Sua´rez-Figueroa, 2012) ... 20
Figure 6: COMPOSITION Collaborative Manufacturing Services Ontology's Class Overview ... 26
Figure 7: "Business entity" class and sub-classes ... 26
Figure 8: "Capability" class and sub-classes ... 28
Figure 9: "Dates and Times" class and sub-classes ... 28
Figure 10: "Delivery method" class and sub-classes ... 29
Figure 11: "Entity" class and sub-classes ... 30
Figure 12: "Generic term" class and sub-classes ... 31
Figure 13: Mapping of vendor specific concepts ... 31
Figure 14: "Operation" class and sub-classes ... 34
Figure 15: "Payment method" class and sub-classes ... 35
Figure 16: "Price specification" class and sub-classes ... 35
Figure 17: "Quantitative value" class and sub-classes ... 36
Figure 18: "Resource" class and sub-classes ... 37
Figure 19: "Service" class and sub-classes ... 39
Figure 20: "Supporting service" class and sub-classes ... 40
Figure 21: "Supporting system" class and sub-classes ... 40
Figure 22: "Warranty" class and sub-classes ... 41
Figure 23: Modelling of KLEEMANN Polishing Procedure using Collaborative Manufacturing Services Ontology ... 42
Figure 24: Apache Jena's framework architecture (Apache Jena, 2017) ... 43
Figure 25: COMPOSITION Semantic Framework high-level architecture overview ... 47
Figure 26: Ontology API Interfaces ... 48
Figure 27: Deployment of Matchmaker Package on COMPOSTITION Production Server ... 51
Figure 28: Class hierarchy view ... 58

10.2 Tables

Table 1: Abbreviations and acronyms are used in this deliverable ... 5
Table 2: MSDL and MASON overlapping classes' alignment ... 22
Table 3: MSDL and GoodRelations Language overlapping classes' alignment ... 22
Table 4: ORSD of COMPOSITION Collaborative Manufacturing Services Ontology ... 25
Table 5: Object Properties of "Business entity" class ... 27
Table 6: Data Properties of "Business entity" class ... 27
Table 7: Data Properties of "Business entity type" class ... 27
Table 8: Data Properties of "Capability" class ... 28
Table 9: Object Properties of "Dates and Times" class ... 29
Table 10: Data Properties of "Dates and Times" class ... 29
Table 11: Data Properties of "Delivery method" class ... 29
Table 12: Object Properties of "Entity" class ... 30
Table 13: Data Properties of "Entity" class ... 30
Table 14: Object Properties of "Offer" class ... 31
Table 15: Data Properties of "Offer" class ... 32
Table 16: Object Properties of "Operation" class ... 33
Table 17: Data Properties of "Operation" class ... 33
Table 18: Data Properties of "Payment method" class ... 35
Table 19: Object Properties of "Price specification" class ... 35
Table 20: Data Properties of "Price specification" class ... 35
Table 21: Data Properties of "Quantitative value" class ... 36

Table 22: Object Properties of "Resource" class ... 38
Table 23: Data Properties of "Resource" class ... 38
Table 24: Data Properties of "Certification" class ... 38
Table 25: Object Properties of "Service" class ... 39
Table 26: Data Properties of "Service" class ... 39
Table 27: Object Properties of "Supporting service" class ... 40
Table 28: Data Properties of "Supporting service" class ... 40
Table 29: Object Properties of "Supporting system" class ... 40
Table 30: Data Properties of "Supporting system" class ... 41
Table 31: Object Properties of "Warranty" class ... 41
Table 32: Data Properties of "Warranty" class ... 41
Table 33: Static analysis' rules set ... 59

11. References

(COMPOSITION, 2016)GRANT AGREEMENT 723145 — COMPOSITION: Annex 1 Research and innovation action

(Ameri, 2006) Manufacturing Service Description Language https://www.researchgate.net/publication/267486591_An_Upper_Ontology_for_Manufacturing_Service_Description

(Lemaignan, 2006) Manufacturing's Semantics Ontology or MASON is a manufacturing ontology, aimed to provide a common semantic net in manufacturing domain. http://ieeexplore.ieee.org/document/1633441/

(GoodRelations, 2017) GoodRelations Language, The Web Vocabulary for E-commerce http://www.heppnetz.de/projects/goodrelations/

(Studer et al, 1998) Studer, R., Benjamins, V. R., & Fensel, D. (1998). "Knowledge engineering: principles and methods", Page(s): 161-197.

(Gruber, 1993) Gruber, T. R. (1993). "A translation approach to portable ontology specifications. Knowledge acquisition", Page(s): 199-220.

(Genesereth and Fikes, 1992) Genesereth, M. R., & Fikes, R. E. (1992). "Knowledge interchange format-version 3.0: reference manual." Computer Science Department, Stanford University, Technical Report Logic-9201, June 1992.

(MacGregor, 1992) MacGregor, R, 1991. "The Evolving Technology of Classification-Based Knowledge Representation Systems. In Principles of Semantic Networks: Explorations in the Representation of Knowledge", edited by J. Sowa. San Mateo, CA: Morgan Kaufmann., San Mateo, California.

(Chaudhri et al., 1998) Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., & Rice, J. P. (1998). "OKBC: A programmatic foundation for knowledge base interoperability". In Innovative Applications of Artificial Intelligence Conference, Page(s): 600-607.

(Motta, 1999) Domingue, J., Motta, E., & Garcia, O. C. (1999). "Knowledge modelling in webonto and ocml: A user guide.", Knowledge Media Institute, The Open University.

(Kifer et al., 1995) Kifer, M., Lausen, G., & Wu, J. (1995). "Logical foundations of object-oriented and frame-based languages.", Journal of the ACM (JACM), 42(4), Page(s): 741-843.

(Luke and Heflin, 2000) Heflin, J., Hendler, J. A., & Luke, S. (2003). "SHOE: A Blueprint for the Semantic Web. Spinning the Semantic Web", Page(s): 1-19.

(Karp et al., 1999) Karp, P. D., Chaudhri, V. K., & Thomere, J. (1999). "XOL: An XML-based ontology exchange language.", Pangea Systems Inc., Artificial Intelligence Center.

(Lassila and Swick, 1999) Lassila, O., & Swick, R. R. (1999). "Resource description framework (RDF) model and syntax specification."

(Brickley and Guha, 2003) Brickley, D., & Guha, R. V. (2003). "Resource description framework (rdf) schema specification 1.0: Rdf schema.", W3C working Draft.

(Fensel et al., 2001) Fensel, D., Van Harmelen, F., Horrocks, I., McGuinness, D. L., & Patel-Schneider, P. F. (2001). "OIL: An ontology infrastructure for the semantic web.", IEEE intelligent systems, Page(s): 38-45.

(Dean and Schreiber, 2003) Dean, M., Schreiber, G., van Harmelen, F., Hendler, J., Horrocks, I.,McGuinness. (2003). "OWL web ontology language reference."

(E. Prud'hommeaux et al, 2008) Prud'hommeaux, E., Seaborne, A.. (2008). "SPARQL query Language for RDF", W3C Recommendation, January 15, 2008.

(Gómez-Pérez et al., 1997) Fernández-López, M., Gómez-Pérez, A., & Juristo, N. (1997). "Methontology: from ontological art towards ontological engineering."

(Waterman, 1986) Waterman, D. A. (1986). "A guide to expert systems. Addison-Wesley."

(M. C. Suárez-Figueroa, 2010) NeOn Methodology for Building Ontology Networks: Specification, Scheduling and Reuse

(Staab et al., 2001) Maedche, A., & Staab, S. (2001). “Ontology learning for the semantic web.”, IEEE Intelligent systems, Page(s): 72-79.

(Apache Jena, 2017) A free and open source Java framework for building Semantic Web and Linked Data applications http://jena.apache.org/index.html

(L. Halilaj, 2016) L. Halilaj, N. Petersen, I. Grangel-González, Ch. Lange, S. Auer, G. Coskun, S. Lohmann: VoCol: An Integrated Environment to Support Version-Controlled Vocabulary Development. Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2016), pp. 303-319, LNAI vol. 10024, Springer, 2016

12. ANNEX I

Usage instructions for Collaborative Manufacturing Services Ontology

Using OWLDoc

A user can explore Collaborative Manufacturing Services Ontology in a web browser by using the offered OWLDoc:

1. Download the *OntologyV02.zip* file and unzip it to a location of your choice.
2. Navigate to *OntologyV02 -> Documentation*
3. Select the file named *index*
4. Then open this file with a double click. A web browser window will launch where the user will be able to explore ontology's details

Using Protégé tool

In this section, we present instructions in order to open and properly use the current version of the Collaborative Manufacturing Services Ontology through the Protégé tool:

1. Download the *OntologyV02.zip tar* file and unzip it to a location of your choice.
2. Download, install and launch Protégé tool (preferred versions 4.2, 4.3 and 5.2)
3. Select File at the top line menu and then select Open at the sub-menu has just been appeared
4. At the new window, navigate to the extracted file from Step 1, *OntologyV02-> Ontology - Files* and choose *COMPOSITIONv02.owl*. Then press Open button
5. Protégé will load the ontology that is contained in the owl file. Protégé will also import the three imported ontologies. Check that the other three OWL files are also located in the extracted file from Step 1. After this step, the user is ready to visualize the whole Ontology

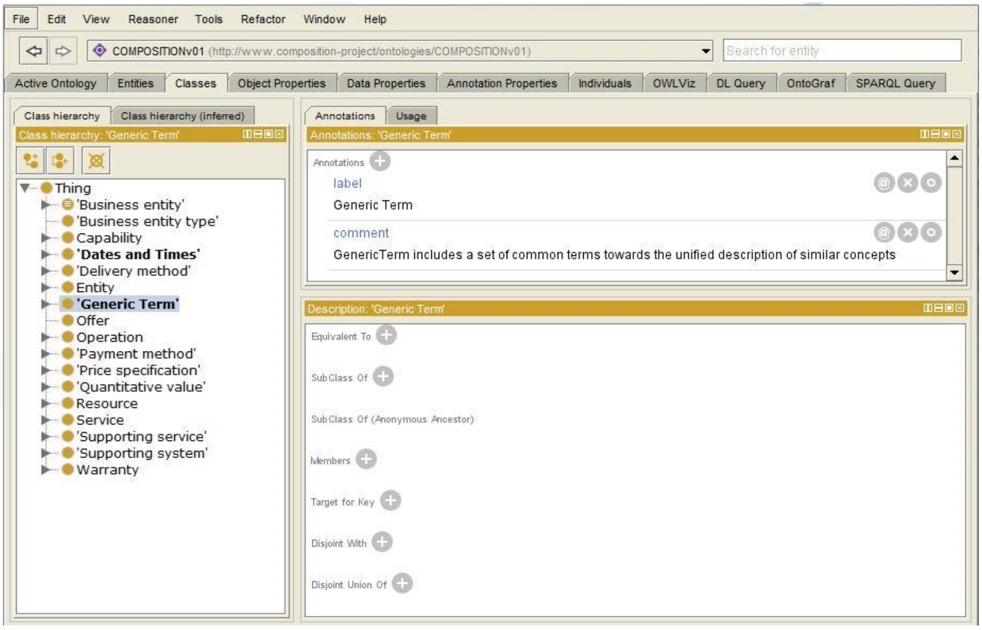

Figure 28: Class hierarchy view

** The ontology will be updated by the end of the project. Agents add individuals and some new concepts will need to be extended for matchmaking etc. Therefore, the last version will be available by the end of the project*

13. ANNEX II

Rule set was used for PMD static analysis

Table 33: Static analysis' rules set

Rules set	Description
Basic	A collection of good practices which everyone should follow
Basic POM	Rules related with dependency management
Braces	Contains a collection of braces rules
Code size	Ruleset contains a collection of rules that find code size related problems
Complexity	Contains a collection of rules related to code's complexity
Controversial	Contains rules that, for whatever reason, are considered controversial.
Design	A collection of rules that find questionable designs
Empty code	A collection of rules that find blocks of code where nothing is done
Import statements	Ruleset to deal with different problems that can occur with a class' import statements
J2EE	Rules related to J2EE
JUnit	Rules related to problems that can occur with JUnit tests
Naming	Contains a collection of rules about names - too long, too short etc.
Optimization	Ruleset deals with different optimizations that generally apply to performance best practices
Security code guidelines	Contains rules which check the security guidelines
Strict Exceptions	Contains strict guidelines about throwing and catching exceptions
String & StringBuffer	Contains rules related with manipulation of the class String or StringBuffer
Style	Ruleset related to name conventions
Unnecessary	Ruleset that find unnecessary blocks
Unused code	Contains rules that find unused code