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

D6.10 COMPOSITION Brokering and Matchmaking Components II

Date: 2019-06-28

Version 1.0

Published by the COMPOSITION Consortium

Dissemination Level: Public

Co-funded by the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement No 723145

Document control page

Document file: D6.10 COMPOSITION Brokering and Matchmaking Components II v1.0.docx
Document version: 1.0
Document owner: CNET - CERTH

Work package: WP6
Task: T6.5
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	Alexandros Nizamis(CERTH), Mathias Axling (CNET)	2019-05-16	Initial TOC based on D6.9
0.2	Mathias Axling(CNET), Christos Ntinas (CERTH)	2019-06-02	Chapter 4 and Chapter 5 Input
0.3	Alexandros Nizamis, Nikolaos Vakakis (CERTH)	2019-06-10	Chapter 6 and Chapter 7 Input
0.4	Nikolaos Alexopoulos, Leonidas Samaras (CERTH), Mathias Axling(CNET)	2019-06-12	Chapter 8 and Chapter 9 Input
0.5	Alexandros Nizamis, Nikolaos Vakakis (CERTH)	2019-06-18	Chapter 5 Input
0.6	Alexandros Nizamis(CERTH), Mathias Axling (CNET)	2019-06-21	Introduction, Conclusions and Document preparation for internal review
1.0	Alexandros Nizamis, Dimosthenis Ioannidis(CERTH), Mathias Axling(CNET)	2019-06-27	Final Improvements after peer review – Ready for final submission

Internal review history:

Reviewed by	Date	Summary of comments
Jannis Warnat (FIT-UC²)	2019-06-25	Some minor corrections of typos etc.
Vasiliki Charisi (ATL)	2019-06-25	Some minor comments and typos

Index:

1 Executive Summary

This report describes the results of Task 6.5 Brokering and Matchmaking for Efficient Management of Manufacturing Processes from M5 to M34. The Matchmaker is a core component of the COMPOSITION Collaborative Ecosystem, providing matching of buyers and sellers in the supply chain, based on services and their capabilities. Moreover, the Matchmaker provides a ranking of offers during marketplace agents' negotiations.

To this end, semantic matching of manufacturing capabilities and marketplace related services is applied to find the best possible supplier to fulfil a request for a service, raw materials or products involved in the supply chain. The work has done in this task mainly affects the WP6 components such as the Marketplace Agents. Moreover, the Matchmaker functionality is exclusively depended on Collaborative Manufacturing Services Ontology that was implemented in the same WP. Furthermore, as the Matchmaker is offered through RESTful services it is connected with Security Framework of WP4.

Different decision criteria for supplier selection according to several qualitative and quantitative factors are considered (e.g. delivery time, distance, due date, quality, price, technical capability, past performance, payment methods and terms, etc.). Special focus was given in dealing with the trade-off between performance and quality of matching, in order to provide responses in a reasonable time while at the same time minimization of computational complexities will be targeted. In order to infer new knowledge and provide matching between requesters and providers, semantic rules are applied in an ontology, which is used as the knowledge base for the COMPOSITION ecosystem. Regarding the estimation of similarity among offers and requests, as well as the evaluation of them, well-established weighted algorithms and metrics are used alongside with the semantic rules in order to address the objectives of COMPOSITION Ecosystem at the best possible way.

To sum up, for Task 6.5 technologies, such as semantics and rules, were applied in a Manufacturing Marketplace for matching and evaluating offers in real-time. This was enabled by the usage of the COMPOSITION Ontology which connects manufacturing with e-commerce domain. The implemented web-based system was able to extend the usage of this Ontology. The Ontology was not used only for interoperability, but it is used also for real-time decision-making capitalizing on knowledge inference. Furthermore, the COMPOSITION Matchmaker enhance its evaluation capabilities by adopting weighted scores algorithms in order to provide a common solution for suppliers matchmaking and real-time offers evaluation. In comparison with existing frameworks which are not completely related to manufacturing domain in connection with the supply chain domain or they are exclusively designed for one system and they are not easily extended and adoptive by other ecosystems, the COMPOSITION Matchmaker is able to support a connected Manufacturing Ecosystem and it can be effortlessly transferred to other ecosystems as its services offered as web services.

2 Abbreviations and Acronyms

Acronym	Meaning
API	Application Programming Interface
AHP	Analytic Hierarchy Process
CXL	COMPOSITION eXchange Language
CFP	Call For Proposal
DLT	Deep Learning Toolkit
FITMAN	Future Internet Technologies for MANufacturing industries
FITMAN-SeMa SE	Metadata and Ontologies Semantic Matching Specific Enabler
GRDDL	Gleaning Resource Descriptions from Dialects of Languages
HTTPS	Hypertext Transfer Protocol Secure
IMPACT	Interactive Maryland Platform for Agents Collaborating Together
JSON	JavaScript Object Notation
LARKS	Language for Advertisement and Request for Knowledge Sharing
MASON	Manufacturing's Semantics Ontology
MSDL	Manufacturing Service Description Language
OWL	Web Ontology Language
RETSINA	Reusable Task Structured-based Intelligent Network Agents
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SDB	SQL DataBase
SPARQL	Simple Protocol and RDF Query Language
TDB	Triple-store DataBase
WPM	Weighted Product Model
WSM	Weighted Sum Model
WP	Working Package

3 Introduction

3.1 Purpose, Context and Scope of this Deliverable

This deliverable presents the work carried out and the results of the Task 6.5 Brokering and Matchmaking for Efficient Management of Manufacturing Processes in total. The work has been carried out in Work Package 6 (WP6), "COMPOSITION Collaborative Ecosystem". The task is tightly integrated with Task 6.4 "Collaborative manufacturing services ontology and language", the final results of which have been described in D6.8 Collaborative manufacturing services ontology and language II. This report will include an overview of the integration with the manufacturing services ontology.

Main updates from the document's previous version

This deliverable is the second and last iteration of D6.9 "COMPOSITION Brokering and Matchmaking components I", which was the first report about Task 6.5 Brokering and Matchmaking for Efficient Management of Manufacturing Processes. The main updates in the Matchmaker component's development, which are illustrated in this report, are the following:

- Matchmaker updates in order to be compatible with the Collaborative Manufacturing Services Ontology's changes. As the functionality of the Matchmaker is exclusively related to the Ontology the modification and extends in COMPOSITION Ontology led to modifications in Matchmaker's functionality and rules.
- Creation of new rules' sets based on new offered information and data descriptions in the Ontology in order to support all the negotiation scenarios of the COMPOSITION Ecosystem (UC KLE-4, UC KLE-7 and UC ATL-1).
- Enhancement of rule-based logic with weighted algorithms for more effective and flexible offers' evaluation. The logic rules were not capable to evaluate with effective way more complex scenarios such as the evaluation of raw materials offers in which the user take into consideration a lot of factors in order to select the best one.
- Development of more web services, integration with COMPOSITION Marketplace agents and Marketplace UIs
- Connection with tools such as DLT and Reputation Model in order to enhance the matching capabilities
- Scalability testing and connection with the project's Security Framework in order to secure the exposed REST endpoints.

3.2 Content and Structure of this Deliverable

The report provides an overview of the role of the Matchmaker component in the COMPOSITION system, a description of the design and interfaces of the Matchmaker and its dependencies on other components, specifically the Collaborative Manufacturing Services Ontology and Marketplace agents. The document is structured as follows:

Section 4 describes how the Matchmaker component is integrated in the overall COMPOSITION architecture and its interactions and dependencies on other COMPOSITION components. Special attention is given to interactions with the Marketplace agents and the Collaborative Manufacturing Services Ontology.

Section 5 Introduces the COMPOSITION use cases which are powered by Matchmaker component.

Section 6 includes a brief description of state-of-the-art analysis and related works presentation performed for the Matchmaker.

Section 7 provides a detailed description of the design and development of the Matchmaker with emphasis at the semantic rules and the newly added weighted algorithms.

Section 8 refers to the quality control during the component's development and the security and scalability design of the component.

Section 9 documents the Matchmaker API that is used by the COMPOSITION Marketplace agents in order to call the Matchmaker and receive its responses. Moreover, the deployment information of the Matchmaker API is drawn in this section.

Section 10 is the conclusions section which provides a summary of contents of the deliverable and lessons learned.

4 Role of Brokering and Matchmaking Components in COMPOSITION Architecture and its Main Interactions

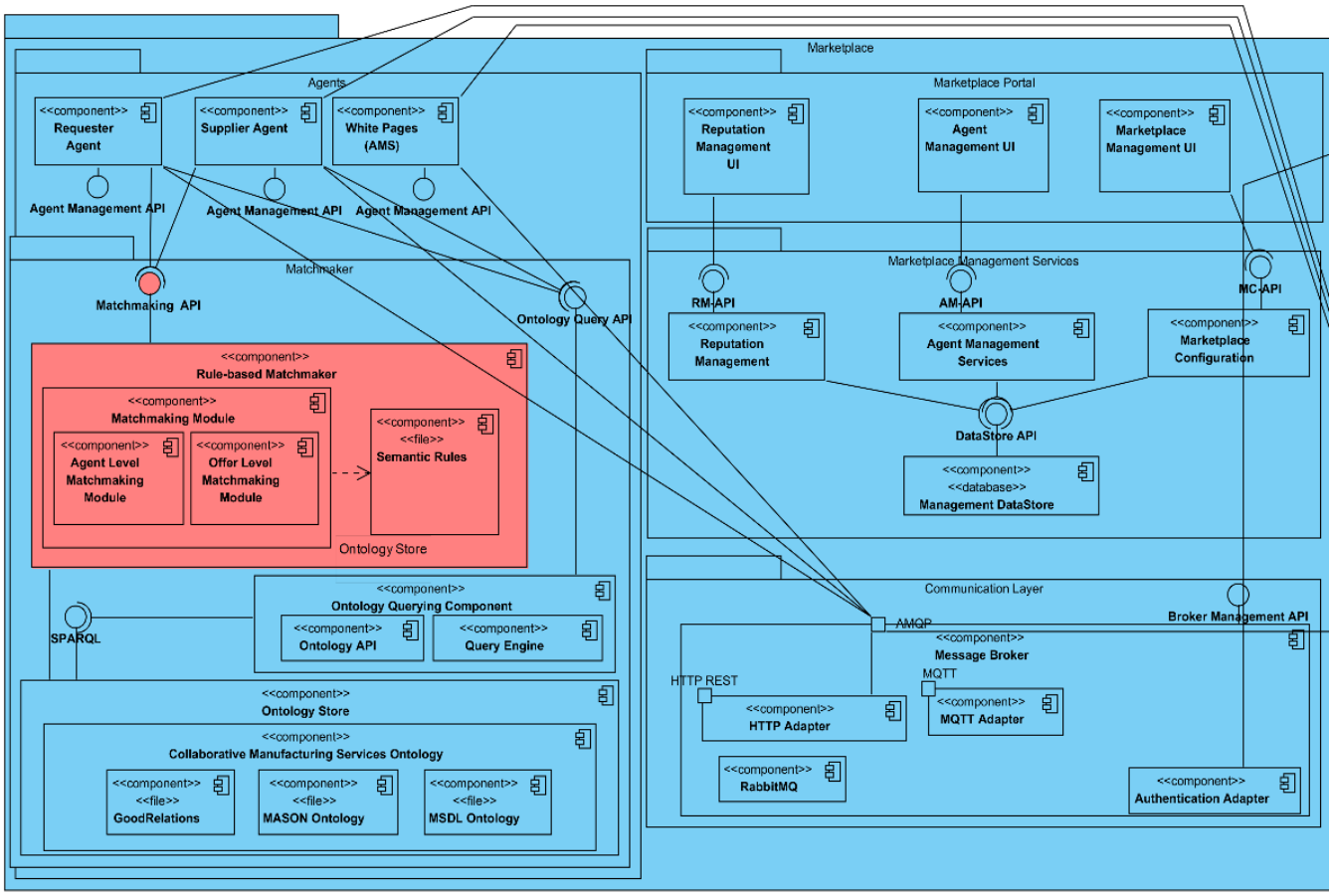

Figure 1: Matchmaker component in relation to COMPOSITION Collaborative Ecosystem architecture

As shown in the above figure, the Brokering and Matchmaking components of the Rule-based Matchmaker (in red) are part of the Matchmaker package, which also includes the Ontology Querying Component and Ontology Store. The Matchmaker package is in turn part of the Agents package.

The Marketplace's agents use the Matchmaking API to get selections of suitable suppliers for call for proposal (CFP) to be sent by a requester agent and to evaluate the offers sent by supplier agents in response to the CFP. The Ontology Query Component provides management and querying of the Collaborative Manufacturing Services Ontology, via the exposed Ontology Query API interface. The Agents can update and query the Collaborative Manufacturing Services Ontology through the interface. The Rule-based Matchmaker component is connected directly to the Ontology Store on which it will apply rules in order to infer new knowledge from the Collaborative Manufacturing Services Ontology. The rules can be applied directly at the file system which contains the Ontology Store, or they can be applied to an Ontology Model which has been loaded in the memory. The design of the Matchmaker is reported in the corresponding section, Design of Brokering and Matchmaking components. A detailed description of the Matchmaker APIs, interaction with Agents and Matchmaker deployment is provided in section Matchmaker APIs and Deployment.

The Matchmaker is involved in Use Cases *UC-KLE-4 Scrap metal collection and bidding process*, *UC-KLE-7 Ordering raw materials*, *UC-ATL-1 Searching for solutions*.

4.1 Collaborative Manufacturing Services Ontology and Language

In this sub-section a brief analysis of Collaborative Manufacturing Services Ontology and Language is presented. The COMPOSITION Matchmaker's functionalities depend exclusively on the Collaborative Manufacturing Services Ontology and Language. The Matchmaker is designed to infer new knowledge by applying rules in terms of this ontology. Collaborative Manufacturing Services Ontology is the knowledge base

for the COMPOSITION Marketplace. It is used as a common vocabulary, which offers interoperability and representation of both meanings and data. The Collaborative Manufacturing Services Ontology enables:

- The description of supply and demand entities participate in the Collaborative Ecosystem
- The description of manufacturing services, capabilities and resources for entities participate in the Collaborative Ecosystem
- The description of waste management concepts and software solutions related to a manufacturing marketplace

The Ecosystem agents will be able to make transactions as the above information will be described using this common ontology. For example an agent who requests a service or a product will be able to find a matching agent who supports this service or product based on knowledge base's information.

The next figure presents the main classes of the Collaborative Manufacturing Services Ontology and Language:

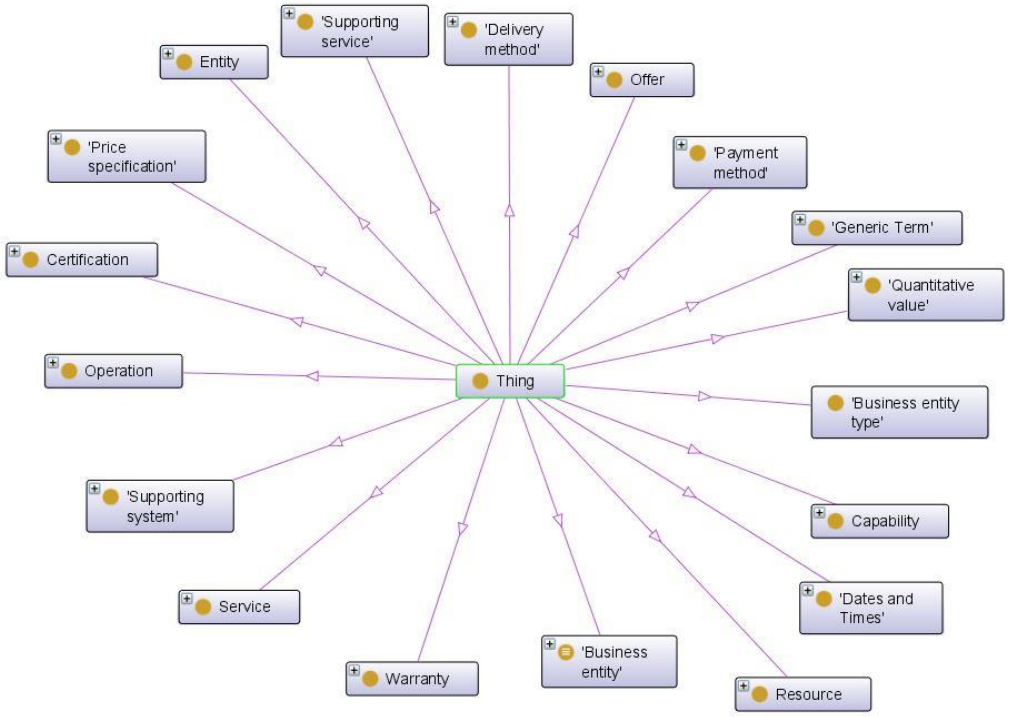

Figure 2: Collaborative Manufacturing Services Ontology Class Overview

MSDL (Ameri, 2006) and MASON (Lemaignan, 2006) ontologies 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. These imports enable for the Collaborative Manufacturing Services Ontology to represent manufacturing services and resources. Furthermore, the COMPOSITION Marketplace 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 the Marketplace. This need leads us to import the GoodRelations Language (GoodRelations Language, 2018) ontology which is one of the most well-known and widely used ontologies in e-commerce domain. All the aforementioned ontological resources were imported and re-engineered using Neon Methodology (M. C. Suárez-Figueroa, 2010) in order to create a stable and consistent version of the Collaborative Manufacturing Services Ontology. The implemented ontology's classes which are depicted in the previous figure are presented in more details in

Table 1:

Table 1: Collaborative Manufacturing Services Ontology Classes

Class name	Description
Business entity	Represents an Ecosystem Agent who has a service (e.g. manufacturing service) and provides or seeks an offer
Business entity type	Represents the legal form, the size and the position of a business entity in value chain
Service	Conceptualizes all operations and processes related to a product in an abstract level
Operation	Represents the processes of a service
Resource	Represents the total set of linked resources of a business entity
Supporting service	Represent services which are not basic services but are related to the basic one and support them
Supporting system	Represents some systems which support a business entity's services
Offer	Represents a public announcement of a business entity that provides or seeks a certain service or product
Warranty	Represents the duration and the scope of free services that will be provided to a customer in case of a possible malfunction or problem
Quantitative value	Represent the range of a certain property
Generic Term	Define common operations, materials and tools
Delivery method	Define the available delivery options for a service or product
Dates and Times	The days that a business entity has opening hours. Also represents the day of delivery or the day of availability of a service
Capability	Represents the capability of a service
Entity	Represents an entity as a result of a manufacturing process and describe its geometric flaw and entity, assembly entity and raw material
Price specification	Specifies the price of a unit, additional delivery costs and additional costs related to a payment method
Payment method	Describes the available procedures for transferring the requested amount for a purchase
Certification	Certification of an entity (service, product, material etc.) e.g. ISO

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. Such a Marketplace is defined as 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 that 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 cash flow 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 COMPOSITION Marketplace Agents

4.3.1 Matchmaker and Agents Communication for Marketplace Requests

Agents are primary actors of the COMPOSITION marketplace. They typically instantiate the supply-chain formation strategy of industry stakeholders and are therefore crucial for the success of the project inter- factory solutions. Although in the long term, many different agent types are expected to coexist in the same marketplace. Two main categories of agents can be defined a priori, depending on the kind of provided services: Marketplace agents and Stakeholder agents.

Marketplace Agents: Following FIPA specifications, an Agent Management System (AMS) is a mandatory component of every agent platform, and only one AMS should exist in every platform. It offers the White Pages service to other agents on the platform by maintaining a directory of the agent identifiers currently active on the platform.

Stakeholder agents are deployed at the stakeholder’s premises and their purpose is to fulfil the stakeholder’s interests. In the following sections the reference implementations for the two different kinds of stakeholder agents will be described. The set of APIs for the interaction with the agents will not be described here, since they have been thoroughly analysed in deliverable D6.5: Connectors for Inter-factory Interoperability and

Logistics I and will be updated at D6.6. Two types of stakeholders' agents have been identified: the Requester agent and the Supplier agent.

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 and matching. Due to the "open" and potentially evolving nature of the marketplace, suitable techniques have to be applied to ensure that the agent's implementation and the data models linked with the Ontology remain aligned.

Therefore, in order the agents to have a fully-transparent communication with the Matchmaker and keep 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

The collaboration scheme and the information flow between agents and the matchmaker is presented in details on chapter 8 of this document.

4.3.2 Matchmaker and Agents Communication for Matchmaker's Connection with Deep Learning Toolkit and Reputation Model

Besides the main process (Marketplace requests and evaluation) in the communication of the two components, the Matchmaker and the Agents communicates again in order to enhance the main process of the requests.

Matchmaker and Deep Learning Toolkit (DLT)

The Matchmaker component is able to evaluate the provided offers and select the best available from them. The price is one of the major criteria in this evaluation process. However, it is not possible for the Matchmaker to determine if the price of an offer is decent or not. As this feature was demanded by the end-user, the Matchmaker component capitalizes on the price forecasting functionality of the DLT.

Deep Learning Toolkit described in D5.4 Continuous Deep Learning Toolkit for real time adaptation II. The DLT is able to derive the latest prediction on the price per ton at which users are likely to accept to buy or sell waste materials within a fixed timeframe in the future. The forecasted price values are retrieved by an HTTPS GET request to a Marketplace Agent which is connected with the DLT.

The Matchmaker uses the retrieved price to define if the provided offers are valid or it should reject some of them as it is estimated that the given prices are beside the market price.

Matchmaker and Reputation Model

Besides the price criterion, the Matchmaker component also uses the Marketplace agents' rating in the evaluation process. The rating are available to the Matchmaker through the Agent-based Reputation Model which is described in details in D4.5 Prototype of the Security Framework II.

The Agent-based Reputation Model Engine computes the reputation values(ratings) of the agents. The Reputation Model is deployed in the Security Framework, which offers a set of REST API which can be invoked by an Agent, whenever it is needed. The reputation/rating value (1 to 5) is posted/updated to the COMPOSITION Ontology by an agent. By the time this value is available to the ontology it can be used by the Matchmaker during the matching and evaluation processes.

5 Introduction to Matchmaker Usage in COMPOSITION Use Cases

The Matchmaker component is connected with inter-factory use cases of the project, which are related to Marketplace services.

5.1 UC-KLE-4 Scrap Metal Collection and Bidding Process

The use case demonstrates the ecosystem enabling actors to exchange requests and offers using the agent-based marketplace. Their goal is to optimize scrap metal collection and bidding process. Usually, the seller wants to get the best price and reduce costs to arrange for immediate pick up of the scrap metal container.

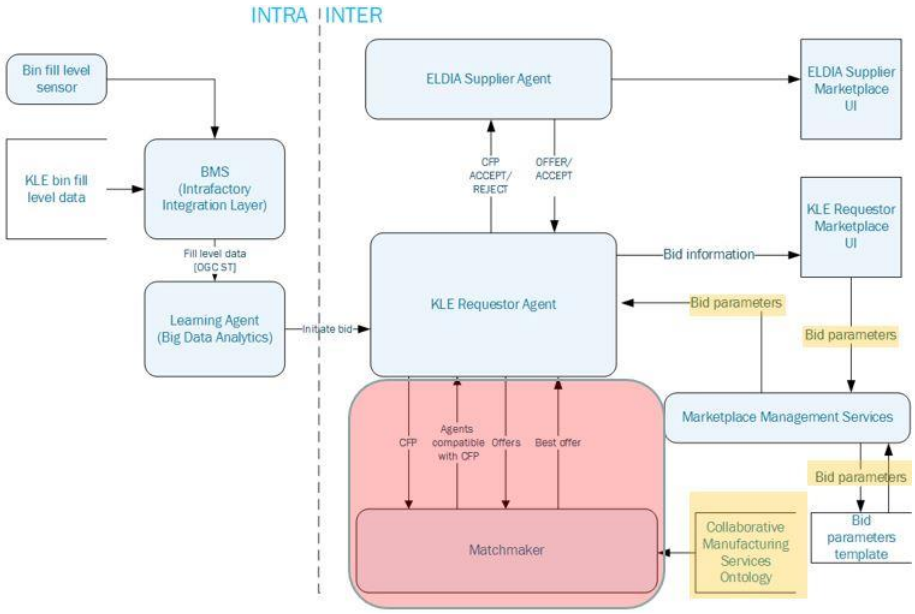

Figure 3: UC KLE-4 Data Flow

In this case the KLEEMANN agent requests waste management solution for scrap metal. The Matchmaker response to this request with the list of possible suppliers based on information on Collaborative Manufacturing Services Ontology. Then the bidding process starts. As soon as the suppliers' offers are available the KLEEMANN agent ask from Matchmaker to evaluate them based on bidding parameters such as price, delivery time and rating. Finally, the best matching offer with these parameters/criteria was returned to the requester agent from KLEEMANN.

5.2 UC-KLE-7 Ordering Raw Materials

The use case is almost similar with UC KLE-4. KLEEMANN agent initialize a bidding process for raw materials. The goal of the purchasing manager of KLEEMANN, who is represented by the corresponding agent, is to get high quality raw materials on the best price and delivered on time. The goal of raw material suppliers is to provide high quality products and to establish good customer relationship. The data flow of this use case is presented in the next figure.

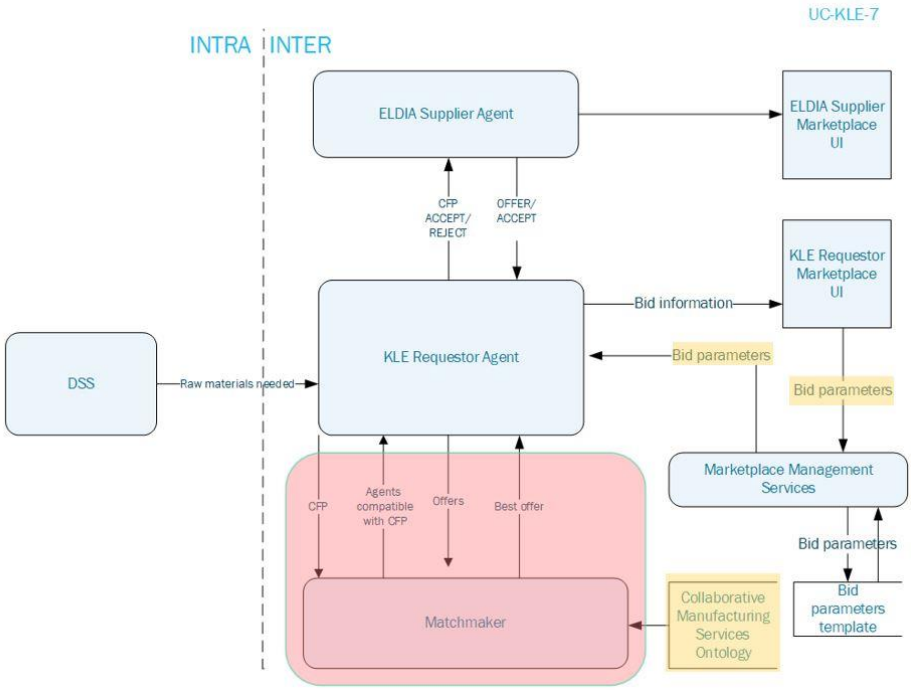

Figure 4: UC-KLE-7 Data Flow

Unlike the UC KLE-7, in this use case the evaluation of the offers is a more complex procedure. In order to evaluate a raw materials' offer the Matchmaker should take into consideration more parameters besides price, rating and delivery time. These can be certifications, shipping costs, payment terms etc. Therefore, the rule-based logic of the Matchmaker is enhanced for this use case with well-weighted algorithms.

5.3 UC-ATL-1 Searching for Solutions

In this use case, ATLANTIS or NXW which are SMEs that provide software solutions related to manufacturing domain are able to advertise their solutions, products and consultancy services to the COMPOSITION eco-system. As soon as a potential client has a problem and requests software solution via the ecosystem, the agent is able to match the requester with ATLANTIS or NXW or a company from the same domain by using Matchmaker capabilities. In contrast with the previous use cases, this scenario stops in the first level of matching as services such as software solutions or consultancy demand communication between clients and providers and it is not so easy to be handled by an automated bidding process.

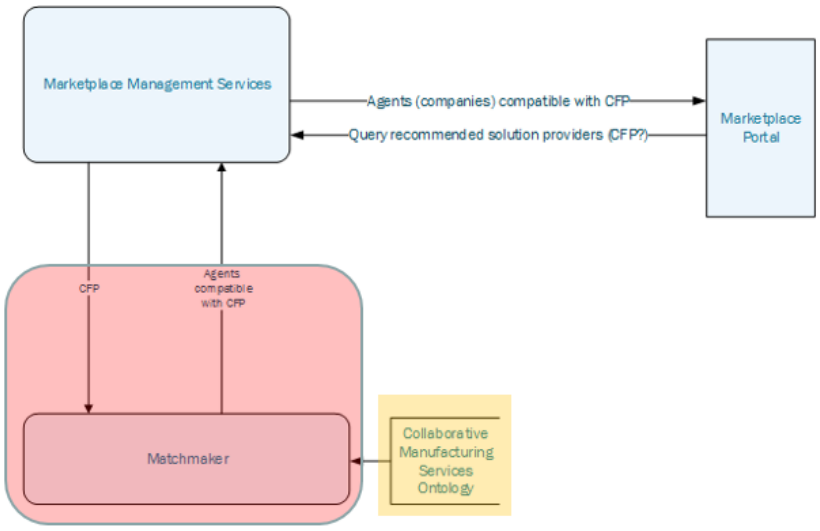

Figure 5: UC-ATL-1 Data Flow

6 Related Works

In this chapter the state-of-the-art analysis and the related works to COMPOSITION Matchmaker are presented. The chapter is divided in two sub-sections. The first is related to semantic matching techniques and the second one is related multi-criteria methods that were used to enhance the rule-based logic in the final version of the component.

6.1 Semantic Representation and Brokering, and Matchmaking Techniques

There are several existing approaches related to manufacturing semantic representation and brokering, and matchmaking techniques. However, the related research mainly presents frameworks which are not completely related to manufacturing domain in connection with the supply chain domain. Furthermore, they are not exclusively designed for one system and they are not easily extended and adoptive by other agent-based ecosystems. The following related works are presented by the perspective of semantic representation and matchmaking.

LARKS

LARKS (Language for Advertisement and Request for Knowledge Sharing) (Sycara, 1999) based matchmaking engine was used in RETSINA 1 (Reusable Task Structured-based Intelligent Network Agents) infrastructure. It was a multi-agent infrastructure that was developed by the Carnegie Mellon University in Pittsburgh, USA and contained a matchmaking engine that relies on service matching. The matchmaking was based in LARKS which express advertisements and requests using the same language. Five different filters were contained in the aforementioned matchmaking engine: key-word-based matching, similarity matching, profile comparison matching, constraint matching and rule-based signature matching. Nevertheless, the RETSINA/LARKS matchmaking framework lacks of features matching. The used language is not focused on manufacturing domain and the LARKS matchmaker needs a manufacturing domain ontology which should be compatible with LARKS in order to be used as the content. Only then it is able to perform matching. However, due to the general nature of RETSINA/LARKS matchmaking engine, it is unable to capitalize on the advantages of the representation of the manufacturing specific services, tools and resources in order to be used in modern collaborative manufacturing ecosystems.

InfoSleuth

An agent-based system which performs different level information management activities was developed by MCC Inc., Texas, USA. This was InfoSleuth (Nodine, 2000). In the set of various agents which were offered by InfoSleuth, some Broker agents existed. These agents provide syntactic and semantic matchmaking between services' providers and requesters. In order to describe requests and advertisements a specific "InfoSleuth ontology" was used by the agents. The broker agents use textual comparisons for syntactic matchmaking of advertisements and queries. In the case of semantic matchmaking, broker agents apply SQL queries and then constraint matchmaking to queries' output in order to eliminate useless results based on advertisement capabilities and formal descriptions of the requests. However, the "InfoSleuth ontology" is not able to represent manufacturing services and resources as it is focused on advertisements and requests description. Thus, the broker agent's matchmaking engine is unable to perform a matchmaking process which covers the requirements of manufacturing collaborative ecosystems.

IMPACT

IMPACT (Interactive Maryland Platform for Agents Collaborating Together) (IMPACT, 2018) is an international research project led by the University of Maryland. It is related to software implementation that facilitates the creation, deployment, interaction, and collaborative aspects of software agents in a heterogeneous, distributed environment. IMPACT provides algorithms supporting a variety of applications including supply chain, logistics, and e-commerce. It supports multi-agent interactions and agent interoperability in an application independent manner. It provides a yellow pages server that performs basic matchmaking among agents based on weighted hierarchies. It maintains a verb and a noun hierarchy of synonyms and retrieval algorithms to compute similarities between given service specifications. So the IMPACT matchmaker uses only similarity and distance algorithms in order to perform matching. Moreover, the IMPACT matchmaker is not designed to support manufacturing domain concepts.

Digital Manufacturing Market

Digital Manufacturing Market (Ameri (AMeri), 2012) is a multi-agent web-based framework that contains a manufacturing services ontology and a matchmaking mechanism which match a consumer's requirements

with suppliers' manufacturing capabilities. The ontology used in this multi-agent framework is MSDL (Ameri, 2006), which stands for Manufacturing Service Description Language. MSDL is a manufacturing domain ontology which enables the representation of services and resources by describing manufacturing capabilities in four levels of abstraction: supply and demand level, shop-floor level, process level and machine level. Both advertisements and requests are expressed by agents using the MSDL as a common language. A middle agent, in order to find possible suppliers for a requested process, performs both features-based and taxonomy-based matchmaking. A list with possible suppliers is returned to the requester agent. The Digital Manufacturing Market approach is the closest one with the presented matchmaker as it uses a common manufacturing ontology and performs semantic matching based on the services descriptions and terms related to this ontology. Besides some similarities in matchmaking logic for service and agent level matchmaking which will be presented in this report, the Digital Manufacturing Market solution does not use e-commerce concepts to extend the matchmaking process in an offer level in which the evaluation of the matching offers can be executed based in different qualitative and quantitative criteria.

FITMAN-SeMa

FITMAN-SeMa (Metadata and Ontologies Semantic Matching SE) (FITMAN-SeMa, 2018) is a component of FIWARE (FIWARE, 2018) for Industry 3 aims to solve interoperability problems in the collaboration of business processes. Furthermore, FITMAN-SeMa provides storing and retrieving functionalities for ontologies and triplets. By using various algorithms FITMAN-SeMa performs effective semantic matching. The FITMAN-SeMa is installable software which matches concepts between two different ontologies. This different approach may enable collaboration and possible matching of two different sources. Nevertheless, it is not a manufacturing agent-based eco-system dedicated solution. In order to achieve a higher level of interoperability FITMAN-SeMa introduces a solution which is not based in a central ontology. But this last feature makes the SeMa unable to extract conclusions from manufacturing domain in order to perform an efficient matchmaking of agents and services as it is not designed for this domain.

In conclusion of the related works analysis, it is perceived that most of the existing solutions are not exclusively designed for the manufacturing domain and lacks the necessary concepts that will enable efficient reasoning in term of manufacturing. Besides this, other approaches are completely related to this domain and lacks the ability to represent e-commerce means which are important for the reasoning and matchmaking over on-line marketplaces.

6.2 Multi-Criteria Decision Methods

Multiple decision-making problems enclose the determination of the optimal alternative from several potential candidates in a decision, depending on several criteria or attribute that may be concrete or vague. The most widely used method is the Weighted Sum Model (WSM). The WSM is described by the following equation (Fishburn, 1967):

$$A_{WSM} = \max_{i} \sum_{j=1}^{N} q_{ij} w_j \,, \quad for\ i = 1,2,3, \dots, M. \tag{1}$$

where A_{WSM} is the WSM score of the optimal of M alternatives, N is the number of decision criteria, q_{ij} is the actual value of the $i-th$ alternative in terms of the $j-th$ criterion, and w_j is the weight of importance of the $j-th$ criterion. This method requires a dataset expressed in the same unit for each alternative, thus it is an utmost convenient method for single-dimensional problems.

A modification of the WSM is the weighted product model (WPM). In WPM, each alternative is compared with the others by multiplying a number of ratios for each criterion. WPM is described by the next equation (Miller and Starr, 1969):

$$R\left(\frac{A_K}{A_L}\right) = \prod_{j=1}^{N} \left(a_{Kj}/a_{Lj}\right)^{w_j} \tag{2}$$

where N is the number of criteria, a_{ij} is the actual value of the $i-th$ alternative in terms of the $j-th$ criterion, and $w_j$is the weight of importance of the $j-th$ criterion. If $R\left(\frac{A_K}{A_L}\right)$ is greater than one, alternative A_K is more preferable than A_L. This method eliminates the dimensionality limit of the previous one.

The analytic hierarchy process (AHP), is another popular method (Saaty, 1994). AHP is similar to WSM, thus it can be applied in both single and multi-dimensional problems, since it uses relative values for each alternative and not the actual ones which add up to one.

A revised AHP method was introduced later on (Belton and Gear, 1981) with some effective modifications. The extension of the method is that instead of calculating relative values of the alternatives sum up to one, each relative value is divided by the maximum value of the relative values.

Last but not least, ELECTRE (Benayoun, et al., 1966) and TOPSIS (Hwang and Yoon, 1981) are two notable methods, with the second being an alternative version of the first one. The basic concept of the ELECTRE method is to use pairwise comparisons among alternatives regarding each criterion and manage "outranking relations". The outranking relationship of $Ai\,.\,Aj$ indicates that even when the $i-th$ alternative does not prevail the $j-th$ alternative quantitatively, then Ai can still be assumed to be a better choice than Aj (Roy, 1973).

Regarding TOPSIS method, the basic concept is that the optimal selection should have the shortest distance from the ideal solution and the farthest distance from the negative-ideal solution in a geometrical sense. TOPSIS assumes that each attribute has a tendency of monotonically increasing or decreasing utility. The evaluation of the relative closeness of alternatives to the ideal solution is calculated by the Euclidean distance approach and the priority order of the alternatives is resulted by the comparison of these relative distances.

7 Design of Brokering and Matchmaking components

COMPOSITION Matchmaker is designed to be the core component of the COMPOSITION Broker. It supports semantic matching in terms of manufacturing capabilities, in order to find the best possible supplier to fulfil a request for a service or products involved in the supply chain. Different decision criteria for supplier selection, according to several qualitative and quantitative factors, are considered by the Matchmaker. Furthermore, the Matchmaker acts as a broker for the Marketplace's bidding processes and enables the automation of these processes as well. The Matchmaker evaluates the available offers from the providers in order to suggest the best one to the supplier.

Since the Matchmaker component is built upon the Apache Jena API, the basic components of this API are presented in this chapter as well. Before the design and implementation details of the Matchmaker, the corresponding requirements are also presented.

7.1 Apache Jena API

Apache Jena (Apache Jena, 2018) is a free and open source Java framework for building Semantic Web and Linked Data applications. 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. Figure 6 below represents Jena framework's architecture.

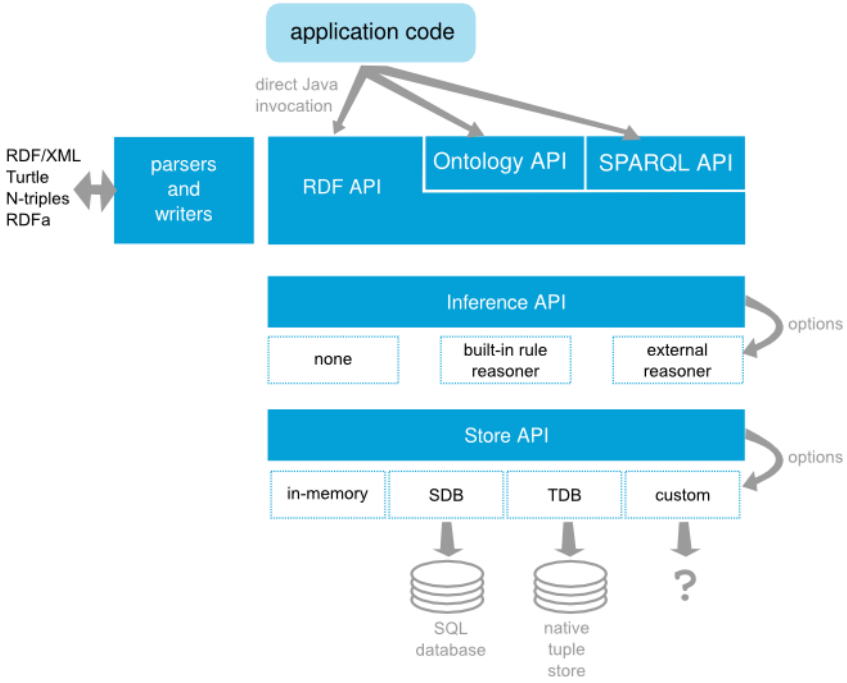

Figure 6: Apache Jena's framework architecture (Apache Jena, 2018)

RDF API

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 one of the entities. 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).

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 the following 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.

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 modelling 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: Java class names are not specific to the underlying language.

In order the 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", 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 6.

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. On the other side, 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.

7.2 Matchmaker Requirements

The design and the implementation of the COMPOSITION Matchmaker were driven by the project's requirements. The main requirements related to the matchmaking component are listed below:

Table 2: Main Matchmaker Requirements

Requirement Number	Title	Short Description
COM-61	Suppliers' product/services shall be matched with a potential customers' needs/problems	This requirement relates to unite both suppliers and potential new customers in an automatic ecosystem, precisely matching the customers' needs with the companies' products and services. The system suggests for example a top five of potential suppliers, based on certain criteria, set by the customer

COM-62	COMPOSITION Marketplace supports participants services' description and potential matching of participants based on these services	Potential customers are subscribed as well as suppliers of different products/services are registered within the ecosystem. This could be realized using the agents (which may have any system, including humans, running them; the only requirement is that they talk CXL) and the Collaborative Manufacturing Services Ontology
COM-63	The system provides an automatic ranking of the suppliers to the buyers, based on the buyers' criteria	The system provides ranking recommendations to companies about suppliers of products/services based on objective criteria
COM-64	The system provides an automatic ranking of the suppliers to the buyers, based on customers' satisfaction and feedback	The system provides ranking recommendations to companies about suppliers of products/services based on customers' satisfaction. This requirement will raise chances for more unknown providers.
COM-86	The Matchmaker shall apply both syntactic and semantic matching	The Matchmaker shall apply both syntactic and semantic matching (both taxonomy-based and feature-based) in terms of manufacturing capabilities, in order to find the best possible supplier to fulfil a request for a service, raw materials or products involved in the supply chain
COM-87	Different similarity algorithms and metrics shall be supported by the Matchmaker	For measuring the similarity among offers and requests, well-established weighted similarity algorithms and metrics will be supported by the Matchmaker and will be further extended if needed, in order to address the objective of COMPOSITION at the best possible way
COM-88	Different decision criteria for supplier selection are supported by the Matchmaker	Different decision criteria for supplier selection according to several qualitative and quantitative factors shall be considered (e.g. size of buyer's organization, cost, time, distance, due date, quality, price, technical capability, financial position, past performance, attitude, flexibility, etc.) in matchmaking
COM-89	Matchmaker shall return a result within a reasonable time frame	The Matchmaker should respond within a reasonable time frame (e.g. 5 seconds)
COM-90	Ecosystem components should be deployed as Docker images	Docker gives ease of deployment and simpler integration of heterogeneous components. The partner developing the component can perform exact configuration of target platform and setup is easy for other partners. Many third-party components are also available as Docker images
COM-148	Matchmaker and Agents components should be able to access and manipulate Marketplace Ontology	The matchmaker and the agent components should be able to access the Ontology Store. Based on type of agents, the should be able to infer knowledge or store and retrieve data from Collaborative Manufacturing Services Ontology

7.3 Matchmaker Implementation Details

The COMPOSITION Semantic Matchmaker is built upon Apache Jena framework. The Semantic Matchmaker aims to infer new knowledge from the Collaborative Manufacturing Services Ontology based on semantic rules in order to perform matchmaking. In the overall COMPOSITION architecture, the Matchmaker block contains the complete semantic framework of the project. This framework contains:

- Collaborative Manufacturing Services Ontology which initialize the Ontology Store (RDF triple store)
- Ontology Query Engine and the corresponding Ontology API which enable the manipulation of the Ontology Store by the Marketplace agents
- Matchmaker, which applies, sets of semantic rules at the Ontology Store. Moreover, the last version of Matchmaker module uses weighted assessment algorithm for offers' evaluation.

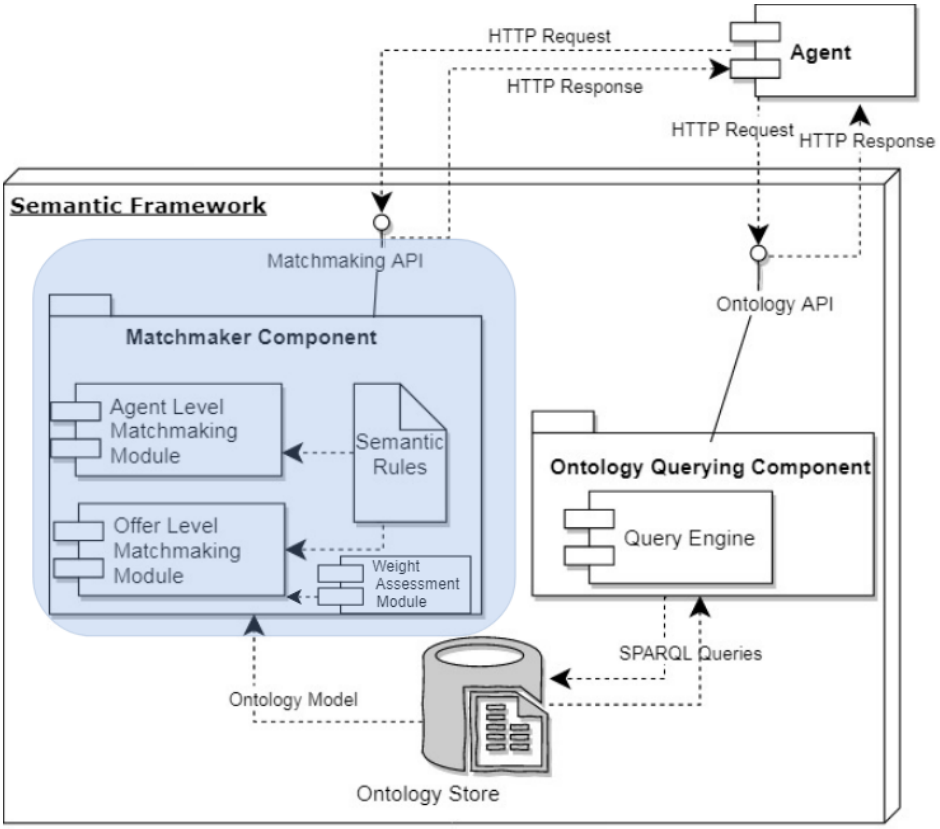

Figure 7: COMPOSITION Semantic Framework Architecture

The third of the aforementioned components will be analyzed in this report as this one is about Brokering and Matchmaking. The other two components were presented at their corresponding report, D6.8 Collaborative manufacturing services ontology and language II (M30). The COMPOSITION Matchmaker is mainly a rule-based matchmaking engine enhanced with multi-criteria (weighted) algorithms for offers' evaluation. However, as it is primary based on rule-logic we are going to refer to it as Rule-based Matchmaker.

7.3.1 Introduction to Semantic Rules

The Rule-based Matchmaking component functionality depends on sets of semantic rules contained in the module. Therefore, in this sub-section the basics of semantic rules are introduced.

The semantic rules are commonly specified by means of an ontology language. These rules are used to infer new knowledge based on the existing one in the knowledge base/ontology and can be added as RDF triples. The rules are fired by reasoners, which can be used and activated in applications. A reasoner is software capable of inferring logical consequences from a set of asserted facts or axioms. In the case of the Rule-based Matchmaker a rule-based reasoner offered by Jena API will be used. A rule for the rule-based reasoner is defined by a Java Rule object with a list of body terms (premises), a list of head terms (conclusions) and an

optional name and optional direction. A term is a triple pattern, or an extended triple pattern or a call to a built-in primitive. A rule set is simply a List of Rules. The following image presents the simplified text rule syntax:

```
Rule      :=   bare-rule .
           or   [ bare-rule ]
         or   [ ruleName : bare-rule ]

bare-rule :=   term, ... term -> hterm, ... hterm    // forward rule
           or   bhterm <- term, ... term    // backward rule

hterm     :=   term
           or   [ bare-rule ]

term      :=   (node, node, node)           // triple pattern
           or   (node, node, functor)        // extended triple pattern
           or   builtin(node, ... node)      // invoke procedural primitive

bhterm      :=   (node, node, node)           // triple pattern

functor   :=   functorName(node, ... node)  // structured literal

node      :=   uri-ref                   // e.g. http://foo.com/eg
           or   prefix:localname          // e.g. rdf:type
           or   <uri-ref>          // e.g. <myscheme:myuri>
           or   ?varname                    // variable
           or   'a literal'                 // a plain string literal
           or   'lex'^^typeURI              // a typed literal, xsd:* type names supported
           or   number                      // e.g. 42 or 25.5
```

Figure 8: Jena Rules Syntax (Apache Jena, 2018)

A rule file has the main basic components:

- *@prefix* defines a prefix which can be used in the rules. The prefix is local to the rule file Example: @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
- *//* are comment lines
- *Triple patterns* – like a triple, but with some named variables instead of fixed parts
- *Rule "Body"* – Set of triple patterns, all of which must match.
- *Rule "Head"* – Set of triple patterns that will be asserted, when the body matches

Table 3: Jena Rule Example

Textual Format	Jena Rule Format
Business Entity X requests an offer And Business Entity X matches with Business Entity Y Which offers an Offer Y Then request X matches with Offer Y	@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. [exampleRule: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?x comp:matchesWith ?y) (?y v1:offers ?Offery) ->(?Offerx comp:matchingOffer ?Offery) //inferred knowledge is that offer x matches with offery]

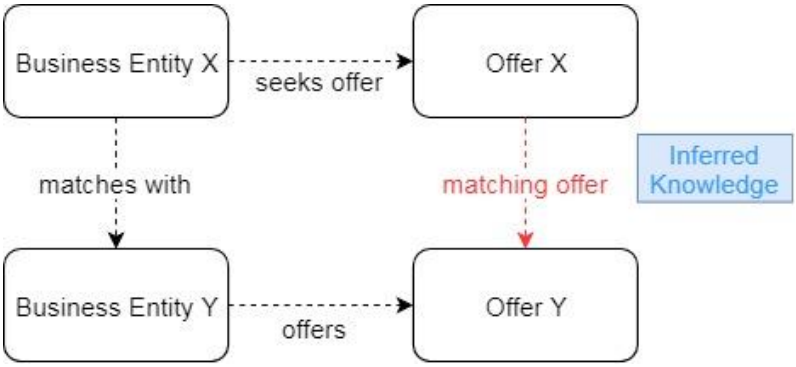

Figure 9: Jena Rule Example Representation

The above simplified Jena rule example explains how new knowledge can be inferred. A request (Offer X) can be matched to an offer (Offer Y) by this simple rule. The two instances, Offer X and Offer Y are connected with the object property 'matching offer'. This is the new knowledge that originally does not exist in the Ontology.

Furthermore, the Jena API offers a wide set of built-in primitives that can be included and used in rules files. The procedural primitives which can be called by the rules are each implemented by a Java object stored in a registry. Each primitive can be used in the rule body, the rule head or both. Some interest built-in primitives which many of them are used by the COMPOSITION Matchmaker are listed below. Moreover, additional/custom primitives can be created.

Table 4: Examples of Built-in Primitives

Built-in Primitive	Short Description
equal(?x,?y) notEqual(?x,?y)	Test if x=y (or x!= y). The equality test is semantic equality
lessThan(?x, ?y), greaterThan(?x, ?y) le(?x, ?y), ge(?x, ?y)	Test if x is <, >, <= or >= y
sum(?a, ?b, ?c) addOne(?a, ?c) min(?a, ?b, ?c) max(?a, ?b, ?c)	Sets c to be (a+b), (a+1), min(a,b), max(a,b)
remove(n, ...) drop(n, ...)	Remove the statement (triple) which caused the n^{th} body term of this rule to match. Drop will silently remove the triple(s) from the graph but not fire any rules as a consequence.
print(?x, ...)	Print a representation of each argument.
noValue(?x, ?p)	True if there is no known triple (x, p,)

7.3.2 Matchmaking Module

The Matchmaking Module is developed in Java and it is built upon the Apache Jena API. The Matchmaker is offered to other components through RESTful web services. Its core functionality is to receive Marketplace Agents' requests via Matchmaker API and to apply sets of semantic rules to the Ontology Store based on these requests. New knowledge will be inferred by the rules' appliance, and then the Matchmaking Module responses to the Agents by using the Matchmaker API. The next steps are followed by the Matchmaking Module:

1. The module receives requests by agent (requests are based on REST and HTTP)
2. The module accesses the Collaborative Manufacturing Services from the Ontology Store. An Ontology Model can be created in the memory or it can be accessed directly from the file system.

3. The module transforms the request from agents' COMPOSITION eXchange Language (CXL is in JSON format) compatible format to terms of the ontology and creates instances (if needed)
4. The module reads the Jena rules as a List of Jena rules files
5. A reasoner is selected. A reasoner can be created by calling an instance of a reasoner class or by retrieving from reasoner registry which contains instances indexed by URI assigned to the reasoner. The GenericRuleReasoner class is selected for the COMPOSITION Matchmaker purposes as it is a reasoner interface that is able to invoke any of the useful rule engine combinations.
6. The rules list is set after the reasoner instance is created. This action indicates to the reasoner the set of rules that should execute
7. An inference model will be created after applying the reasoner to data.
8. The module accesses the information stored in inference model. The content of the inference model is the generated output after performing inference
9. The module transforms the inferred information to agents' CXL
10. The output is returned as a response via Matchmaker API (REST and HTTP) to the Agent in a format compatible to CXL

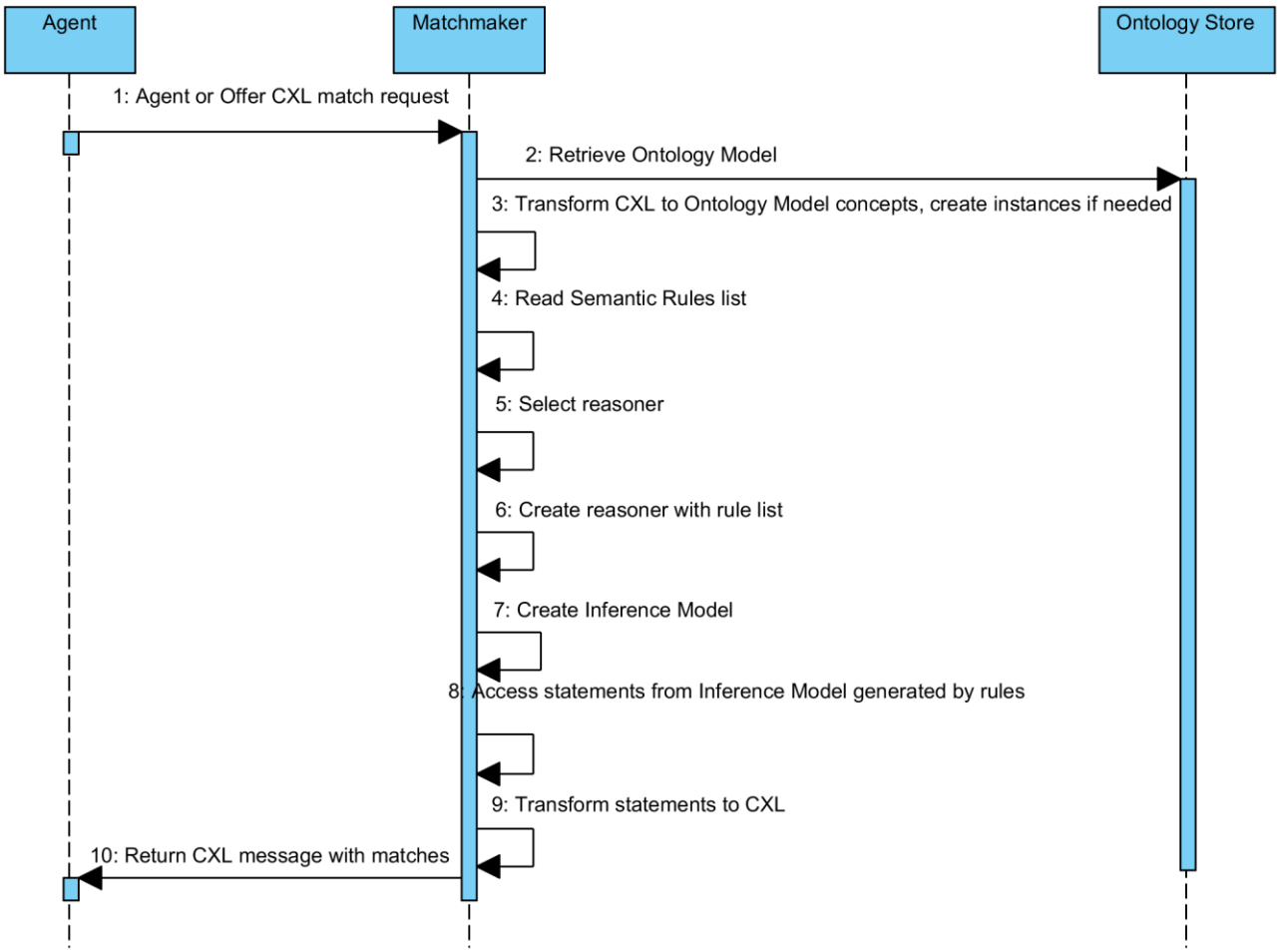

Figure 10: Agent to Matchmaker request sequence diagram

The Matchmaking Module contains two sub-modules. The Agent Level and the Offer Level matchmaking modules which are described in detail below.

7.3.2.1 Agent Level Matchmaking

The Agent Level module aims to match Marketplace agents which are possible customers and suppliers. In this level of matching the Matchmaker applies rules which are based on ontology's classes: Business Entity, Generic Term, Capability, Service, Operation and Resource. The applied rules targets to infer knowledge that enables the beginning of negotiation among the Marketplace stakeholders. The matchmaker indicates to a requester agent a list of possible supplier agents based on some requested criteria.

At this level of matching the semantic rules are focused on service level. For an agent who requests a service in the COMPOSITION Ecosystem, the Matchmaker will provide the agents which offers this service. In order to find possible providers of this service, the Matchmaker applies the following semantic rule based on terms of the ontology:

Table 5: Rule for Matching Business Entities

Textual Format	Jena Rule Format

Business Entity X
requests an offer which
includes a service which
supports a specific OperationX
and is related to MaterialX
Business Entity Y
offers a service which
supports an operation which
based on Generic Terms Catalog
is
mapped with operation Y and
the related material is mapped to a
common material too
Then
Entity X matches with Entity Y
	`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.`
	`@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.`
	`@prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>.`
	`@prefix v1: <http://purl.org/goodrelations/v1#>.`
	`@prefix p1: <http://www.owl-ontologies.com/mason.owl#>.`
	`@prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>.`
	
	`[agentLevelMatching:`
	
	`(?x rdf:type v1:BusinessEntity)`
	`(?x v1:seeksOffer ?Offerx)`
	`(?Offerx v1:includes ?Servicex)`
	`(?Servicex comp:seeksOperation ?Operationx)`
	`(?Operationx p1:allowedProcessFor ?Materialx)`
	`(?y rdf:type v1:BusinessEntity)`
	`(?y MSDL:hasService ?Servicey)`
	`(?Servicey comp:hasOperation ?Operationy)`
	`(?Operationy comp:mappedToCommonTerm ?Operationx)`
	`(?Operationy p1:allowedProcessFor ?Materialy)`
	`(?Materialy comp:mappedToCommonMaterial ?Materialx)`
	`->`
	`(?x comp:matchesWith ?y)`
	
	`]`

Using the previous rule, the semantic matchmaker is able to match services, more precise operations based on some common term instances that exist in the Collaborative Manufacturing Services Ontology. Every business entity uses its own terms to describe one of its offered services, products and materials. However, every one of these vendor specific terms will be mapped with a common generic term. 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.

Figure 11 is an illustration of the Agent Level matching process, searching a company which provides the raw material "Tube". The first column is the Agent Request flow and the above rule is applied on Business Entities located in the ontology, searching for a company that provides raw materials and specifically, "Tube". The second column is the check for "Company_B", the search is dropped when the service provided does not match with "Provide_raw_materials". The third column matches "Company_D" with the requested service, but the name of the provided raw material does not match with "Tube". The last column is the successful match of the request with "Company_E", which provides both raw material operation and "Tube" material.

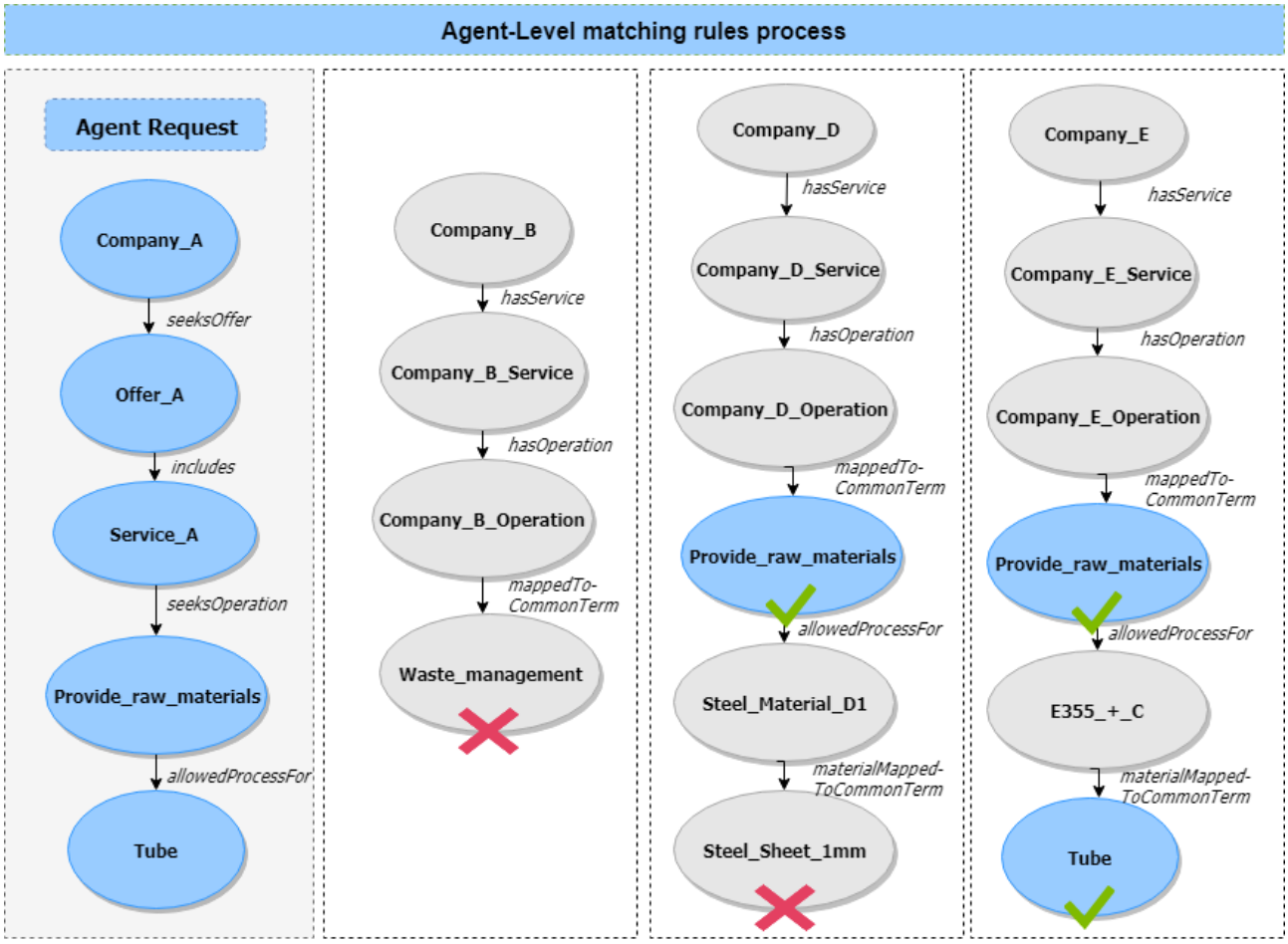

Figure 11: An example of the matchmaking of an agent request for provision of raw material Tube

Moreover, further rules can be added in order to give a matchmaking result based also on some criteria by the requesters as a kind of filtering. For example, the requester can ask for a supplier who offers a specific service and has a Marketplace rating greater than a requested value. The following rule describes the aforementioned user's requirement.

Table 6: Rule for Filtering Based on Rating Requirement

Textual Format	Jena Rule Format

Business Entity Y
requests an offer from a Business Entity and
Matches With a Business Entity X
And Business Entity Y's request has a requested minimum rating for possible supplier
IF Business Entity X' rating is less than the demanded minimum rating
Then
Drop Business Entity X from the matching list
	`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.`
	`@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.`
	`@prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>.`
	`@prefix v1: <http://purl.org/goodrelations/v1#>.`
	`@prefix p1: <http://www.owl-ontologies.com/mason.owl#>.`
	`@prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>.`
	
	`[filterRating:`
	
	`(?y rdf:type v1:BusinessEntity)`
	`(?y comp:matchesWith ?x)`
	`(?y v1:seeksOffer ?Offery)`
	`(?Offery comp:hasMinRating ?minRating)`
	`(?x rdf:type v1:BusinessEntity)`
	`(?x comp:hasRating ?ratingx)`
	`lessThan(?ratingx, ?minRating)`
	`->`
	`drop(1)`
	`]`

Additional criteria by the requester agent can improve the Matchmaker's result. After the initial matching based on the provided services the Matchmaker is able to apply more rules in order to exclude some suppliers from its final output. The rules that will be applied are related to quantitative criteria of the services. For example, a waste management service is capable to handle a limited number of waste tonnages or a manufacturing service is able to produce a specific number of units/products. The next generic rule is applied for the exclusion of agents (business entities) from the matching ones based on services' capabilities:

Table 7: Rule for Capability Fulfilment

Textual Format	Jena Rule Format

Business Entity X
requests an offer which
has a quantity requested specification
with value quantity X
And Business Entity X
matches with Business Entity Y
Which has s a service with
Capability of Value quantity Y
If quantity Y is less than quantity X
Then drop Business Entity Y
from them which matches with Entity X
	`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.`
	`@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.`
	`@prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>.`
	`@prefix v1: <http://purl.org/goodrelations/v1#>.`
	`@prefix p1: <http://www.owl-ontologies.com/mason.owl#>.`
	`@prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>.`
	
	`[capabilityFulfillment:`
	
	`(?x rdf:type v1:BusinessEntity)`
	`(?x v1:seeksOffer ?Offerx)`
	`(?Offerx v1:hasEligibleQuantity ?QuantitySpecx)`
	`(?QuantitySpecx v1:hasValue ?Quantityx)`
	`(?x comp:matchesWith ?y)`
	`(?y MSDL:hasService ?Servicey)`
	`(?Servicey MSDL:hasCapability ?Capabilityy)`
	`(?Capabilityy v1:hasEligibleQuantity ?QuantitySpecy)`
	`(?QuantitySpecy v1:hasValue ?Quantityy)`
	`lessThan(?Quantityy, ?Quantityx)`
	
	`->`
	`drop(4)`
	
	`]`

Agent Level Matchmaking – Backwards Process

Besides the matchmaking as it has been described in the previous section, the Matchmaker component supports a backward functionality for some scenarios. There is a *find possible customers'* functionality in which the requester agent does not request explicitly a service but the agent can ask for possible customers based on its own services and the knowledge stored in the Ontology and it is related to manufacturing services and tools.

For example, in the following hypothetical scenario:

- COMPOSITION Marketplace contains Companies A, B, C which are manufacturers and Companies E and F which are waste management providers.
- Company D is a new waste management company at the Marketplace
- Company D collects and manage a wide catalogue of materials
- Company D wants to find possible customers at the Marketplace in order to advertise their services

Problem: It is not so useful for Company D to advertise its services in other waste management companies or to manufacturers that do not work with materials that Company D is able to handle

Solution: The Matchmaker capitalizes on information related to machine processes and materials in order to provide an effective matching for participants who search for new customers in the Marketplace. The semantic rules explore the manufacturing services which are associated with machines and tools, and they are usable on specific materials in order to perform an efficient matchmaking

As depicted in next figure, the Matchmaker is able to match the Company D only with the Companies A and B which are possible new customers for the Company D. By applying the rule which is described in Table 8 the Matchmaker returns to the requester an optimal list of possible future customers that does not contain other

companies of the same domain (actually, they are competitors) or manufacturers that do not produce waste able to be handled by the requester.

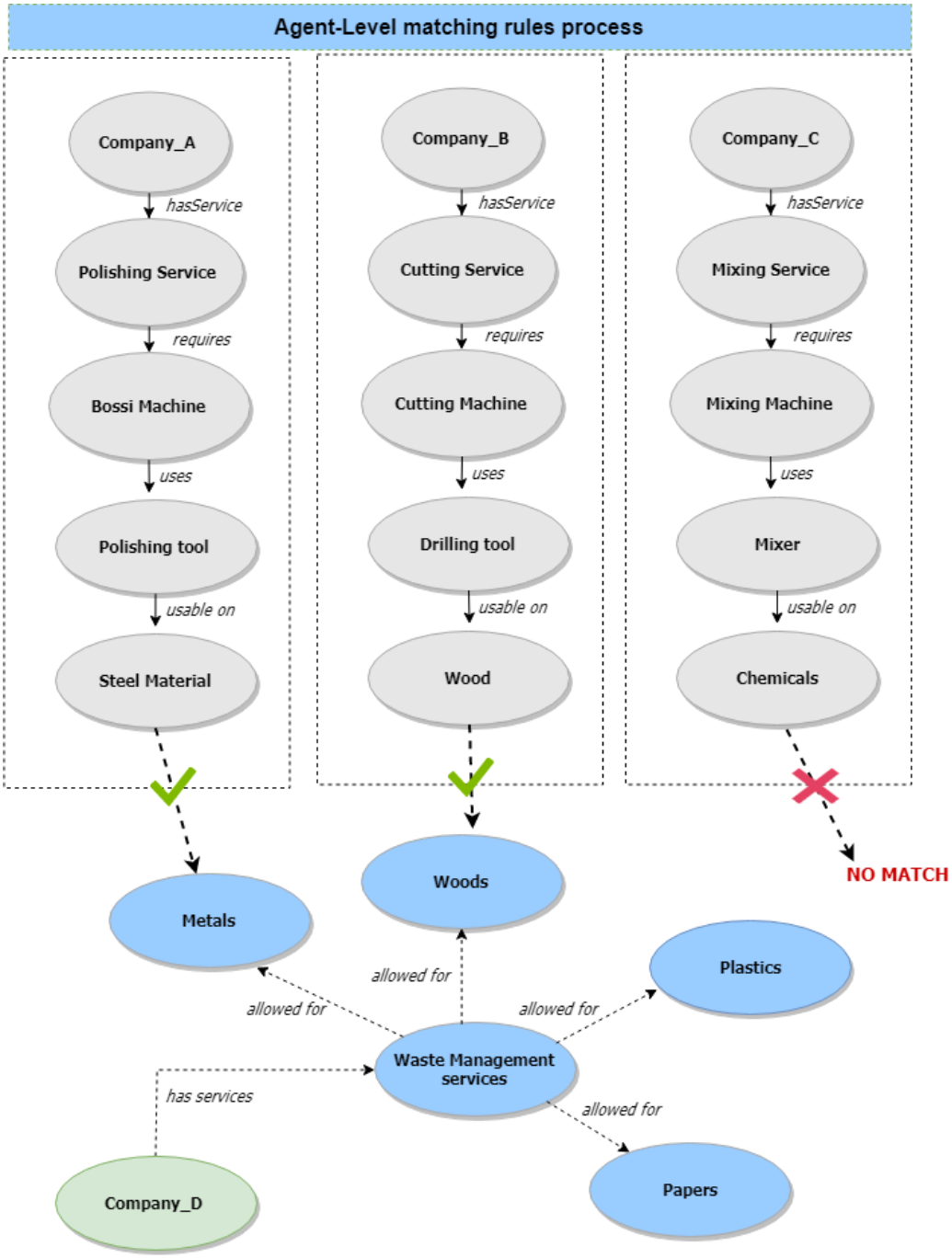

Figure 12: Find Possible Customers Based on Materials Capability

Table 8: Rule for Finding Possible Customers

Textual Format	Jena Rule Format

	`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.`
	`@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.`
	`@prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>.`
	`@prefix v1: <http://purl.org/goodrelations/v1#>.`
	`@prefix p1: <http://www.owl-ontologies.com/mason.owl#>.`
	`@prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>.`
	
	`[matchBusinessEntities:`

Business Entity Y has a service supports an operation that is related to a material Y Business Entity X has s a service with an operation that requires a machine which uses a tool This tool is usable on a material X which based on Generic terms catalogue is mapped to material Y Then Entity Y matches with Entity X X	(?y rdf:type v1:BusinessEntity) (?y MSDL:hasService ?Servicey) (?Servicey comp:hasOperation ?Operationy) (?Operationy p1:allowedProcessFor ?materialy) (?x rdf:type v1:BusinessEntity) (?x MSDL:hasService ?Servicex) (?Servicex comp:hasOperation ?Operationx) (?Operationx p1:requiresMachine ?machinex) (?machinex p1:usesTool ?toolx) (?toolx p1:toolUsableOn ?materialx) (?materialy comp:mappedToCommonMaterial ?materialx) ->(?y comp:matchesWith ?x)]

7.3.2.2 Offer Level Matchmaking

The Offer Level Matchmaking module is related to offers' evaluation. A Marketplace agent can provide to the Matchmaker a set of offers that this agent had received from supplier agents in order to ask for offers' evaluation. Based on this feature, the Matchmaker can act as Broker who aims to match the needs of the requester agents with the best available offer based on different kind of criteria.

The offer evaluation process depends on the kind of service requested by the agent. The two possible requested services are:

1. *Waste Management service*
2. *Provide Raw Material service*

The offers' evaluation of Waste Management service requests is being performed by just applying rules of different criteria according to the preferences of user. As in this use case the end-user wants to sell its scraps, there are few criteria for the decision make that can be handled by the rule-based logic. On the other hand, the offers of Provide Raw Material service are more complex and the system has to evaluate more complex criteria that cannot be handled by semantic rules. In this case the provided offers are evaluated by a combination of semantic rules and a multi-criteria decision-making method. The semantic rules are used to solve some true or false criteria and the multi-criteria decision-making method covers the main part of the evaluation process. Both techniques are explicitly described below.

Offer Level Matchmaking - Waste management service:

In this service the Matchmaker applies sets of semantic rules based on end-users ranked criteria.

As a first level of inspection, the matchmaking module uses DLT price prediction to reject unfair offers. The DLT is able to derive the latest prediction on the price per ton at which users are likely to accept to buy or sell waste materials within a fixed timeframe in the future. In order to determine if an offer is decent there is a mechanism comparing the offered prices with the price forecasting extracted by the DLT. If an offer's price is close enough to the predicted one, considering the DLT calculated accuracy, the offer continues to the next stage of evaluation. Otherwise the offer is rejected as it is estimated that the given price is beside the market price. The forecasted price values are retrieved by an HTTPS GET request to a Marketplace Agent which is connected with the DLT.

In the next level of matchmaking the sets of rules were designed for quantitative values' comparisons and evaluation. The algorithm which is followed is a kind of an elimination process in which the instances that do not fulfil a request's requirement are excluded from the matching set. The rules are constructed in a generic way, in order to provide different evaluation results if they are applied to the same offers but in a different sequence based on requesters' ranked preferences. These ranked preferences are taken into account by the Matchmaker's decisions. For example, for a set of identical offers, a requester, who wants quick delivery over the price, will get a different result by the Matchmaker than a requester who has the price as the top priority. After the matchmaking process, the best matching offer and the corresponding supplier agent are returned to the requester agent.

A pseudocode which explains the steps which are followed and executed in the Offer Level Matchmaking module is presented in Figure 13:

1. Read the provided offers

2. Read the requester's ranked N preferences

3. For every offer
4. create the corresponding ontology instance

5. Set all the available offers as best available

6. Create an ordered list of N rule sets based on the ranked preferences

7. For rule sets 1 to N
8. Apply Rule set #1 to the Ontology Model and exclude the matching offers which did not fulfil this rule from the best available
. . .
Apply Rule set #N to Ontology Model and exclude the matching offers which did not fulfil this rule from the best available

9. *Return the best available offer*

Figure 13: Pseudocode of Offer Level Matchmaking Module

In order to create a generic matchmaking engine which will be easily used in collaborative manufacturing ecosystems the rules were developed to cover the most important factors in the negotiations and transactions in such ecosystems. Based on research and discussions with the project's pilot partners the most important factors in their transactions are the following:

1. The *price* as in almost any transaction the target of the traders is the maximum profit.
2. The *quick delivery* of a product or service. In many cases this factor is of great importance. For example, in cases in which the production line is running continuously as there are a lot of orders the quick delivery of raw materials is more important than the price.
3. The *trust*. Before a transaction the requester of a service or product wants to be sure that the supplier is trusted, with good reviews by previous users etc.

Based on these factors the following sets of rules are created. Actually, they are pairs of semantic rules. The logic behind these pairs is that the first rule finds the best available value of a factor, and the second rule excludes the offers that contain values of this factor that do not match to the best one.

Table 9: Find Best Available Price

Textual Format	Jena Rule Format
Business entity requests an Offer X which has a price value X And Offer X matches to Offer Y which has price value Y if this value is less than value X Then the requested Offer X has price value equal to price value Y	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [findMinPriceOffer: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?Offerx v1:hasPriceSpecification ?PriceSpecx) (?PriceSpecx v1:hasCurrencyValue ?Valuex) (?Offerx comp:bestMatchingOffer ?Offery) (?Offery v1:hasPriceSpecification ?PriceSpecy) (?PriceSpecy v1:hasCurrencyValue ?Valuey) lessThan(?Valuey, ?Valuex) -> drop(3) (?PriceSpecx v1:hasCurrencyValue ?Valuey)]

Table 10: Rule to Match Request to Best Price

Textual Format	Jena Rule Format
Business Entity X requests Offer X Offer X has best available price Value X Offer X matches best with Offer Y Offer Y has price Value Y If Value Y is not equal to best price Value X Then remove Offer Y from the best matching Offers	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [matchRequestToBestOfferByPrice: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?Offerx v1:hasPriceSpecification ?PriceSpecx) (?PriceSpecx v1:hasCurrencyValue ?Valuex) (?Offerx comp:bestMatchingOffer ?Offery) (?Offery v1:hasPriceSpecification ?PriceSpecy) (?PriceSpecy v1:hasCurrencyValue ?Valuey) notEqual(?Valuey, ?Valuex) ->drop(4)]

The same pairs of rules have been implemented for the other two factors: Delivery time and the Rating of the agents in the Marketplace:

Table 11: Rule to Find Best Available Delivery Time

Textual Format	Jena Rule Format
Business entity requests an Offer X which has a best delivery time value X And Offer X matches to Offer Y which has delivery time value Y if this value is less than value X Then the requested Offer X has best delivery time value equal to price value Y	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [bestDeliveryLeadTime: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?Offerx v1:hasEligibleQuantity ?deliveryTimex) (?deliveryTimex v1:hasMinValueInteger ?deliveryMax) (?Offerx comp:bestMatchingOffer ?Offery) (?Offery v1:hasEligibleQuantity ?deliveryTimey) (?deliveryTimey v1:hasMinValueInteger ?deliveryMiny) lessThan(?deliveryMiny, ?deliveryMax) ->drop(3) (?deliveryTimex v1:hasMinValueInteger ?deliveryMiny)]

Table 12: Rule to Match Request to Best Delivery Time

Textual Format	Jena Rule Format
Business Entity X	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [matchToBestDeliveryLeadTime: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx)

requests Offer X Offer X has best available delivery time Value X Offer X matches best with Offer Y Offer Y has delivery time Value Y If Value Y is not equal to best Value X Then remove Offer Y from the best matching Offers	(?Offerx v1:hasEligibleQuantity ?deliveryTimex) (?deliveryTimex v1:hasMinValueInteger ?deliveryx) (?Offerx comp:bestMatchingOffer ?Offery) (?Offery v1:hasEligibleQuantity ?deliveryTimey) (?deliveryTimey v1:hasMinValueInteger ?deliveryMiny) notEqual(?deliveryx, ?deliveryMiny) -> drop(4)]

Table 13: Rule to Find Best Available Rating

Textual Format	Jena Rule Format
Business entity requests an Offer X which has a best available rating value X And Offer X matches to Offer Y which provided by Business Entity Y with rating value Y if this value is greater than value X Then the requested Offer X has best available rating value equal to price value Y	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [matchRequestToBestRatings: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?Offerx comp:hasMinRating ?minRating) (?Offerx comp:bestMatchingOffer ?Offery) (?y rdf:type v1:BusinessEntity) (?y v1:offers ?Offery) (?y comp:hasRating ?ratingy) greaterThan(?ratingy, ?minRating) -> drop(2) (?Offerx comp:hasMinRating ?ratingy)]

Table 14: Rule to Match Request to Best Rating

Textual Format	Jena Rule Format
Business Entity X requests Offer X Offer X has best available rating Value X Offer X matches best with Offer Y Offer Y is offered by Business Entity Y which has rating Value Y If Value Y is not equal to best Value X Then remove Offer Y from the best matching Offers	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [matchRequestToBestOfferByRating: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?Offerx comp:hasMinRating ?minRating) (?Offerx comp:bestMatchingOffer ?Offery) (?y rdf:type v1:BusinessEntity) (?y v1:offers ?Offery) (?y comp:hasRating ?ratingy) notEqual(?ratingy, ?minRating) -> drop(3)]

As mentioned before these rules are constructed in order to provide different evaluation results if they are applied to the same offers but in a different sequence based on requesters' ranked preferences. In the case that the requester prefers price over delivery time and the Marketplace rating is its last preference the pairs of rules will be applied in the sequence that they are presented above. However, in the case that the requester

prefers delivery time as first choice, the rating as the second and the price as the last one then the Matchmaker will execute the rules pair from Table 11 and Table 12 then the rules from Table 13 and Table 14 and last the rules from Table 9 and Table 10.

Offer Level Matchmaking - Raw material service:

Several criteria have been determined for the evaluation of Raw material service offers. Each criterion owns a weighting factor, defined by the end-user, and a multi-criteria decision-making method is being implemented. Out of all the methods cited in the literature review, the Analytic Hierarchy Process was chosen. The key advantage of this method is the automated evaluation of the criteria based on the preferences provided by the user. Once the criteria weights are calculated, each alternative offer gains a weighted score, with the best one indicating the best offer.

Figure 14 is a high level illustration of the offer evaluation process, highlighting the involvement of several criteria extracted from the agent's input. The properties of the *Offer Level Request* are annotated in order to form the weighted criteria. Initially, service, *Transportation* and *Insurance Price* sum up to an overall price. *Payment Terms* and *Delivery Time* of service are extracted from the agent's input. Next, *Payment* and *Delivery Methods*, two optional properties defined by the initial request, are checked by a fulfilment rule and if they match with the offers' available methods, their value is set to 1, otherwise to 0. Similarly, a fulfilment rule is applied in the *Certification* property and its value (0 or 1) is determined. The last criterion taken into consideration is the *Rating* of a Business Entity. All these criteria enter the *Automated Criteria Weighting* mechanism where each is assigned with a weight. Finally, the evaluation scores for each offer alternative are calculated and result in the best suggested offer.

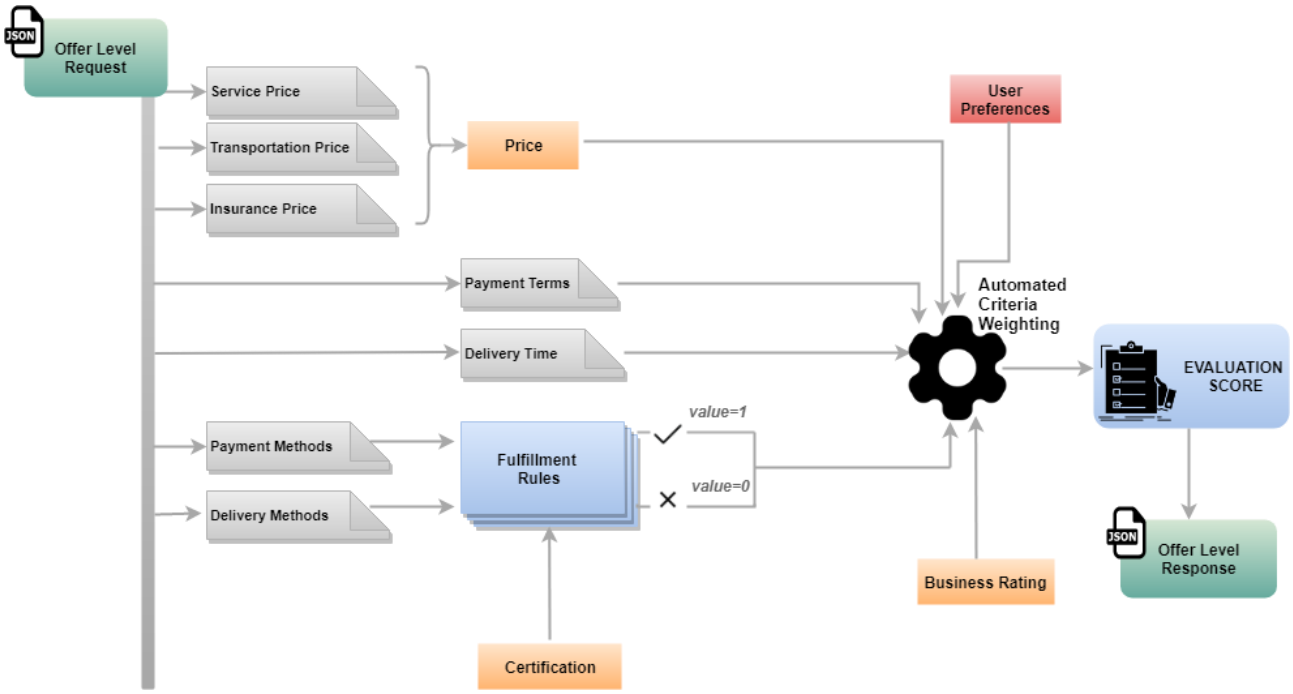

Figure 14: Offer Level evaluation process

Automated Criteria Weighting

The AHP method computes the weights of the criteria starting by a pairwise comparison matrix with MxM dimensions, where M is the number of available criteria. Each element a_{jk} of the matrix is the importance of the $j-th$ criterion in relation with $k-th$ criterion. Specifically,

- $a_{jk} > 1$, when $j-th$ criterion is more important than $k-th$ criterion
- $a_{jk} < 1$, when $j-th$ criterion is less important than $k-th$ criterion
- $a_{jk} = 1$, when $j-th$ criterion is equally important with $k-th$ criterion

The elements a_{jk} and a_{kj} satisfy the constraint:

$$a_{jk} \cdot a_{kj} = 1 \quad (3)$$

The importance between two criteria is measured according to a numerical scale from 1 to 9, as shown in Table 15, where it is assumed that the $j-th$ criterion is equally or more important than the $k-th$ criterion.

Table 15: Table of criterion relative scores

Value of a_{jk}	Meaning
1	j criterion is equally important with k
3	j criterion is slightly more important than k
5	j criterion is more important than k
7	j criterion is strongly more important than k
9	j criterion is absolutely more important than k

The relative importance of the criteria is indirectly defined by the user (*User Preferences* property). The user selects the order of preference among all the available criteria with the first being the most important and the last being the least important. There is an initial default order in case the user does not wish to choose, presented below.

The pairwise comparison matrix is filled out automatically based on the user's preferences. The normalized Eigen vectors of the pairwise comparison matrix are then calculated. The principal Eigen vector is the Eigen vector that corresponds to the highest Eigen value and is called priority vector since it contains the final weights of the criteria.

Everything being considered, the criteria are presented below, given in the following default order:

- Price (service, transportation and insurance)
- Delivery time
- Payment terms (credit)
- Business ranking
- Certificate
- Delivery methods (optional)
- Payment methods (optional)

Evaluation Score

The weighted scores are then calculated by an alternative version of equation (1) and the best score indicates the best offer. The best score is:

$$Best\ Score = \min_i \sum_{j=1}^{N} \left(\frac{q_{ij}}{\max(q_j)} w_j\right), \qquad for\ i = 1,2,3,\dots,M. \quad (4)$$

where N is the number of decision criteria, q_{ij} is the actual value of the $i-th$ offer in terms of the $j-th$ criterion, $\max(q_j)$ is the maximum in terms of the $j-th$ criterion and w_j is the weight of importance of the $j-th$ criterion. Note that the best score is the minimum of the sum, since the most important criterion usually is price and the optimal case is the minimum price. Correspondingly, the values of the criteria that are optimized with small values (e.g. Delivery time) are assigned with a positive sign, whereas the values of the criteria that are optimized with great values (e.g. Business ranking) are assigned with a negative sign.

Figure 15 is a diagram that shows the way the criteria are related to the offers with the aim of finding the best offer for the raw material service.

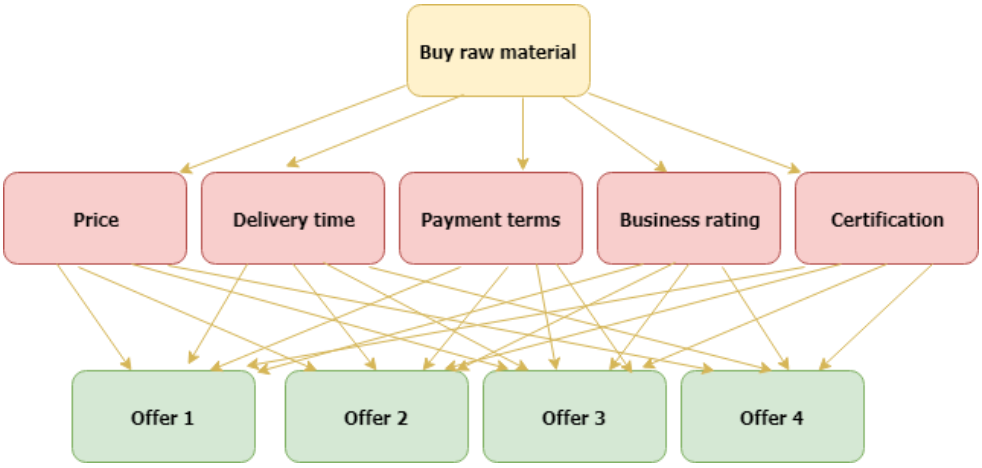

Figure 15: The evaluation criteria diagram for raw material service

At this point, a complete paradigm of Analytic Hierarchical Process is presented. The priority of preferences is the default aforementioned order and the optional properties are not taken in consideration. In accordance with Table 15, the pairwise comparison matrix is created.

Table 16: The pairwise comparison matrix and the priority vector

	Price	Delivery time	Payment terms	Business rating	Certificate	Priority Vector
Price	**1**	3	5	7	9	**51.28**
Delivery time	1/3	**1**	3	5	7	**26.15**
Payment terms	1/5	1/3	**1**	3	5	**12.9**
Business ranking	1/7	1/5	1/3	**1**	3	**6.34**
Certificate	1/9	1/7	1/5	1/3	**1**	**3.33**

Briefly, the Price criterion is slightly more important than the Delivery time criterion ($a_{12} = 3$), more important than the Payment terms criterion ($a_{13} = 5$), strongly more important than Business rating criterion ($a_{14} = 7$) and absolutely more important than Certificate criterion ($a_{15} = 9$). Obviously, the diagonal elements are equally important with themselves ($a_{11} = 1$). The lower triangular matrix is filled by the equation (3). The rest of the matrix is filled respectively.

Next, the normalized Eigen vectors are calculated and the principal Eigen vector gives the Priority Vector, hence the weight for each criterion presented in the last column of Table 16. Finally, Table 17 shows the four offers with their values and the extraction of best score. Offer 2 raised the best score apparently, although it does not have the best price or the best ranking, it owns the best delivery time value and the lowest payment term value.

Table 17: A paradigm of best score calculation

WEIGTHS:	**0.51**	**0.26**	**0.13**	**0.65**	**0.35**	
	Price (€)	**Delivery time (Days)**	**Payment terms (Days)**	**Business rating (1-5)**	**Certificate (0/1)**	**SCORE**
Offer 1	222	2	60	3	**0**	**0.57**
Offer 2	210	1	50	4	**1**	**0.45**
Offer 3	195	3	100	3	**0**	**0.55**
Offer 4	212	2	60	5	**1**	**0.51**

8 Matchmaker Quality Control, Scalability and Security

8.1 Quality Control

A quality control plan has been followed during the Task 6.5 and its development processes. Before, the implementation face, a thorough analysis of related works and state-of-the-art was conducted.

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

- Identification of the requirements related to the Matchmaker
- 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 D2.4-The COMPOSITION architecture specification II. These will ensure the compatibility with other project's components.
- Use of software tools which were proposed at D1.1 & D1.2 Project Quality Control Plan I & II and support quality of software:
    - Use of Eclipse IDE[1] as the development environment
    - Use of Git for control versioning. The EGit[2] plugin from Eclipse IDE was used.
    - Use of Maven[3] as build tool for dependency management and build of source code
    - Use of Docker[4] for deployment
- Test procedures were applied. For software quality assurance both static and dynamic analysis techniques applied:

  In *static analysis* the PMD[5] tool was used. It is an open source tool which offers source code analysis and offered as an Eclipse IDE plugin. 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

  About 300 rules from 20 different rules sets were used. The rules sets were the following: Basic, Basic POM, Braces, Code size, Complexity, Controversial, Design, Empty code, Import statements, J2EE, Junit, Naming, Optimization, Security code guidelines, Strict Exceptions, String & StringBuffer, Style, Unnecessary and Unused code.

  The analysis results were evaluated during the development face and the most important possible bugs were handled. At the current version of code there are no block, critical, important and warning violations. Currently there are only few urgent violations which are related to excessively long variable names, variables with short names, multi occurrences of some string literals etc. These violations are considered as false positives.

  In *dynamic* analysis, tests in runtime have been executed. Generally, in dynamic analysis Unit tests, Integration tests and System tests were executed.

  We built automated tests in 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 project at an external and using an external HTTP client. We used Eclipse Jetty server which provides a Web server and javax servlet container. 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. The tests were called both separately or in combination.

[1] https://www.eclipse.org/ide/
[2] http://www.eclipse.org/egit/
[3] https://maven.apache.org/
[4] https://www.docker.com/
[5] https://pmd.github.io/

Moreover, the Matchmaker was tested in integration with the Marketplace agents. Both Matchmaker and agents were deployed as Docker images. The Agents image was able to call successfully the Matchmaker's APIs services from the deployed image as well. Furthermore, the correctness of the Matchmaker responses was checked too. More details about agents and Matchmaker interaction, Matchmaker APIs and the deployment of the component are presented to the following chapter.

8.2 Scalability

The COMPOSITION Matchmaker has been designed in order to offer high performance in matching processes and support large Marketplaces with numerous of participants and services. It is designed after a thorough research for available tools, technologies, related works and methodologies as it was documented in previous sections of this deliverable.

As the Matchmaker component (Matchmaker, Ontology Query API and Ontology Store) 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 [6] the server is able to support over than 8000 connections.

Furthermore, an 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. 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. A native triple store is faster 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 and its included components was tested for the COMPOSITION use cases such as UC KLE-4 and the online bidding process. The Matchmaker responses in a reasonable time (less than 5 seconds). However, in order to examine the performance of some 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 considered as a serious problem as the Marketplaces was initialized once and after that every new instance is added as soon as a new company arrives at the Marketplace or offers a new service etc.

8.3 Security

COMPOSITION Matchmaker exposes its endpoints as RESTful web services. These services 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)[7] can secure these APIs such as the one from the 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

[6] http://tomcat.apache.org/tomcat-8.5-doc/config/http.html
[7] https://docs.linksmart.eu/display/BGW

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[8] framework that a component is authorized to call both GET and POST methods on Matchmaker endpoints.

[8] https://www.keycloak.org/

9 Matchmaker APIs and Deployment

In this chapter the services offered by the Matchmaker API are presented. Furthermore, the deployment of the Matchmaker component is presented as well.

9.1 Matchmaker API Web Services

The Matchmaker is connected with the Marketplace agents through RESTful web services and HTTP protocol. An API is offered to the agents. The implemented Matchmaker API contains two web services. As depicted in Figure 16 below, both of the offered web services are POST requests

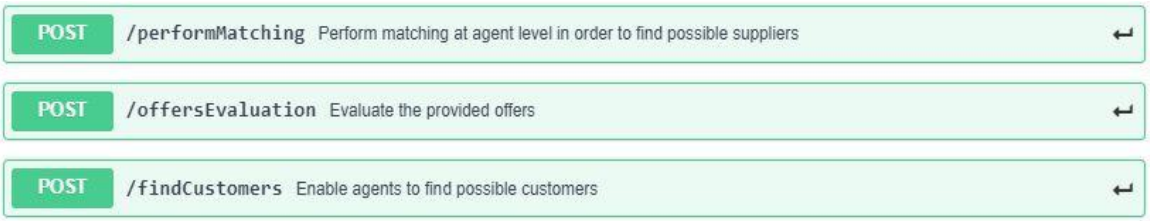

Figure 16: Matchmaker API Services

This web service was designed in order to support the online bidding processes over the COMPOSITION Marketplace. Based on the request's body the matchmaker decides if it is going to apply Offer Level or Agent Level matchmaking as both are required in a bidding process. The collaboration scheme of the agents and the Matchmaker presented to the following figure:

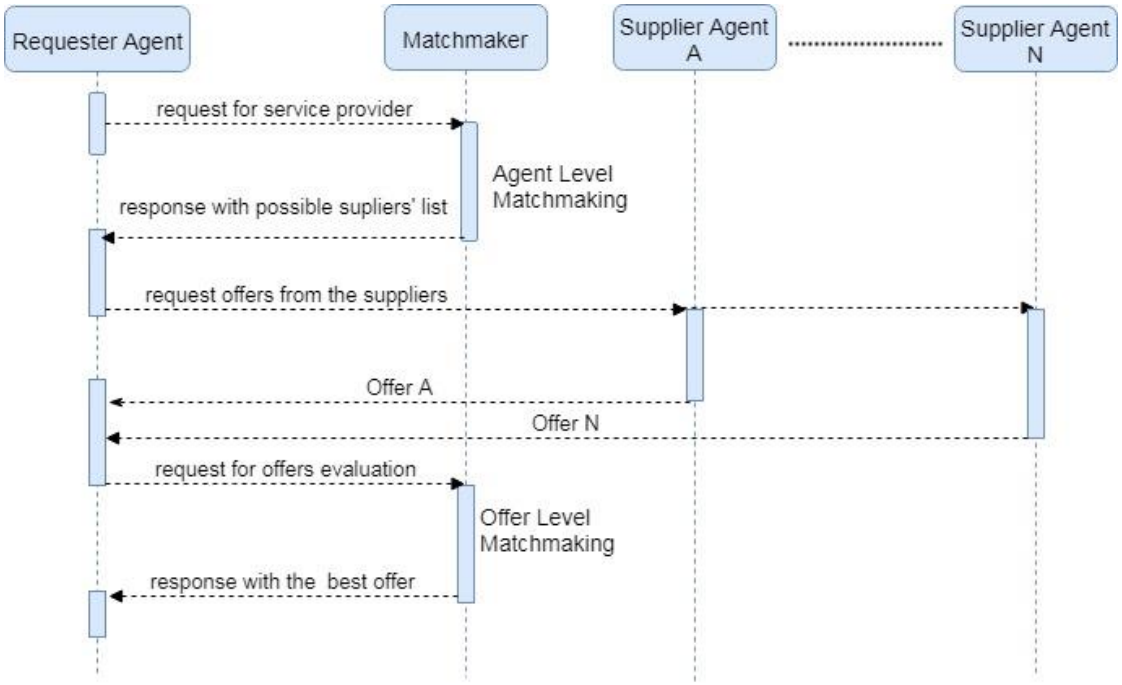

Figure 17: Matchmaker and Agents Communication

9.1.1 Service "performMatchmaking"

As mentioned before the request's body defines the type of the matchmaking which will be triggered. The body is defined in JSON in a format compatible with the agents' CXL. In order to trigger the Agent Level matchmaking, the following body is posted to the Matchmaker:

```
conversation_id  string
                 example: kjhfewKJDGWHJGWH7856186GBFWE
                 required: true
```

```
sender_id          string
                   example: agent_req_1
                   required: true
agent_owner        string
                   example: KLEEMANN
                   required: true
type               string
                   example: OFFER
                   required: true
service            string
                   example: Provide_raw_materials
                   required: true
offer_details      {
                   good              string
                                     example: Tube
                                     required: true
                   expiration        string
                                     example: 2017-06-07T24:00:00+01:00
                                     required: true
                   currency          string
                                     example: USD
                                     required: true
                   quantity          double
                                     example: 120.0
                                     required: true
                   quantity_uom      string
                                     example: tons
                                     required: true
                   delivery_methods  string array
                                     example: ["Delivery mode freight", “DHL”]
                                     required: false

                   payment_methods   string array
                                     example: [ "Direct debit", "Cash"]
                                     required: false
                   }
offers             [                 Json array
                                     required: true (empty if “type” :  “CFP”)
                   ]
```

Figure 18: Request Body for Agent Level Matchmaking

In the abovementioned request’s body:

- *conversation_id* is the unique id of the conversation allows to track request / reply sequences
- *sender_id* is the id of the requester agent
- *agent_owner* describes the business entity’s agent generating the message
- *type* describes the type of the request and defines the level of matchmaking (Agent or Offer)
- *service* describes the type of service requested, either Waste_management or Provide_raw_material service
- *offer_details* object describes the details of the request such as the good/service type that is requested, the expiration date of the request, the currency, the requested quantity and its corresponding unit of measurement. Also, delivery and payment methods are optional and describe additional requirements that the requester can set in order to receive a more accurate matchmaking result based on services capabilities were described at the Collaborative Manufacturing Services Ontology.

The Agent Level matchmaking sends back the response with the matching business entities. As depicted in Figure 19 the response's body contains the agent details (*conversation_id, agent_owner, sender_id*) and the details of the matching business entities (*agent_owner* and *sender_id*).

```
conversation_id     string
                    example: kjhfewKJDGWHJGWH7856186GBFWE
                    required: true
sender_id           string
                    example: agent_req_1
                    required: true
agent_owner         string
                    example: KLEEMANN
                    required: true
matching            [
BusinessEntities
                    {
                    agent_owner                     string
                                                    example: Company_A
                                                    required: true
                    sender_id                       string
                                                    example: agentA@composition
                                                    required: true
                    }
                    ...
                    ]
```

Figure 19: Response of Agent Level Matchmaking

9.1.2 Service "offersEvaluation"

The Offer Level requests consists of both the Agent-Level request and the provided offers details, as shown in Figure 20.

```
conversation_id     string
                    example: kjhfewKJDGWHJGWH7856186GBFWE
                    required: true
sender_id           string
                    example: agent_req_1
                    required: true
agent_owner         string
                    example: KLEEMANN
                    required: true
type                string
                    example: OFFER
                    required: true
service             string
                    example: Provide_raw_materials
                    required: true
offer_details       {
                    good                string
                                        example: Tube
                                        required: true
                    expiration          string
                                        example: 2017-06-07T24:00:00+01:00
                                        required: true
                    currency            string
                                        example: USD
                                        required: true
```

```
                quantity          double
                                  example: 120.0
                                  required: true
                quantity_uom      string
                                  example: tons
                                  required: true
                delivery_methods  string array
                                  example: ["Delivery mode freight", “DHL”]
                                  required: false

                payment_methods   string array
                                  example: [ "Direct debit", "Cash"]
                                  required: false
                }
offers          [                 Json array
                                  required: true (empty if “type” :  “CFP”)
                {
                offer_details     {           Json object
                                              required: false
                                  sender_id   string
                                              example: agent_supplier_1
                                              required: true
                                  agent_owner string
                                              example: Company_A
                                              required: true
                                  good        string
                                              example: Tube
                                              required: true
                                  delivery    {
                                              time          double
                                                            example: 2
                                                            required: true
                                              methods       string array
                                                            example: ["Delivery mode
                                                            freight", “DHL”]
                                                            required: true
                                              }
                                  payment     {
                                              methods       string array
                                                            example: [ "PayPal",
                                                            "Direct debit", "Discover",
                                                            "Cash"]
                                                            required: true
                                              terms         double
                                                            example: 90
                                                            required: true
                                              currency      string
                                                            example: USD
                                                            required: true
                                              }
                                  price       {
                                              service       double
                                                            example: 120
                                                            required: true
                                              insurance     double
                                                            example: 30
                                                            required: true
```

```
                                        transportation  double
                                                        example: 20
                                                        required: true
                                        }
                                }
                }
                ...
                ]
preferences     {               Json object
                                required: true
                priority_1      integer (1-7)
                                example: 2
                                required: false
                priority_2      integer (1-7)
                                example: 5
                                required: false
                ...

                priority_7      integer (1-7)
                                example: 1
                                required: false
                }
```

Figure 20: Request Body for Offer Level Matchmaking

The *offers* field is a mandatory field which is empty in case of Agent Level request or contains the available Offers with the corresponding details in case of Offer Level request. In order to call the Offer Level of matchmaking in the request body the *type* property is set as “OFFER”. The array containing the offers which were provided by the supplier agents is added to the body object with the following properties:

- *sender_id* is the id of the supplier agent
- *agent_owner* describes the business entity’s agent generating the offer
- *good* is the type of good which is provided by the business entity
- *delivery* including *methods* and *time*, are the details of delivery of the offered good
- *payment* including *methods*, *currency* and *terms*, are the details of payment for the offered good
- *price* including *service*, *insurance* and *transportation,* are the details of the cost of the offered good

The ranked preferences of the Offer Level matchmaking are added as well, defining the priority of the following criteria, each with a number assigned:

1. Price
2. Delivery time
3. Payment terms
4. Business ranking
5. Certificate
6. Delivery methods (optional)
7. Payment methods (optional)

The *preference* property includes all the above criteria for the Raw_material service and the first three for the Waste_management service.

Respectively, Figure 21 presents the response body of an Offer Level Matchmaking request. Specifically, it contains the request details and the suggested offer produced by the Multi-criteria decision-making module of the Matchmaker, along with the corresponding information. The offer_details property is exactly the same as in the request body of Offer Level Matchmaking (Figure 20).

```
Request:  json
          Required:true
          {
          conversation_id   string
                            example: kjhfewKJDGWHJGWH7856186GBFWE
                            required: true
          sender_id         string
                            example: agent_req_1
                            required: true
          agent_owner       string
                            example: KLEEMANN
                            required: true
          suggested_offer   [              Json array
                                           required: true
                            offer_details  {            Json object
                                                        required: false
                                           }
                            ...
                            ]
```

Figure 21: Response of Offer Level Matchmaking

9.1.3 Service "findCustomers"

This web service is designed in order to enable agents to find possible customers for their services in the COMPOSITION Marketplace. This functionality is related to Atlantis use case scenarios in which the Marketplace should offer solutions to its participant. This service offers the opportunity to the Marketplace participants to advertise its services and products to possible customers. The functionality of this service was presented in more details at chapter 6 of this report. The request's body schema is presented to the following figure:

```
conversation_id   string
                  example: kjhfewKJDGWHJGWH7856186GBFWE
                  required: true
sender_id         string
                  example: agent_req_1
                  required: true
agent_owner       string
                  example: KLEEMANN
                  required: true
Request_type      string
                  example: KLEEMANN
                  required: true
```

Figure 22: Request Body for findCustomers Service

The response of the findCustomers services has the same schema as the Agent Level Matchmaking response presented at Figure 19.

Figure 23 depicts the supported services that can be requested by an agent in the COMPOSITION ecosystem and the specific goods/products that are available for each service.

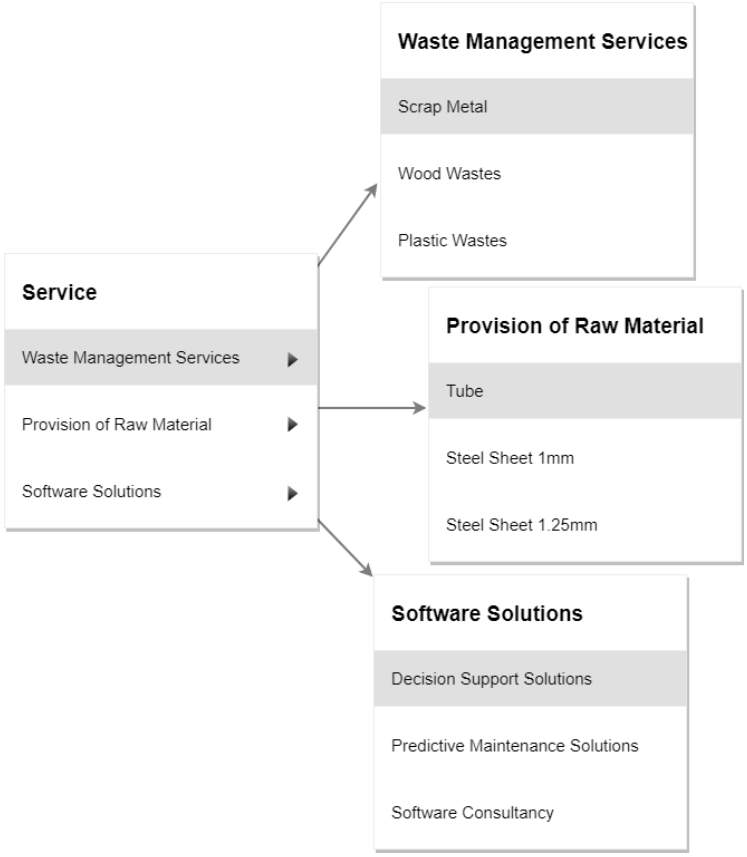

Figure 23: Matchmaker supported services and goods

9.2 Matchmaker Deployment

The Matchmaker component was decided to be deployed as a Docker image as the rest of the project's component based on the Deployment View of D2.3 The COMPOSITION architecture specification I.

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's Docker image contains the complete component as it is described at Figure 1 at chapter 4. In this image the Rule-based Matchmaker, the Query Engine, the Ontology Store and their corresponding APIs are containing.

In order to create the Matchmaker's Docker image and the corresponding container the official Docker image for Apache Tomcat (Apache Tomcat, 2018) was used. Tomcat was selected as the web server environment as it is web server environment in which Matchmaker'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 inter-factory Portainer (Docker, 2018) which offers management of Docker environments. A view of the COMPOSITION inter-factory Portainer which is related to Marketplace components presented at Figure 24.

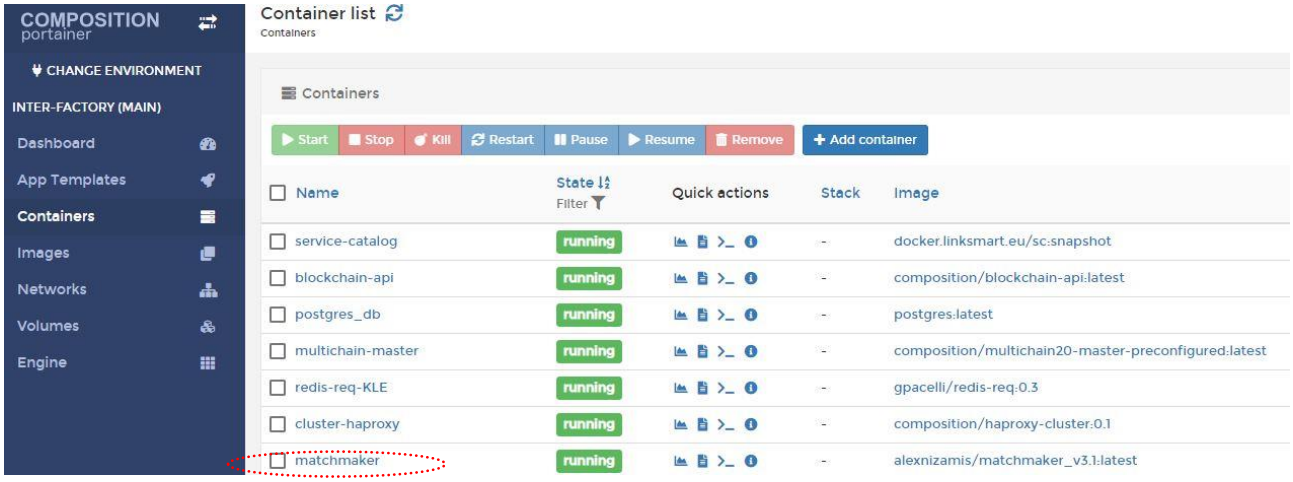

Figure 24: COMPOSITION Inter-factory Production Server

10 Conclusions

In conclusion, this deliverable describes the effort spent from M5 to M34 and represents the final outcome of Task 6.5 - Brokering and Matchmaking for Efficient Management of Manufacturing Processes of WP6. Moreover, this report documents the implemented COMPOSITION Matchmaker. Some minor updates are possible as part of the continuous evaluation of the complete system by the end of the project (M36) and they can be related with the UIs modifications and updates.

The COMPOSITION Matchmaker has been implemented and presented after an analysis of related works and available tools and technologies. Moreover, the implemented version of the Matchmaker was presented in this report with emphasis on semantic rules as it is a rule-based matchmaker which infer new knowledge by applying rules. Furthermore, the enhancements in offers evaluation which are based on weighted scores algorithms are presented as well.

After consideration of project's requirements and architecture, and after an analysis of available technologies and tools, a Matchmaker API is developed in Java and it is offered through RESTful web services. It provides to the Marketplace agents access to the matchmaking functionalities. The last working version of the Matchmaker component which contains the Rule-based Matchmaker and its corresponding API, the Ontology API and the Ontology Store has been deployed as a Docker container in the COMPOSIITON inter-factory container. This deployment enables the usage of these components by the Marketplace agents in a secure way by using the provided capabilities of the implemented Security Framework from WP4.

The outcome of this deliverable mainly affects the WP6 and its components, the agents. By using the Matchmaker services the agents are able to execute automated bidding processes in the Collaborative Ecosystem or to find possible future customers within this ecosystem. In Task 6.5 were used already known technologies such as semantics and rules, but they were applied in a Manufacturing Marketplace for matching and evaluating offers in real-time which is not so usual. The implemented system was able to extend the usage of Ontology. It was not used only for interoperability, but it is used also for real-time decision-making capitalizing on knowledge inference. Furthermore, the COMPOSITION Matchmaker enhance its evaluation capability by adopting weighted scores algorithms.

Finally, as it is perceived, the Matchmaker package is a complete system that can support online manufacturing ecosystems that are focused on requests for suppliers and online negotiations. Further research and development will be conducted for this component in future research projects (such as eFactory EU project). There, the work has done in the COMPOSITION Matchmaker will be extended and its functionalities will be combined with machine learning techniques which are going to upgrade the component's intelligence.

11 List of Figures and Tables

11.1 Figures

11.2 Tables

12 References

(Ameri, 2012) Ameri, F. and Patil, L. (2012). Digital manufacturing market: a semantic web-based framework for agile supply chain deployment. Journal of Intelligent Manufacturing, 23(5), 1817-1832.

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

(Apache Jena, 2018) Retrieved from JENA: https://jena.apache.org/documentation/inference/

(Apache Maven, 2018) Apache Maven: https://maven.apache.org/

(Apache Tomcat, 2018) Apache Tomcat: http://tomcat.apache.org/

(Belton and Gear,1983) Belton, V., and T. Gear, "On a short-coming of Saaty's method of analytic hierarchies," Omega, 228-230, 1983.

(Benayoun, et al., 1966) Benayoun, R., B. Roy, and N. Sussman, "Manual de reference du programme electre, Note de Synthese et Formation," No. 25, Direction Scientifique SEMA, Paris, Franch, 1966.

(Docker, 2018) Docker: https://www.docker.com/

(Fishburn, 1967) Fishburn, P.C., Additive Utilities with Incomplete Product Set: Applications to Priorities and Assignments, Operations Research Society of America (ORSA) Publication, Baltimore, MD, 1967.

(FITMAN-SeMa, 2018) Retrieved from http://www.ware4industry.com/?portfolio=metadata-and-ontologies-semantic-matching-sema/

(FIWARE, 2018) Retrieved from FIWARE: http://www._ware4industry.com/

(GoodRelations Language, 2018) Retrieved from GoodRelations Language: http://www.heppnetz.de/projects/goodrelations

(Hwang and Yoon,1981) Hwang C.L. and K. Yoon, Multiple Attribute Decision Making: Methods and Applications, Springer-Verlag, New York, NY, 1981.

(Keycloak, 2018) Retrieved from Keycloak: https://www.keycloak.org/

(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/

(Miller and Starr, 1969) Miller, D.W., and M.K. Starr, Executive Decisions and Operations Research, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1969.

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

(Nodine, 2000) Nodine M., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M., and Unruh, A. (2000). Active information gathering in InfoSleuth. International Journal of Cooperative Information Systems, 9(1/2)

(Roy, 1973) Roy, B., "How the outranking relation helps multiple criteria decision making." In: Multiple Criteria Decision Making, Cochrane and Zeleny (Eds.), University of South Carolina Press, SC, 179-201, 1973.

(Saaty, 1994) Saaty, T.L., Fundamentals of Decision Making and Priority Theory with the AHP, RWS Publications, Pittsburgh, PA, U.S.A., 1994.

(Sycara, 1999) Sycara, K., Klusch, M., Wido, S., and Lu, J. (1999). Dynamic Service Matchmaking Among Agents in Open Information Environments, volume 28, 47-53. ACM, New York, NY, USA.

13 ANNEX

Table 18: Rule for Certification Fulfilment

Textual Format	Jena Rule Format
Business Entity X requests an Offer X And Business Entity X matches with Business Entity Y Which has an Offer that includes a Service with an Operation allowed for a Material that has certification Cert Y Then Offer Y has a certification fulfillment	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [certificationFulfillment: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?x comp:matchesWith ?y) (?y v1:offers ?Offery) (?Offery v1:includes ?Servicey) (?Servicey comp:hasOperationy ?Operationy) (?Operationy p1:allowedProcessFor ?Materialy) (?Materialy comp:hasCertification ?Certy) -> (?Offery comp:hasCertificationFulfillment ?Certy)]

Table 19: Rule for Payment Methods Matching

Textual Format	Jena Rule Format
Business Entity X requests an Offer X that has price specification Price Spec X that applies to Payment Method X And Business Entity X matches with Business Entity Y Which has an Offer that has accepted Payment Method X Then Offer Y has a payment method match	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [PaymentMethodsMatch: (?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?Offerx v1:hasPriceSpecification ?PriceSpecx) (?PriceSpecx v1:appliesToPaymentMethod ?Methodx) (?x comp:matchesWith ?y) (?y v1:offers ?Offery) (?Offery v1:hasAcceptedPaymentMethod ?Methodx) -> (?Offery comp:hasMatchingPaymentMethod ?Methodx)]

Table 20: Rule for Delivery Methods Matching

Textual Format	Jena Rule Format
	@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>. @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>. @prefix comp: <http://www.composition-project/ontologies/COMPOSITIONv01#>. @prefix v1: <http://purl.org/goodrelations/v1#>. @prefix p1: <http://www.owl-ontologies.com/mason.owl#>. @prefix MSDL: <http://www.composition-project.eu/ontologies/MSDL#>. [DeliveryMethodsMatch:

Business Entity X requests an Offer X that is available for Delivery Method X And Business Entity X matches with Business Entity Y Which has an Offer that has Available Delivery Method X Then Offer Y has a delivery method match	(?x rdf:type v1:BusinessEntity) (?x v1:seeksOffer ?Offerx) (?Offerx v1:hasAvailableDeliveryMethods ?Methodx) (?x comp:matchesWith ?y) (?y v1:offers ?Offery) (?Offery v1:hasAvailableDeliveryMethods ?Methodx) ->(?Offery comp:hasMatchingDeliveryMethod ?Methodx)]