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

D2.3 The COMPOSITION architecture specification I

Date: 2017-07-17

Version 1.1

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:	D2.3 The COMPOSITION architecture specification 1.1.docx
Document version:	1.1
Document owner:	COMPOSITION
Work package:	WP2 Use Case Driven Requirements Engineering and Architecture
Task:	Task 2.3 COMPOSITION Architecture
Deliverable type:	R
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	Mathias Axling	2017-03-28	Initial version
0.11	Mathias Axling	2017-05-18	Workshop updates
0.12	Dario Bonino	2017-06-01	Added required contributions
0.2	Dimosthenis Ioannidis, Nikolaos Kaklanis, Alexandros Nizamis	2017-06-09	Functional view descriptions: Simulation and forecasting tool, Matchmaker. Information view descriptions: Digital Factory Model, Marketplace Ontology
0.3	Paolo Vergori, Matteo Pardi, Gianluca Insolvibile, Jesús Benedicto	2017-06-16	ISMB, NXT, ATOS contributions added
0.4	Javier Romero	2017-06-19	ATOS security contribution added
0.5	Mathias Axling	2017-06-21	Merged contributions, added content
0.6	José Ángel Carvajal Soto, Junhong Liang	2017-06-23	FIT contributions
0.7	Mathias Axling	2017-06-26	Edited version, added content
0.8	Mathias Axling, Peeter Kool, Matts Ahlsen	2017-06-28	Edited version, added content
0.9	Mathias Axling, Peter Rosengren	2017-06-29	Contribution to be added
0.91	Mathias Axling	2017-06-29	Ready for peer review
1.0	Mathias Axling, Peter Rosengren	2017-07-10	Peer Review Comments incorporated
1.1	Peter Rosengren	2017-07-17	Updated conceptual architecture. Final Version

Internal review history:

Reviewed by	Date	Summary of comments
Dimitris Gkortzis, ELDIA	2017-07-07	Minor comments
Martina Beer, FIT-WI	2017-06-30	Minor comments

Index:

1 Executive Summary

In this deliverable, the first version of the software architecture for the COMPOSITION project is described.

COMPOSITION has two main goals: The first goal is to integrate data along the value chain inside a factory into one integrated information management system (IIMS) combining physical world, simulation, planning and forecasting data to enhance re-configurability, scalability and optimisation of resources and processes inside the factory to optimise manufacturing and logistics processes.

The second goal is to create a (semi-)automatic ecosystem, which extends the local IIMS concept to a holistic and collaborative system incorporating and interlinking both the supply and the value Chains. This should be able to dynamically adapt to changing market requirements.

The objectives are achieved by the use of number of IoT enabling technologies and services together with sophisticated big data analytics and deep learning as well as a trusted framework based on blockchain technology. The main services realised by COMPOSITION are:

- Material and Component Tracking
- Product Quality Monitoring
- Manufacturing Forecasting
- Automated Procurement
- Ecosystem Collaboration Framework

The COMPOSITION architecture has been designed with consideration to compliance with RAMI 4.0 (Reference Architecture Model Industrie 4.0).

1.1 Content and structure of this deliverable

The deliverable closely follows the structure outlined by the selected documentation approach. The remainder of the document is structured as follows:

Section 2 - Terminology: defines the terminology specific to the COMPOSITION domain.

Section 3 - Introduction: identifies the purpose, scope and context of the deliverable, and the architecture design and description methodology used.

Section 4 – Stakeholders, provides an overview of the stakeholders, concerns, and requirements that drive the architecture design.

Section 5 - Architectural views: documents the architecture in four views: Context, Functional, Information, and Development.

Section 6 - System Quality Perspectives: documents quality attributes cross-cutting several views in two architecture perspectives: Security and Scalability.

Section 7 - Summary and future work: presents a summary of the current state of architecture development and how future architecture design will proceed.

2 Terminology

The currently adopted domain-specific terminology used in the remainder of the document is presented in Table 1. COMPOSITION-specific terminology below.

Table 1. COMPOSITION-specific terminology.

Term	Definition
Agent Container	An agent container is a set of intelligent agents interacting through the same, shared transport protocol and referring to shared platform services such as the Directory Facilitator, DF and the Agent Management Service, AMS.
COMPOSITION Marketplace	A COMPOSITION Marketplace is an agent container.
Closed Marketplace	COMPOSITION Marketplace owned by one stakeholder and typically offered to a trusted subset of other COMPOSITION stakeholders. The Closed Marketplace can be public or private. A public, closed market will accept join requests by agents living in the Open Marketplace A private, closed marketplace will accept agents only by invitation. A Closed Marketplace is structurally equivalent to the open marketplace A Closed Marketplace is physically separated to the Open Marketplace and has typically a separate infrastructure of shared platform services including the broker, AMS, DF, etc.
Virtual Marketplace	A Virtual Marketplace, or group is a "multicast" group of agents interacting with each other in the context of a negotiation. The group can be: – persistent over negotiations or – just be defined for a single negotiation exchange. A Virtual Marketplace lives in, and exploits the infrastructure of the Open Marketplace.
Integrated Information Management System (IIMS)	The Integrated Information Management System is a digital automation framework that optimizes the manufacturing processes by exploiting existing data, knowledge and tools to increase productivity and dynamically adapt to changing market requirements.
COMPOSITION Ecosystem	The supply chain part of a COMPOSITION system, implemented by a COMPOSITION Marketplace and involving suppliers, producers and logistics services.
Supply chain	The sequence of processes involved in the production and distribution of a commodity

Value chain	The process or activities by which a company adds value to an article, including production, marketing, and the provision of after-sales service.

3 Introduction

The results of the software architecture design activities for the COMPOSITION system is reported in this deliverable, D2.3: “The COMPOSITION architecture specification I”. It provides a description of the current state of the architecture for the M2 milestone in month 10 in the project. This will be followed by the D2.4 “The COMPOSITION architecture specification II”, providing an updated description at month 24. The system architecture design activities are carried out in Work Package 2 (WP2), “Use Case Driven Requirements Engineering and Architecture”, in the COMPOSITION work package structure defined by the project specification (COMPOSITION, 2016).

3.1 Purpose, context and scope of this deliverable

The purpose of this report is to provide a high-level overview of the COMPOSITION system: document the main elements of the system and the relations between these elements. It also documents the design decisions that affect the system on an architectural level and the stakeholder concerns - expressed in the project specification and user requirements – that drive the architecture design. Detailed descriptions of the elements of the architecture, e.g. the Security Framework, Decision Support System or Digital Factory Model, will be available as separate deliverables as outlined in the project specification. The reader should refer to these for implementation details and specifications. This deliverable will focus on the fundamental concepts and properties of the COMPOSITION system.

Several key functional requirements and architectural constraints are defined in the project specification, available at the start of the project. Gathering and validation of requirements and definition of pilot scenarios and use cases have been performed in parallel to the architecture definition process. The results of these activities have been reported in D2.1 “Industrial use cases for an Integrated Information Management System” and D2.2 “Initial requirements specification”, which have provided input to the architecture design activities in WP2. The D2.5 report “Lessons Learned and updated requirements report I” will provide an update of the requirements which will provide input to D2.4.

3.2 Architectural Design and Documentation Approach

The documentation will adhere to the IEEE 42010 standard, using several viewpoints to frame the concerns of the system stakeholders and illustrate the design decisions taken. Specifically, the IEEE 42010 compliant framework presented in (Rozanski & Woods, 2012) will be used. This has been extended with the concept of perspectives, which are used to evaluate quality attributes cross-cutting several viewpoints, e.g. security, evolvability or scalability.

The architecture reference model RAMI 4.0, developed in the Industrie 4.0, is used for integration of research and technical development efforts in the area of industrial IoT. This collaboration and integration with other initiatives is a strategic objective of the project (COMPOSITION, 2016).

3.2.1 Methodology

The inception phase (Kruchten, 2004) of the architecture design is documented in the project specification, which introduces several canonical architectural elements connected to technical objectives, tasks and deliverables, providing a basic functional decomposition of the system. An initial list of system components was derived from this source in architecture workshops early in the project. Developing and integrating these components is necessary to ensure that the strategic and technical objectives of the project can be met.

In subsequent workshops, this bottom-up design approach has been complemented by additional components and design decisions on standards and architectural mechanisms (Kruchten, 2004) to integrate the components. The design of individual components has been carried out in parallel to the architecture design. Evaluation and revision of this design is conducted continuously in workshops and design meetings (no formal architecture evaluation has been performed). As the components mature and feedback from pilot development and revised requirements are produced, the architecture design will be predominantly top-down. With the component design in place, strategies and mechanisms for scalability, evolvability and other quality attributes will be can be elaborated.

The COMPOSITION architecture design process and the architecture description in this document follows the ISO/IEC/IEEE 42010 “System and software engineering – Architecture description” (ISO/IEC/IEEE42010, 2011), which superseded the IEEE 1471 “Recommended Practice for Architectural

Description for Software Intensive Systems" (IEEE, 2000). See the conceptual model of architecture descriptions from (ISO/IEC/IEEE42010, 2011) below.

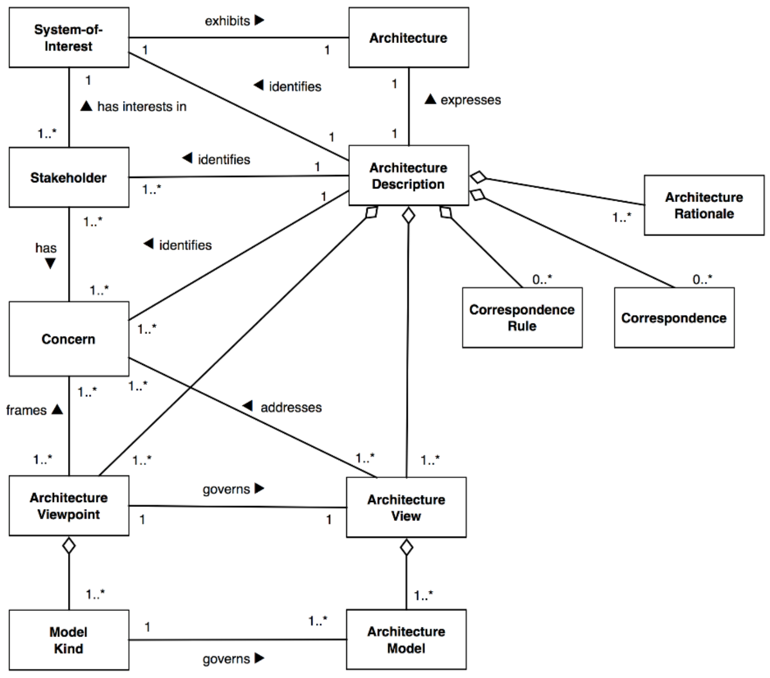


Figure 1. ISO/IEC/IEEE 42010 Architecture Description Conceptual Model[1].

As can be seen from the ISO/IEX/IEEE 42010 conceptual model of architecture descriptions, a viewpoint uses a set of model kinds to frame a specific set of concerns that stakeholders have about a system. However, quality properties such as security, performance or availability need to be considered across several viewpoints. In (Rozanski & Woods, 2012), the complementary concept of architectural perspectives is introduced to address these cross-cutting concerns.

For this first version of the architecture documentation we have addressed the system design from four viewpoints – context, functional, information, and deployment - and two perspectives, the security perspective and the scalability perspective. Further on in the architecture design process, we expect to employ other viewpoints, e.g. operational and concurrency, and address other quality properties, e.g. evolvability.

3.2.2 Reference Architecture Model Industrie 4.0

In COMPOSITION, the he Reference Architectural Model Industrie 4.0 (RAMI 4.0)[2] will be adopted to communicate the scope and design of the system, to further collaboration and integration with other relevant initiatives by framing the developed concepts and technologies in a common model.[3]

3.2.2.1 **Background & purpose**

RAMI 4.0 is a reference architecture model for Industrial Internet of Things (IIoT). The first version has been developed by the Industrie 4.0 platform and submitted as DIN SPEC 91345. RAMI 4.0 is modeled on Smart Grid Architecture Model (SGAM), IEC 62262, Enterprise-control system integration (IEC62264, 2013) and the IEC 62890 "Life-cycle management for systems and products used in industrial-process measurement, control and automation" (IEC, 2013). The focus of RAMI 4.0 is on manufacturing, primarily modelling systems for the production process and product life cycle.

[1] http://www.iso-architecture.org/42010/cm/
[2] https://www.zvei.org/fileadmin/user_upload/Themen/Industrie_4.0/Das_Referenzarchitekturmodell_RAMI_4.0_und_die_Industrie_4.0-Komponente/pdf/5305_Publikation_GMA_Status_Report_ZVEI_Reference_Architecture_Model.pdf
[3] Pictures in this section copyright "Umsezungsstrategie Industrie 4.0 – Ergebnisbericht, Berlin, April 2015"

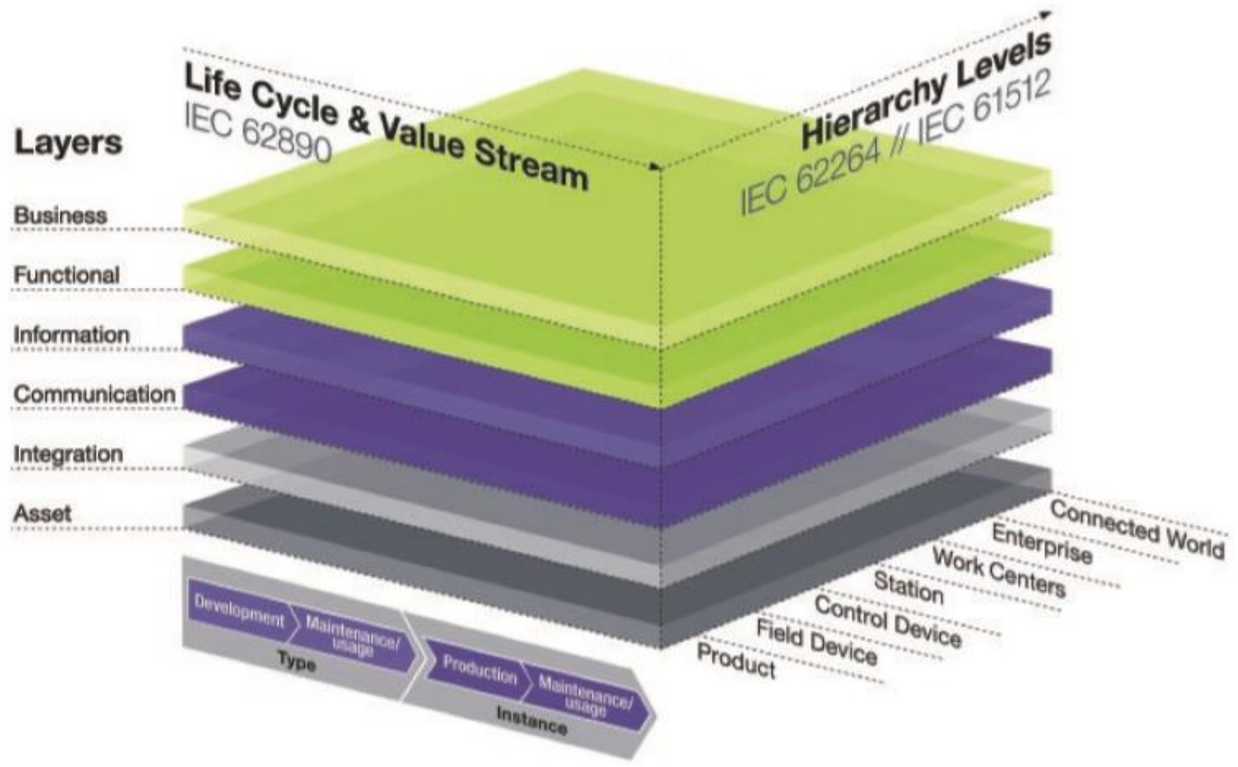


Figure 2. The three dimensions of the RAMI 4.0. (Status Report Reference Architecture Model Industrie 4.0 (RAMI4.0), 2015).

In the three dimensional model, existing standards and architectures and candidate solutions can be plotted, overlaps and gaps can be identified and resolved. It provides a map of Industry 4.0 components, solutions and requirements by the three axes IT Layers, Hierarchy Levels and Life Cycle and Value Stream.

The purpose of the reference architecture model is to promote common understanding of different architectures for industry 4.0. It can be used to derive specific architecture models and align existing solutions. Examples of applications are:

- Provide a shared understanding of the function provided by every layer and the defined interfaces between the layers.
- To see where existing and emerging architectures fit in, and allow discussing associations and details of components.
- Identification of overlaps and the scope of preferred solutions
- Identification of existing standards, closure of gaps and loopholes in standards, minimization of the number of standards involved
- Identification of use cases for Industry 4.0

3.2.2.2 **Model**

The six layers on the vertical axis represent a layered IT system structure, with loose coupling between the layers and high cohesion within each layer. The layering is strict; i.e. components in a layer may only communicate internally or with adjacent layers.

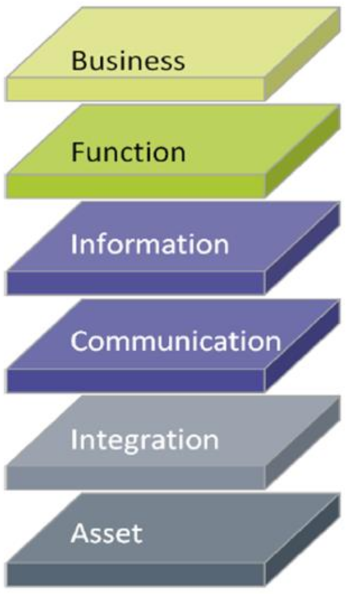


Figure 3. The IT Layers of RAMI 4.0.

3.2.2.2.1 Asset Layer

The asset layer spans the physical components of a system; physical things in the real world. E.g. production lines, manufacturing machinery, field devices, products and also the humans involved.

3.2.2.2.2 Integration Layer

The mapping from the physical world to the digital is performed by the Integration layer, which performs provisioning of information on the assets in a form which can be processed by computer. This involves all digitization of assets, such as connected sensors and other field devices, but also Human Machine Interfaces (HMI).

3.2.2.2.3 Communication Layer

The Communication Layer performs transmission of data and files. It standardizes the communication from the Integration Layer, providing uniform data formats, protocols and interfaces in the direction of the Information Layer. It also provisions the services for controlling the Integration Layer.

3.2.2.2.4 Information Layer

In the Information Layer, data and events are processed, integrated and persisted. This layer ensures the integrity of data, performs message translation and annotation and manages data persistence. It provides the service interfaces to access structured data from the Functional Layer and also applies event rules and transformation of event to the models and formats used in that layer. This is the run-time environment for Complex Event Processing (CEP), data APIs and data persistence mechanisms.

3.2.2.2.5 Function Layer

The Function Layer is the primary location of rules and decision-making logic and contains the formal descriptions of functions and service models. It is the run time environment for applications and services that support the business processes.

3.2.2.2.6 Business Layer

The services provided by the Functional Layer are orchestrated by the Business Layer. It maps the services to the business (domain) model and the business process models. It also models the business rules, legal and regulatory constraints of the system. The Business Layers receives events that advance, link and integrate the business processes.

3.2.2.2.7 Hierarchy Levels

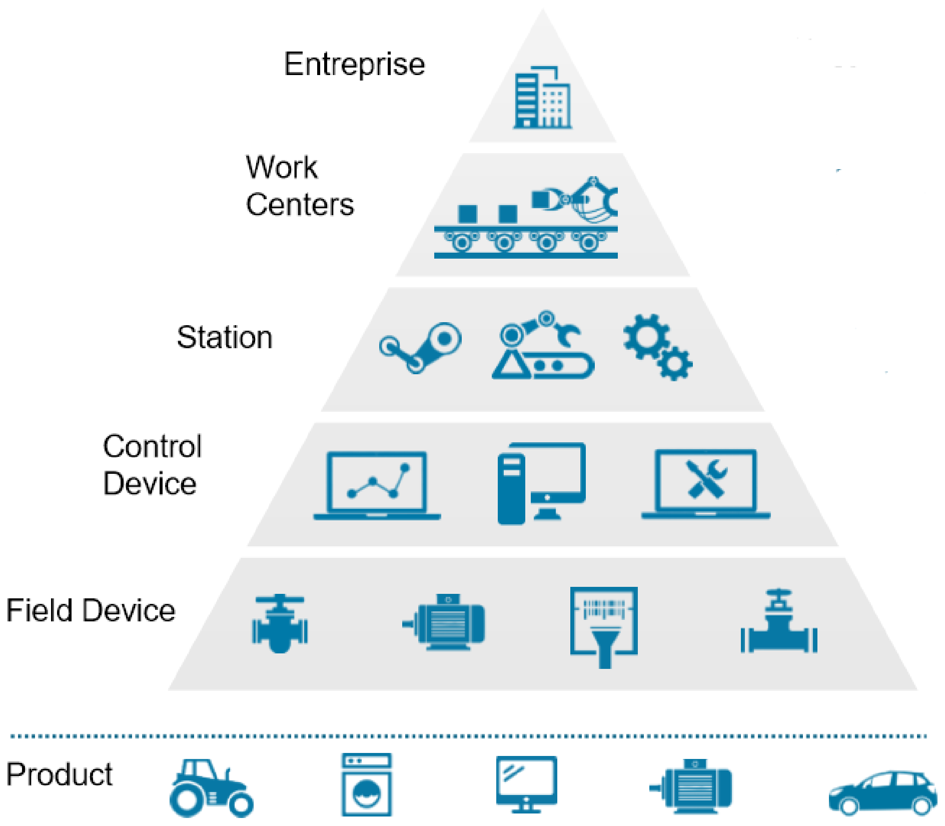


Figure 4. Hierarchy Levels of RAMI 4.0 (Status Report Reference Architecture Model Industrie 4.0 (RAMI4.0), 2015).

The right horizontal axis represents a hierarchy of different functionalities within factories or facilities. The ones shown in the pyramid in Figure 4, from "Field device" to "Enterprise" are derived from the IEC 62264 (IEC62264, 2013) international standards series for enterprise IT and control systems. The standard originated by modelling "wired" connections between functions performed by hardware in the factory, but today the functions are implemented in software. To represent the Industry 4.0 environment, the functionalities of IEC 62264 have been expanded to include workpieces, labelled “Product” (both the type and the instance, through the entire lifecycle), and the connection to the Internet of Things and Services, labelled “Connected World”. The "Connected World" involves the manufacturing ecosystem: groups of factories, collaborations with external engineering firms, component suppliers and customers.

3.2.2.3 Life Cycle and Value Stream

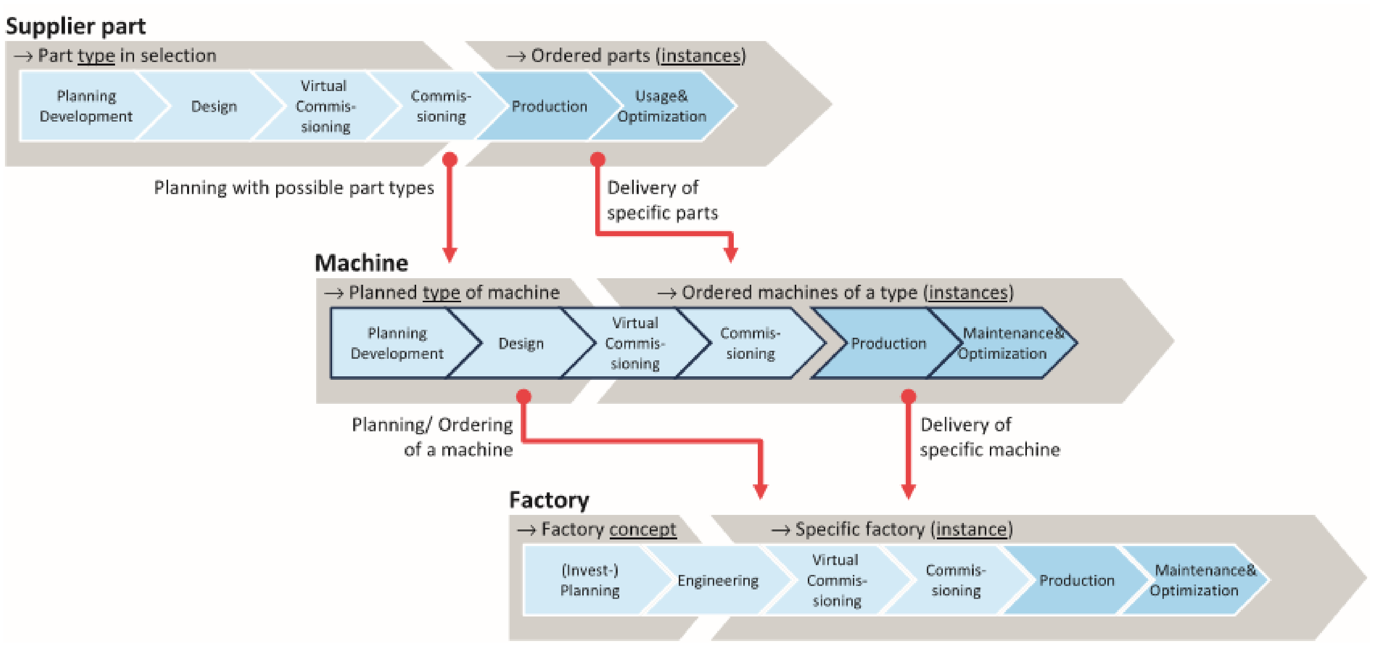


Figure 5. Type and instance lifecycles in RAMI 4.0 (Status Report Reference Architecture Model Industrie 4.0 (RAMI4.0), 2015).

The left horizontal axis in RAMI 4.0 represents the life cycle of facilities and products, based on the IEC 62890 (IEC, 2013). Distinction is made between types and instances; design and prototyping involve types, and each actual product being manufactured is an instance of this type.

As illustrated by Figure 5, this life cycle and value stream does not only cover the planning, design, production and maintenance of parts and products, but also types and instances of production equipment and factories.

3.2.2.4 Industrie 4.0 Component Administrative shell

An I4.0 component is the digitization of assets in the manufacturing process: it can be a factory, a production system, an individual station, or an assembly inside a machine. It consists of one or more assets and an administrative shell. The administrative shell is the virtual representation of an asset. The manifest of the administration shell describes the data provided by the asset and the resource manager provides access to the data and functionality of the asset. The I4.0 component is located within the layers of RAMI 4.0, up to the Functional Layer. It can adopt various positions in the life cycle and value stream, and occupy various hierarchical levels.

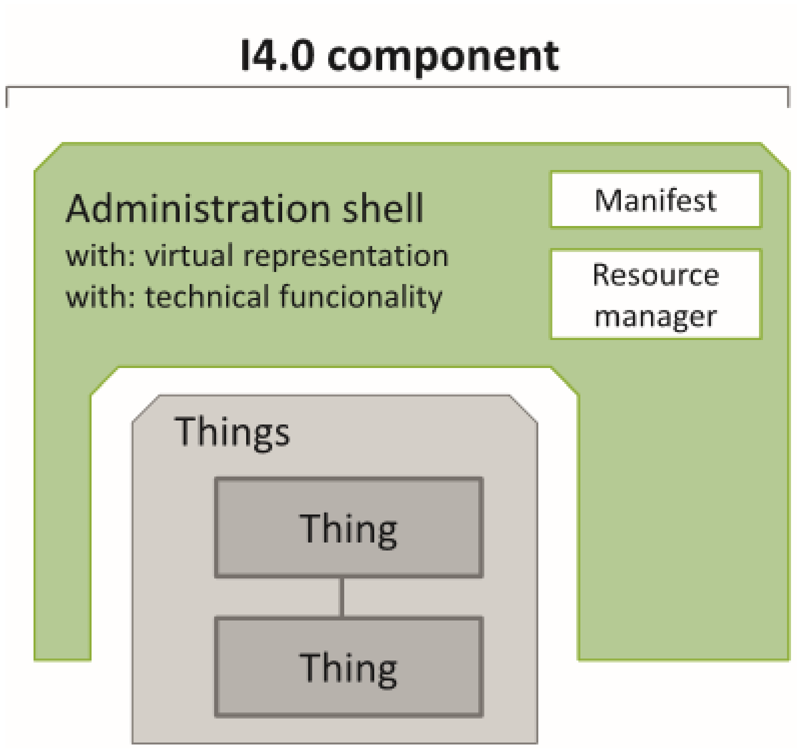


Figure 6. The I4.0 component (Status Report Reference Architecture Model Industrie 4.0 (RAMI4.0), 2015).

An asset may have several administration shells for different purposes and aspects of the manufacturing process. I4.0 components may be nested and accessed directly of as part of the implementation of the services of another I4.0 component. The administrative shell may be deployed in the run-time environment of the asset – if it possesses the necessary computational capabilities – or remotely, e.g. in a cloud environment.

4 Stakeholders, Concerns and Architecture Decisions

This section describes the stakeholders of the COMPOSITION system and their concerns. These concerns are expressed in different form and in different artefacts. Scenarios and requirements express some of these concerns. Although the system envisioned in the project specification is scoped to address the needs of all these, priorities must be made. Finally, architecture decisions pertaining to fundamental concerns are documented.

4.1 Stakeholders

The COMPOSITION system has several stakeholders, whose interests and concerns are expressed in the project governing documents. These may be categorized in groups, here we use the canonical ones from (Rozanski & Woods, 2012). The three key stakeholder groups for COMPOSTION have been identified as developers and maintainers (grouped together since these are basically the same in this case), acquirers, and users.

Acquirers are the European commission in H2020 framework, whose goals and concerns are stated in the project specification (COMPOSITION, 2016) and the technical and strategic objectives therein. These describe the main goals of the system, some software artefacts that will be delivered, and the need for collaboration with other projects, and re-use of results, in the industrial IoT and factory of the future programmes.

The developer stakeholder group consist of the technical partners in the project, commercial- and research-oriented. The concern of commercial partners is to produce exploitable results that can be sold as products or services, and produce innovations that can provide a competitive advantage in their respective market. Research organizations need to produce significant contributions to their respective field and build platforms and knowledge for further research. The concerns of these stakeholders are captured in the innovation and exploitation documents, the DOA and to some extent in the requirements.

The user stakeholder group are the pilot partners and future users of the system, whose concerns are mainly expressed in the scenarios, use cases (D2.1 "Industrial Use Cases for an Integrated Information Management System") and requirements (D2.2 "Initial requirements specification"). These capture the needs of the manufacturing industry and the priorities of the pilot partners.

4.2 Requirements

In a process parallel to the scenario development, described in report D2.2 "Initial requirements specification", several user requirements have been elicited. These have been entered into the project management system (Atlassian JIRA) and complemented by additional non-functional and operational requirements added by the developer stakeholders. In this initial requirements phase, 105 requirements were gathered, quality checked and improved.

The development efforts will be guided by the tasks in the project management system directly connected to the requirements. The current list of requirements will be revised in future design iterations and the results reported in D2.5 "Lessons Learned and updated requirements report I" and D2.6 "Lessons Learned and updated requirements report II".

4.3 Scenarios

Scenario workshops with mainly the user stakeholder group and some participants from the developer stakeholder group have been conducted to evaluate how COMPOSITION could optimise processes for manufacturing, logistics and supply chain collaboration within the scope of the pilots defined by Technical Objective 3.1. This resulted in nine (functional) scenarios describing application areas of the COMPOSITION system. These were detailed in 16 use cases for the pilots that capture user stakeholder concerns, presented in D2.1 "Industrial Use Cases for an Integrated Information Management System". These scenarios were prioritized by the developer and user stakeholders and two scenarios, one intra-factory (value chain) and one inter-factory (supply chain) of these were selected as the first scenarios to be implemented, together with corresponding use cases. These were selected on the basis of importance to the user stakeholders and the developer partner's estimate of the coverage they provide of the systems intended functionality.

The scenarios the workshop prioritized to work through to ensure that the architecture provides coverage of the base functional requirements are:

INTRA-2 Predictive Maintenance

Replacing a machine part on a schedule, possibly before it is needed, causes unnecessary costs. Conversely, a failing machine means that products being processed may have to be discarded and that the manufacturing process will have to be stopped. The COMPOSITION system will predict failures of critical components based on collected real-time information from the production equipment, e.g. levels and temperature of solvents, noise levels or vibration of machines.

INTRA-2: UC-BSL-2 Predictive Maintenance

The predictive maintenance scenario is instantiated by this use case in which the system will monitor and analyse fan parameters such as noise and current change. Users will be able to access fan performance data and will be notified of impending fan failure. This scenario involves the system components from sensors through intra-factory integration, to big data analytics, deep learning and human-machine interfaces.

INTER-1 Scrap Metal Management

The system will automatically collect and compare data of scrap metal levels per bin in the factory. When the scrap metal reaches a threshold, that indicates that the bin is or will be full soon, an event is generated by the system. The system may learn which level is accurate for the threshold. The system may use price forecasting to select how and when to negotiate for the price of scrap metal. The event initiates a request for pick-up through the factory COMPOSITION agent system. Other agents representing scrap metal recycling companies in the marketplace will negotiate for the contract. Selected and rejected companies will be notified automatically about the final decision. A pick-up date and time is agreed with the selected scrap metal recycling company and drivers are notified. Human control gates may be implemented at points in the process.

INTER-1: UC-KLE-4 Scrap metal collection process

This use case deals with optimizing the scrap metal collection process. The waste management companies want to optimize the transportation for the scrap metal collection. The sensors monitoring the scrap metal are continually feeding information in the level, allowing waste management companies to plan. Negotiation between agents on pick-up arrangements are preformed and all parties are notified of the outcome.

The above scenarios will be the primary instruments to use for analysing the functional suitability of the design. The functional scenarios will be complemented with system quality scenarios describing the system reactions and responses to changes in its environment, e.g. workload fluctuations, security threats, operational and deployment modifications.

4.4 Concerns and Architectural Decisions

4.4.1 Concerns

The goals of the COMPOSITION system are stated in the strategic and technical objectives in the project specification (COMPOSITION, 2016) and can be found summarized in the table below. These are necessary objectives stated by the acquirers of the system. There is an emphasis on interoperability, integration and analysis of information from heterogenous sources, dynamic adaptation to market requirements and innovativeness.

- Strategic Objective 1: Create a digital automation framework (the COMPOSITION IIMS) that optimizes the manufacturing processes by exploiting existing data, knowledge and tools to increase productivity and dynamically adapt to changing market requirements.
    - Technical Objective 1.1: Innovate and extend the FI-WARE and FITMAN catalogues of Generic Enablers with an innovative CPS-aware library of open, standard connectors specialised for real-time architectures for interoperability in manufacturing to ease the integration and coupling of data, information and knowledge from existing, heterogeneous, sources in the factory.
    - Technical Objective 1.2: Research and develop innovative, multi-level, cross-domain analytics detecting complex patterns in manufacturing big data sets, and implementing a continuous deep learning toolkit for re-adaptation and adjustments of operational metrics, in real time.
    - Technical Objective 1.3: Develop a set of modelling and simulation tools including a Decision Support System (DSS) to help users build the digital models of processes and products and to forecast impacts of reconfigurations of the production process.
- Strategic Objective 2: Enable the *COMPOSITION ecosystem* by designing and implementing a technical operating system supporting connected and interoperable factories, with their stakeholders and, by optimising manufacturing and logistics processes through new innovative services and practices.
    - Technical Objective 2.1: Design and implement *a Log Oriented Architecture*, based on blockchain technology, ensuring the trusted, secure and automated exchange of supply chain data among all authorized stakeholders, to connect factories and support interoperability and product traceability along the supply chain.
    - Technical Objective 2.2: Provide *end-to-end security* from factory floor to cloud services encompassing major mechanisms in a seamless and fully integrated manner including authentication and access control, transport security, as well as system security, while maintaining suitable levels of IPR and knowledge protection.
    - Technical Objective 2.3: Develop an *interoperable agent-based marketplace*, where each party is represented by one or more agents, endowed with sufficient autonomy to set up exchanges and to enable new economic collaboration models.
- Strategic Objective 3: *Demonstrate and validate reference implementations* of the full COMPOSITION ecosystem in real value and supply chains to foster take-up and re-use at European level.
    - Technical Objective 3.1: Implement, demonstrate and validate the COMPOSITION operating system in *two multi-sided pilots*.
    - Technical Objective 3.2: *Collaborate and integrate* successful concepts and technologies with other relevant initiatives such as Industrial Data Space and FITMAN.

Figure 7. The strategical and technical objectives of COMPOSITION.

The developer stakeholders, i.e. the technical partners, are interested in the exploitability of COMPOSITION results. The system should be compatible with the existing products and stakeholders should be able to supply components and services complementing and extending the system on the COMPOSITION aftermarket. Developer stakeholders use different programming languages and make use of existing software frameworks in the project.

Developer and maintainer stakeholders also have an interest of offering their (software) services using the COMPOSITION system (D2.1).

This creates requirements for the extensibility and evolvability qualities of the system, and the need of a set of standards and interfaces that companies developing a component extending the COMPOSITION system can adhere to. Components should not use programming language or platform specific inter component communication. Opens standards should be used and special consideration should be taken to the ones already supported by the development stakeholder products.

The formats, protocols and interfaces (“open, standard connectors”) should be designed to enable both use of and extension of FI-WARE and FITMAN Generic Enablers as well as the integration of concepts and technologies from other initiatives in Industrial IoT.

The context in which the COMPOSITION system will be deployed is expected to be heterogenous, with different factories using different infrastructure. The architecture design will have to take this into account and allow for flexibility in deployment of components and adaptation to existing infrastructure.

COMPOSITION services and applications should be possible to deploy independently of each other under different licences to accommodate for the interests of the commercial consortium partners. Licensing must allow for commercial usage of individual components or the entire system. Incorporating or applying open source licensing affecting the possibility of commercial exploitation, such as GPL, is explicitly forbidden (COMPOSITION, 2016).

Security should be seamlessly integrated in the entire system and allow for integration of components from external sources into the COMPOSITION platform. The use of open standards is thus a requirement from the security perspective as well.

4.4.2 Architectural decisions

In architecture workshops and discussions, with input from the design of components, a number of system-level decisions for architectural mechanisms (Kruchten, 2004) have been made. Below is a summary of major architectural decisions at the time of writing.

The COMPOSITION project specification (COMPOSITION, 2016) provides a basic functional decomposition of the system. This considers the objectives of the acquirers, the frameworks and components brought to the project by developer stakeholders and provides a division of development work in alignment with the project plan. The decision was made to build the system bottom up starting from the components given by the breakdown in the project specification and revise this as needed.

Existing components form developer partners will be integrated in the system. COMPOSITION will re-use earlier results and frameworks familiar to the partners, e.g. the LinkSmart Middleware and the Symphony BMS. This provides a code base to build on and provides compatibility with existing product lines, enhancing exploitability of the results for the partners. The system should also allow for the use of other frameworks providing similar functionality.

4.4.2.1 **Communication mechanism**

Given the emphasis of extensibility, interoperability, analysis of heterogenous data and loose coupling in the COMPOSITION system, the general communication mechanism for the system will be data-centric and messaging-based, where factory data is published and interested components (performing e.g. analytical or supervisory functions) subscribe to this data without direct addressing between components. This will be built using standard message broker components with extensions for security, multi-protocol and multi-format support.

The AMQP protocol will be used for component communication and message routing. It is a very flexible protocol that may be configured for different message routing schemes and emulation of other protocols such as MQTT, STOMP, XMPP or the Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)[4]. The project has selected RabbitMQ as the implementation of this mechanism. RabbitMQ is open source software, extensible and has support for multiple platforms. As MQTT may be transparently used by clients on top of an AMQP broker architecture, this protocol will be used for the components that already implement MQTT support.

The focus of COMPOSITION is on functionality that requires “human scale” response time, e.g. visualization, simulation, forecasting rather than real-time device control in the sub-millisecond range. It is therefore not required to build on very fast device-device integration protocols (e.g. DDS) as a communication layer, but rather include such protocols as a possible asset layer should it be needed. Interoperability and integration of heterogenous data sources for analysis, optimization and decision support are the primary concerns for the communication mechanism design.

The IoT interoperability functionality will build on LinkSmart, which uses MQTT as its message-based communication mechanism. The developer stakeholders have extensive experience with the LinkSmart platform, which has been used in several large IoT projects previously, and using this will be effective in developing the core interoperability functionality of the project. However, alternatives were considered. Standards such as the Foundation Open Platform Communications-Unified Architecture (OPC-UA)[5] and the Data Distribution Service (DDS)[6] are already used in industrial applications. In terms of architectures for industrial applications, the proposed solution has more similarities with the message-centric design of DDS than the more device-centric model of OPC-UA. However, the LinkSmart platform allows both for directly addressing devices and requesting data, and subscribing to data by type without knowledge about the

[4] https://www.rabbitmq.com/community-plugins.html
[5] https://opcfoundation.org/about/opc-technologies/opc-ua/
[6] http://www.omg.org/spec/DDS/

hardware involved. Providing support for existing de-facto standards such as DDS or UPC-UA as asset layers, should this be required in the future, is possible within the LinkSmart platform.

The external interfaces of components in the COMPOSITION system will use RESTful HTTP interfaces for request-response communication. For message-based communication, the MQTT and AMQP protocol will be used.

The OGC SensorThings API Data Model will be used for system-generated information passed between components.

4.4.2.2 **Deployment and System Management mechanism**

The use of heterogenous platforms and frameworks as well as existing products from several development stakeholders within the COMPOSITION system will result in different build chains and platforms to be used. To provide both a consistent deployment and system management mechanism, all components will be made available as pre-configured, container-based instances. As described in section 5.5.1, have chosen Docker as the container implementation and will use Shipyard or Portainer as a management tool.

4.4.2.3 **Security mechanism**

For authentication and authorization of both system components and users, Keycloak will be used. It will be integrated into the message broker, thus allowing all components to use the security system in a uniform manner. Blockchain functionality will also be integrated in the broker functionality, providing distributed trust for any message sent through this mechanism. After evaluation and tests, Multichain has been selected as the blockchain implementation in COMPOSITION. Transport Layer Security will be used as part of the communication security protocols.

4.4.2.4 **Data persistence mechanism**

Component-specific configuration data and caching is handled inside the components. Due to the bottom-up design approach and that analysis and pattern detection is performed on data streams rather than static data, a common data persistence mechanism for system-generated data has not been prioritized in the initial work. However, there are components that will rely on querying structured data generated by the system and the design choice for these is an OGC SensorThings API compliant data store. There are several implementations available. This is discussed in section 5.4.2.

4.4.2.5 **Metadata mechanism**

The Digital Factory Model (DFM) (COMPOSITION, 2016), described in section 5.4.1.2, is the system source of information on classes and instances in the factory. It contains information on production lines, sensors, the id of a sensor, what phenomenon it reports data for, format and unit of measurement.

Other parts of the system, such as the middleware, the message broker and the human computer interfaces, will need this information when searching for or subscribing to messages containing data on specific objects or types of objects. E.g., the intra-factory interoperability layer (section 0) will publish information coming from a sensor. This may be published containing metadata in-band, e.g. containing information on the unit of measurement or associated production line, or the metadata may be located out-of band. In the latter case, components subscribing to data for a production line will have to first locate the relevant data sources using the DFM and then subscribe to data based in the identifiers of these data sources.

Whether metadata should be communicated in- or out-of-band has not yet been decided. This has an impact on the design of topic structures for the message broker and the information flow when components request new types of information from the system. In the case of out-of-band metadata, bindings for the system components could be set up at deployment time using a designated system configuration component.

5 Architectural views

5.1 Overview

The strategic objectives of COMPOSITION state two main deliverables of the project: a digital automation framework to integrate data along the value chain inside the factory, and a largely automatic ecosystem to interconnect different stakeholders in the supply chain.

COMPOSITION has two main goals: The first goal is to integrate data along the value chain inside a factory into one integrated information management system (IIMS) combining physical world, simulation, planning and forecasting data to enhance re-configurability, scalability and optimisation of resources and processes inside the factory to optimise manufacturing and logistics processes.

The second goal is to create a (semi-)automatic ecosystem, which extends the local IIMS concept to a holistic and collaborative system incorporating and interlinking both the supply and value chains. This should be able to dynamically adapt to changing market requirements.

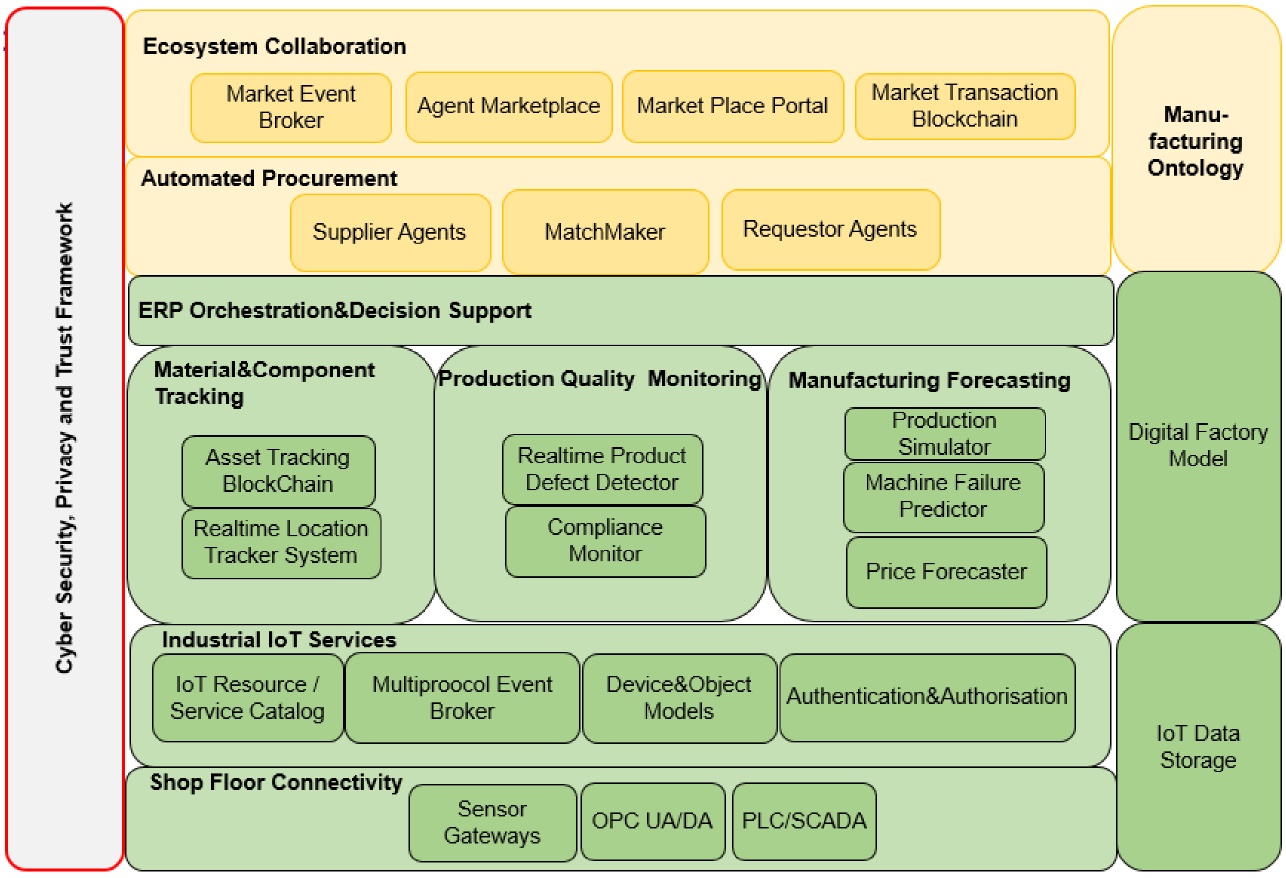

Cyber Security, Privacy and Trust Framework
Ecosystem Collaboration
Market Event Broker
Agent Marketplace
Market Place Portal
Market Transaction Blockchain
Manu-facturing Ontology
Automated Procurement
Supplier Agents
MatchMaker
Requestor Agents
ERP Orchestration&Decision Support
Material&Component Tracking
Asset Tracking BlockChain
Realtime Location Tracker System
Production Quality Monitoring
Realtime Product Defect Detector
Compliance Monitor
Manufacturing Forecasting
Production Simulator
Machine Failure Predictor
Price Forecaster
Digital Factory Model
Industrial IoT Services
IoT Resource / Service Catalog
Multiproocol Event Broker
Device&Object Models
Authentication&Authorisation
IoT Data Storage
Shop Floor Connectivity
Sensor Gateways
OPC UA/DA
PLC/SCADA

Figure 8. Composition conceptual architecture.

The digital automations framework combines the data sources in the factory value chain, data from the production lines, ERP systems, forecasting, simulation and analytics data to form an integrated information management system (the COMPOSITION IIMS). At the lowest level the *Shop Floor Connectivity* provides access to devices, machines, equipment and sensors installed in the factory. The *Industrial IoT Services* layer creates an Internet of Things environment and enables standardised communication, discovery, data exchange and service innovation mechanisms.

The *Industrial IoT Services* feeds a number of business services with collected IoT and other production data:

Material&Component Tracking

A *Realtime Location Tracker System* keeps track of where products and other valuable components are on the shop floor while an *Asset Tracking Blockchain* is used to log transfer and movements of components in the manufacturing chain.

Product Quality Monitoring

The *Compliance Monitor* is responsible for checking that a product is manufactured and handled according to relevant regulations. The *Realtime Product Defect Detector* uses advanced data fusion and big data analytics to detect any deficiencies in a product.

Manufacturing Forecasting

The *Machine Failure Predicto*r uses deep learning and advanced big data analytics to predict failures of machine and needs of maintenance. The *Price Forecaster* uses trained artificial neural networks to forecast the price of products and components. A Production Simulation and Forecasting Engine allows shop managers to simulate effects of re-configuration of processes inside the factory to optimise manufacturing and logistics processes.

Automated Procurement

One of the main innovations of Composition is the use of agent technologies to automate the procurement and negotiation process. Autonomous *Supplier* or *Requestor Agents* that negotiate and reach agreements with other stakeholders. A *Matchmaker* helps in find and matching best available offers with request.

Ecosystem Collaboration Framework

A virtual marketplace is envisioned where each party is represented by one or more semi-autonomous agents. To enable the COMPOSITION ecosystem an infrastructure for an *Agent Marketplace* is developed to support dynamic and automated connections between stakeholders in the supply chain, making manufacturers, suppliers and logistics interoperable and optimizable. The *Market Event Broker* propagates message between different actors in the marketplace. Trust is achieved by the use of an *Audit Log Blockchain* to maintain an immutable ledger of agreements and transactions.

Meta Data and Storage

Finally, *IoT Storage* allows for logging and storing of historical data from the shop floor. The *Digital Factory Mode*l is a high-level representation of the shop floor, stations, cells, productions lines and all the IoT sensors. The *Manufacturing Ontology* contains semantics about the market place.

Cyber Security, Privacy and Trust Framework

The Security Framework managing Cyber Security, Privacy and Trust, is a cross-cutting concern spanning the entire platform, providing end-to-end security by means of standard and widely used protocols for identification and distributed trust (e.g. OpenID and the Bitcoin blockchain protocol).

5.2 Context View

The Context View describes the system boundaries and interactions with its environment: how the system is connected to actors in the marketplace and other systems, e.g. existing factory infrastructure.

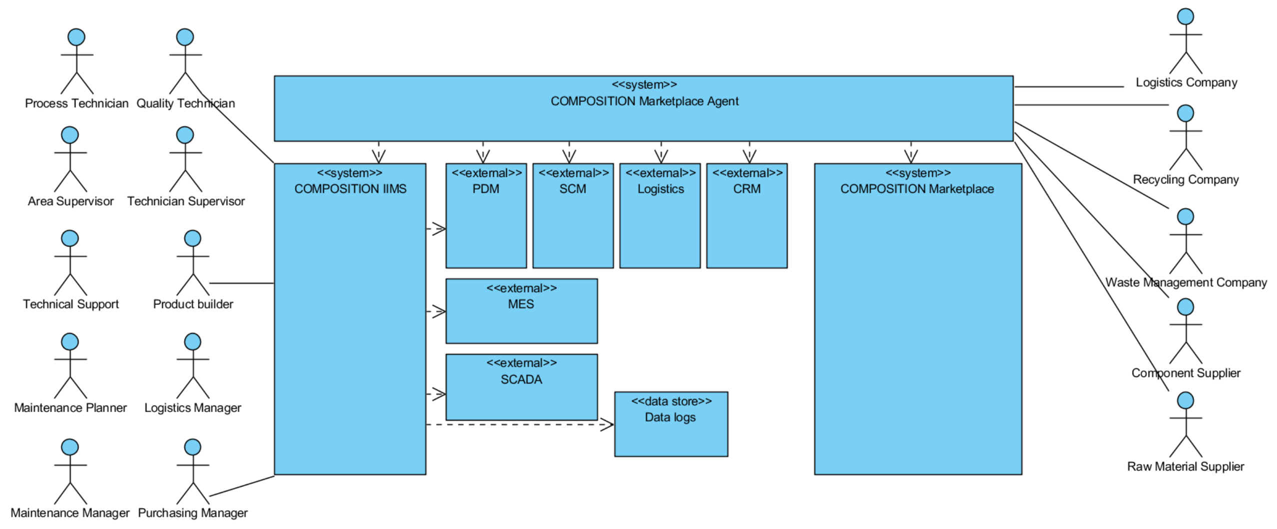


Figure 9. The COMPOSITION system context view.

The value chain IIMS interacts with the actors in the value chain and external systems in the factory, e.g. Product Data Management (PDM), Manufacturing Execution System (MES) and Supervisory Control And Data Acquisition (SCADA). Some analytics components in COMPOSITION use external data logs as input. The COMPOSITION Marketplace Agent in the intermediary between the factory IIMS and the COMPOSITION Marketplace. The agent uses information from external systems for Product Data Management (PDM), Supply Chain Management (SCM), Logistics or Customer Relationship Management (CRM) and/or data from COMPOSITION to initiate and guide the actions it takes in the Marketplace.

5.2.1 Concepts

The COMPOSITION virtual marketplace is envisioned as the “virtual” place where the “digital” representatives (agents) of involved stakeholders meet, exchange information, transfer operations data and negotiate to dynamically form supply chains. Such chains may either involve physical goods being exchanged (i.e., supply materials) or immaterial assets such as services, consultancy, etc. Exchanges on the marketplace typically happen at a relatively high frequency, can involve potentially high numbers of stakeholders and do not require direct human intervention. Nevertheless, humans are part of the loop being involved by their respective agents every time their judgment or approval is needed, e.g., to confirm a pre-negotiated deal.

This autonomous, but supervised behaviour allows for increasing the flexibility and speed at which supply chains are formed, executed and destroyed, with potentials to improve the current supply-level processes in many domains within the Industry 4.0 landscape.

The overall logic architecture of a COMPOSITION marketplace is depicted in Figure 10, which mainly highlights the relation between the IIMS on the stakeholders’ side and the respective agents on the marketplace side. According to the original project specification (COMPOSITION, 2016), “every factory involved in the COMPOSITION ecosystem is able to generate supply-chain formation primitives (supply needs) described in a high-level, machine understandable format. Such requests are handed over to the factory representatives in the COMPOSITION market place (the COMPOSITION Agents), triggering autonomous exchange of information, and negotiation with peer agents. Such exchanges are regulated by negotiation parameters, security and trust rules and by reputation mechanisms, thus ensuring a reliable, and somewhat predictable dynamic formation of supply chains. Each representative agent processes, filters and dynamically masks exchanged information (e.g., request for services / material) according to agent-specific policies. Resulting information is then be matched against goals set-up at the single factory level (intra-factory side), possibly mediated by a human user. Throughout the whole negotiation process, security, data protection and obfuscation will be applied to ensure that exchanges between factory representative agents are protected.”

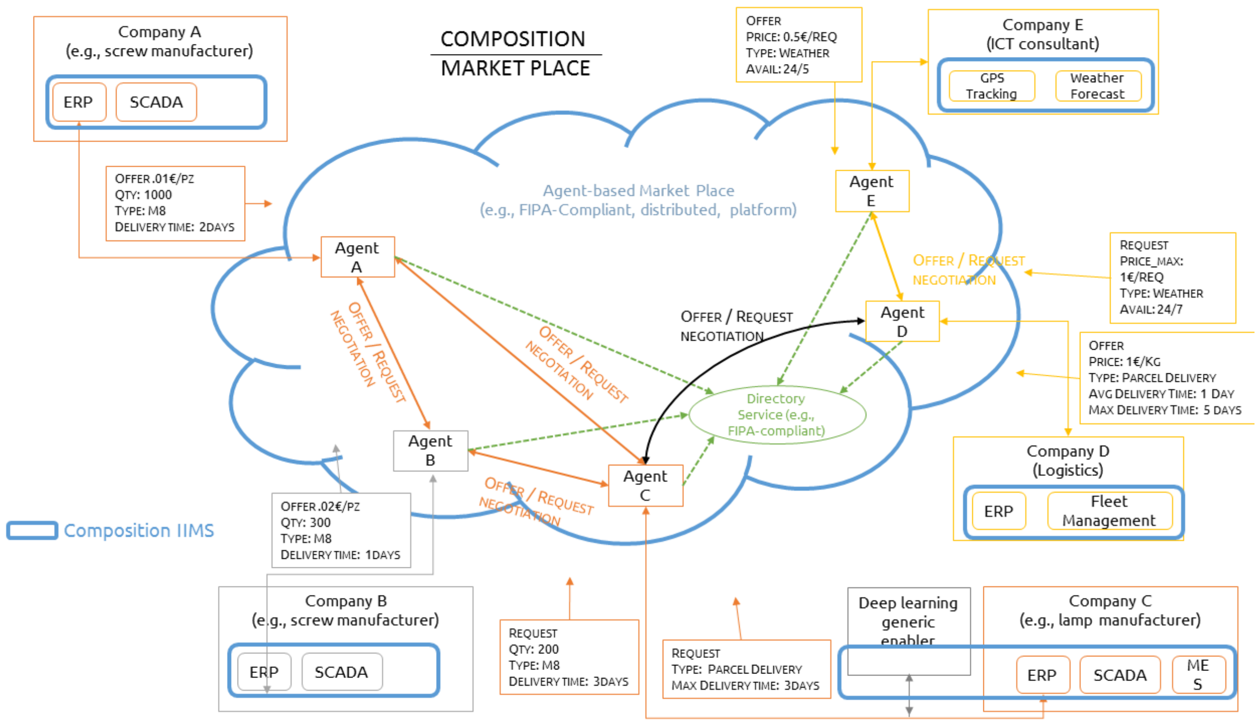


Figure 10. The logic architecture of a COMPOSITION marketplace.

Starting from the definition in (COMPOSITION, 2016), integrated with evidences resulting from the initial requirements gathering process (D2.1), a more formal definition of marketplace has been identified, together with some corollary information, better detailed in the following.

5.2.1.1 **Definitions**

The COMPOSITION marketplace can be seen as a particular variation of a Multi-Agent System (MAS). MAS have been widely investigated in research, and their application domains range from Distributed Constraints Optimization (DCO) problems to coordination and delegation of computational tasks. While the adoption of agent systems in automatic negotiation, i.e., for DCO problems, is not new, as witnessed by the huge amount of literature available, application of such techniques in real-industrial environments, in a fully decentralized set-up still presents some research challenge and offers possibilities for advancing the state of the art. As part of the architecture specification process documented in this deliverable, activities on the agent marketplace mainly lead to a fully de-centralized definition of MAS, including the de-materialization of traditional agent containers into a much more light set of collaborating software (agents) sharing a common communication infrastructure and common agency services (i.e., white and yellow pages).

According to the COMPOSITION approach, agent containers are defined as follows.

An agent container is a set of intelligent agents interacting through the same, shared broker (can be a cluster) and referring to shared platform services such as the Directory Facilitator[7] *and the Agent Management Service."*

Differently from approaches, in which the agent container is seen as a central runtime environment where all the agents belonging to a certain system live, in COMPOSITION agents are designed to live at the stakeholder premises (or in its IT infrastructure). This permits on one hand, to improve trustworthiness of agents, and acceptance, as no real code access is possible for entities other than the agent owner itself. On the other hand, it permits to remove typical constraints of traditional MAS systems, e.g., (a) the single point of failure represented by the Agent Container, (b) the scalability issues, (c) the techniques for enabling container-to-container communication, (d) the performance issues related to central deployment of computationally intensive agents. Moreover, the fully distributed approach proposed in COMPOSITION, reduces as much as possible the typical overhead of intercommunicating agent containers. Agency services

[7] In COMPOSITION a more advanced version of such an agent, namely the MatchMaker, operating based on ontology models is adopted.

are in fact shared naturally among distributed agents, thus removing the typical issues of duplication among containers and the related synchronization and/or delegation problems of activities needed for effectively supporting agent search and/or directory services.

While being decentralized by design, the COMPOSITION marketplace definition is centred on a so-called communication broker, which might be identified as single-point-of-failure for the architecture. However, several studies, and results in literature, show that design solutions can be adopted, based on clustered deployment, which can ensure high resilience to failures for these kind of broker-centric messaging infrastructures (see Section 6.2).

In COMPOSITION, an agent-based marketplace is simply defined as an "agent container".

Despite this simple, technical, definition, several variations of the marketplace concept are introduced including the distinction between open and closed marketplaces as well as the introduction of temporary association of agents, or "virtual marketplaces".

COMPOSITION foresees the possibility to have more than one market place running at the same time, serving different communities. However, according to the project specification (COMPOSITION, 2016), the marketplace must support the discovery of stakeholders not part of established supply chains. Assuming that in first instance a single market place corresponds to an extended supply chain, the concept of a so-called "open marketplace" can be introduced.

A COMPOSITION Open Marketplace is "a COMPOSITION Marketplace open to any stakeholder having valid COMPOSITION credentials".

All players of the COMPOSITION ecosystem shall have a representative in the Open Marketplace. However, some stakeholder might decide to invite other stakeholders to participate in a Closed Marketplace, e.g., to protect/isolate certain supply chains. Such an invitation is managed through suitable agent interaction (i.e., messages) and/or through a dedicated marketplace portal. Closed Marketplaces are structurally equivalent to open marketplaces. The main difference with respect to an open marketplace is that a closed marketplace is a separated marketplace with its own infrastructure, e.g., AMS, DF and communication broker. Closed Marketplaces typically run on the premises of the marketplace owner and are subject to additional join and/or participation policies defined by the marketplace owner. The closed market place operations and exchanges are "isolated" from the open marketplace. A closed marketplace is defined as follows.

"*A COMPOSITION Closed Marketplace is a COMPOSITION Marketplace owned by one stakeholder and typically offered to a trusted subset of other COMPOSITION stakeholders. The Closed Marketplace can be public or private. The former will accept join requests by agents living in the Open Marketplace while the latter will accept agents only by invitation. A Closed Marketplace is physically separated by the Open Marketplace and has typically a separate infrastructure including the broker, AMS, DF, etc."*

In case collaboration within agents shall occur on a temporary basis, *Virtual Marketplaces*, are supported, e.g., through grouping mechanisms similar to multicast communication. In particular,

"*A Virtual Marketplace, or group, is a "multicast" group of agents interacting with each other in the context of a negotiation. The group can be persistent over negotiations or can just be defined for a single negotiation exchange. A Virtual Marketplace lives in, and exploits the infrastructure of an Open Marketplace*."

While these technical innovations are still subject of active research, and will certainly be refined during the project lifespan, they already open new exploitation possibilities for the COMPOSITION marketplace concept, and contribute to lower the technology acceptance level for industrial stakeholders.

More specifically:

- The *Distributed Marketplace* derives from strict interactions with the composition industrial partners and provides means to ensure trust on the system, as the involved stakeholders retain full control on their software representatives on the marketplace. Moreover, it opens possibilities for new businesses in the supply chain, e.g., the marketplace infrastructure provider, which can be independent from involved stakeholders and might require a fee for using provided services. Such services include basic connectivity, agency services and the possibility for stakeholders to define and run their own *Closed Marketplaces*.

- The *Closed Marketplace* allows the marketplace owner, typically the “central actor” of a supply chain, to keep control on involved partners and to ensure a certain degree of reliability of actors involved in the chain(s). This concept provides a tuneable tool to trade-off the need of marketplaces open to possibly new stakeholders (*Open Marketplace*) and the contrasting need of having trusted, certified suppliers able to guarantee proven quality in provided materials / services. This ability to tune the “openness” of a certain marketplace is a relevant factor for effective adoption of COMPOSITION, possibly opening access to very controlled supply chains, e.g., those subject to strict certification processes.
- While being central to the marketplace, supply chain formation and related activities (e.g., post-sell services) are not the only focus of the marketplace. Active advertisement and support to service / stakeholder search is a valuable asset, as witnessed by explicit requirements set by the project SME providing added value services, e.g., consultancy, integration and customization. The inclusion of such needs in the initial design of the COMPOSITION marketplace shall increase the overall exploitability of the project outcome, by widening the possible stakeholder base.

5.3 Functional View

5.3.1 High-level functional view

The above diagram describes the COMPOSITION system from business architecture functional view. Generic functional components like Complex Event Processing (CEP) and Deep Learning ANNs (Artificial Neural Networks) are used to implement business specific functionality, e.g. machine failure prediction.

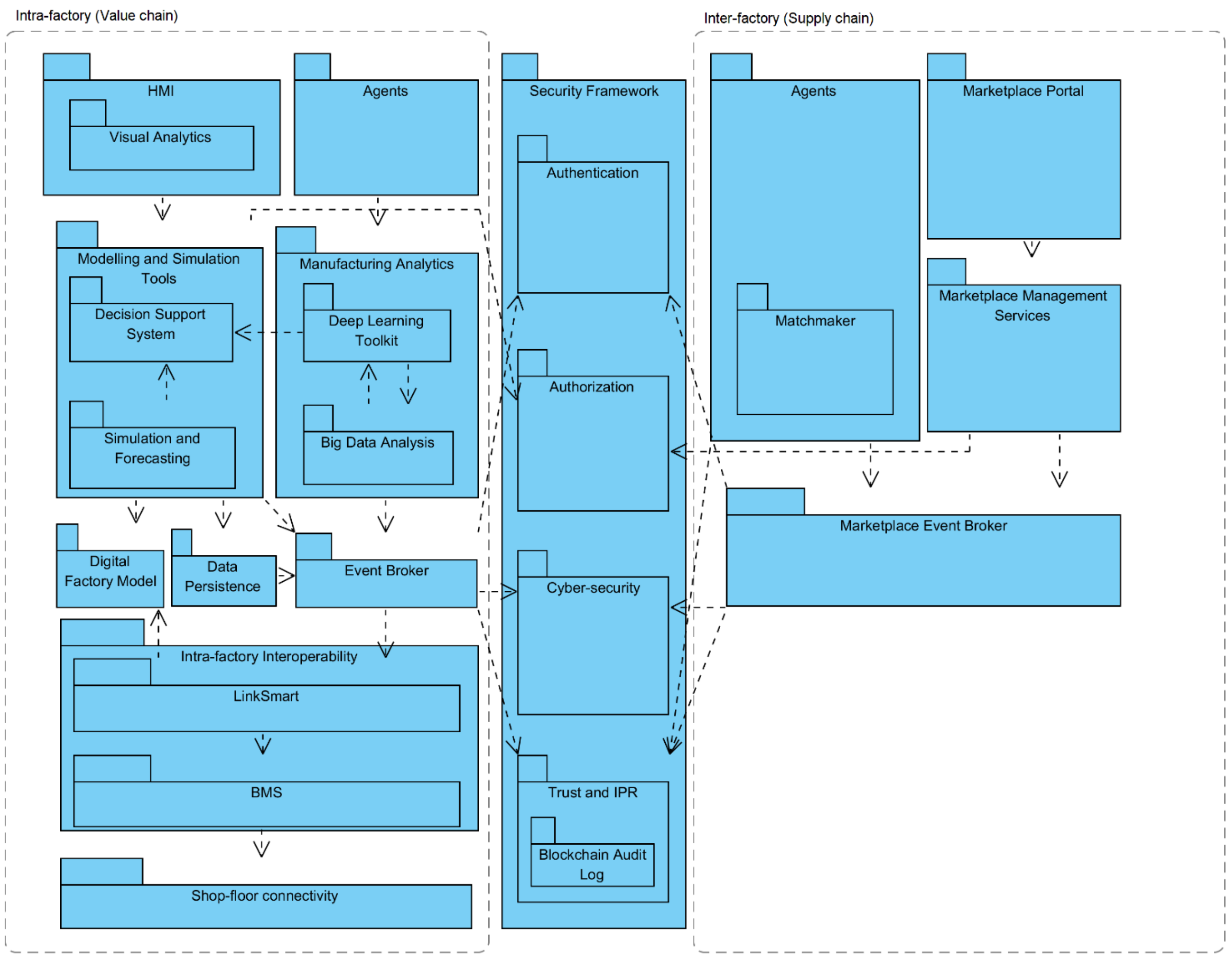


Figure 11. High-level functional view of COMPOSITION architecture.

The above diagram provides a high-level view of the components and their interactions. Detailed diagrams are available in the sections for the respective component. Components which have been identified at a late stage in the process do not yet have detailed descriptions, e.g. the various HMI components such as Visual Analytics.

The design decision has been taken to use broker-based communication, with adapters for request-response based communication in both the IIMS and the Marketplace. This makes it possible to secure communication using the broker and having this as the policy enforcement point for communication between components. It also allows for logical addressing of the components through the broker. As mentioned above, AMQP has been selected as the primary protocol to use. In the above diagram, the communication mechanism is regarded as infrastructure and is thus not modelled.

5.3.1.1 **RAMI 4.0**

A mapping of the initial COMPOSITION system components to RAMI 4.0 Layers can be seen in Figure 12. This diagram is expected to evolve further over the duration of the project.

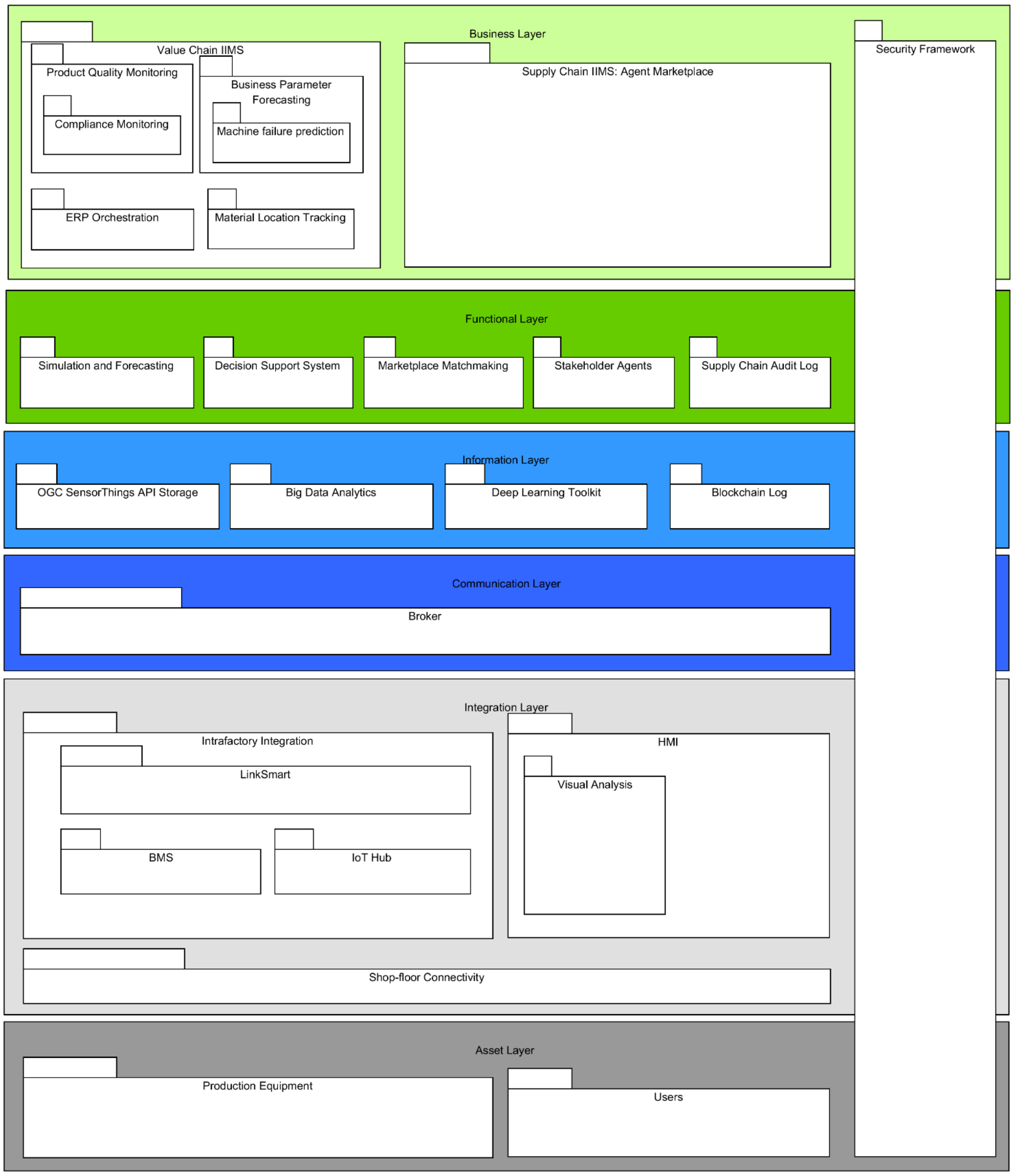


Figure 12. A mapping of COMPOSITION functional packages to the RAMI 4.0 Layers.

The COMPOSITION system scope and pilots cover the intra-factory functionality from "Field Device" to "Work Center" via the IIMS and has a special emphasis on the inter-factory ecosystem of the "Connected World", provided by the interoperable agent-based marketplace and the blockchain-based log-oriented architecture, providing secure and trusted exchange of supply chain data between independent parties.

Life cycles of both types and instances of products and machines is covered by COMPOSITION, where complex pattern detection, deep learning networks and simulation capabilities may be used both for operational management and continuous improvement of factory equipment and products.

The administrative shell can be implemented at various levels in the COMPOSITION system. The different possible implementation mechanisms of the Intrafactory Interoperability Layer create administrative shells for the connected assets (see section 5.3.3.2). More complex administrative shells for production lines are implemented inside the IIMS using other components such as the Big Data Analytics, Decision Support System or Simulation and Forecasting Tool. The I4.0 components will be layered on top of each other and more than one administrative shells may exist for the same asset or combination of assets. Further work will be performed to align the configuration of a COMPOSITION system instance with the concept of I4.0 components.

5.3.2 Market Event Broker and Real-time Multi-Protocol Event Broker

In COMPOSITION, there is a need to support the most common IoT Messaging Protocols to integrate data from multiple sources in the intra factory and support flexible component integration. There is also a need to be able to secure the messaging using the services provided by the COMPOSITION security framework. Furthermore, as an intermediary, decoupling system components, the Message Broker also provides the means to manage scalability in a consistent manner.

The COMPOSITION Message Broker will be the communication mechanism in both in the intra factory and in the COMPOSITION market place. Note that this will be two completely different instances but they will provide the same basic mechanism of communication and they are configured individually, i.e. the components will be the same but they will be used differently.
The Advanced Message Queuing Protocol (AMQP) is an application layer protocol for message-oriented middleware, provided as an open standard. AMQP is highly configurable and can emulate other protocols, e.g. MQTT.
The implementation mechanism chosen for the COMPOSITION Real Time Multi-Protocol Event Broker is based on RabbitMQ[8], an open source component supplied under the Mozilla Public License. RabbitMQ is an open source message broker that is in wide use[9] and has an extensible architecture. It implements the AMQP 0-9-1 protocol[10] and through adapters supports the most common messaging protocols, e.g. MQTT, STOMP and XMPP. Extensions and adapters can be written to support other messaging patterns, protocols and security management solutions.

The main messaging scenarios that the Real Time Multi-Protocol Event Broker will support are the following:

- **Simple queueing:** Where messages are queued between producer and consumer, see fig below, acting as a buffer.

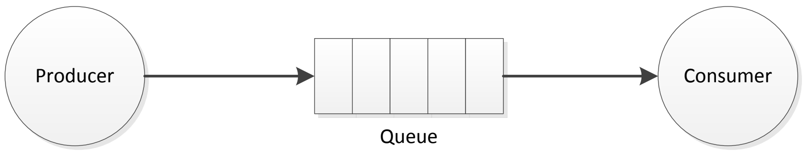


Figure 13. Simple queuing.

- **Publish/Subscribe:** A common pattern for message based architectures where a producer publishes messages typically with a topic pattern and consumers subscribe to different patterns.

[8] https://www.rabbitmq.com/
[9] At the time of writing 35.000 production deployments , https://www.rabbitmq.com/
[10] http://www.amqp.org/sites/amqp.org/files/amqp0-9-1.zip

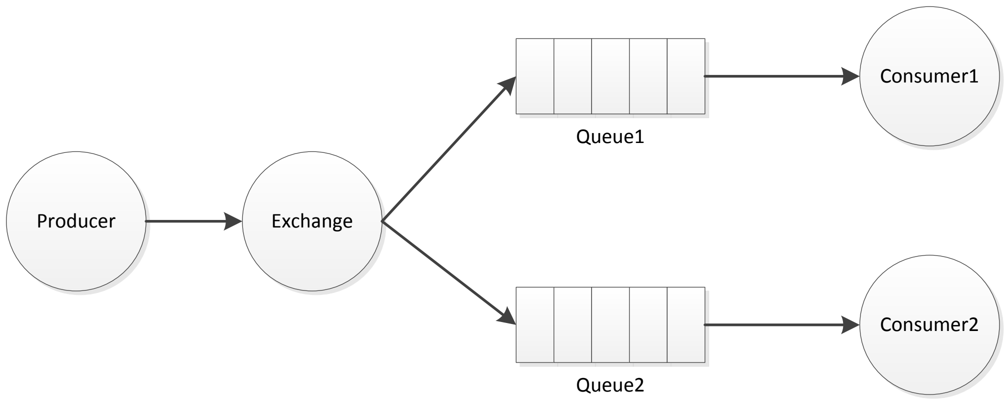


Figure 14. Publish-subscribe.

- **RPC (Remote Procedure Calls):** Where the message broker is used as an exchange and queue for procedure calls. This is useful both for ensuring security as well providing mechanism to manage scalability.

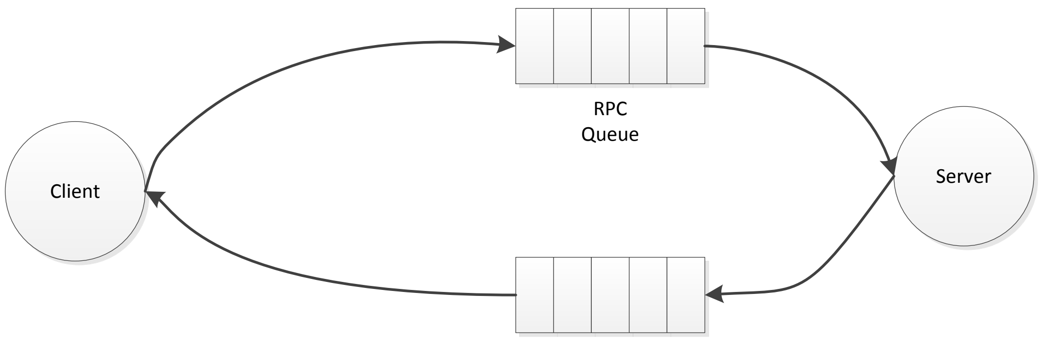


Figure 15. Remote Procedure Call.

To provide an integrated security solution for COMPOSITION, an adapter allowing the authentication and authorization mechanisms of RabbitMQ to be managed by Keycloak is being developed. The same security system can thus be used for intra-factory business user identity, marketplace partners and system components. An adapter for the blockchain distributed trust mechanism will be built to allow the integrity and non-repudiation of broker messages. This is described in section 5.3.11 and section 6.1.

When the broker is used for inter-component communication, logical addressing of components can be used – a component identifier instead of a network address and port – decoupling components and providing a consistent way to address and find them for other components. As mentioned above, authentication and authorization can also be managed in a uniform manner via the broker. As extensibility is a concern for the developer stakeholders, it is desirable to use the broker for all component communication. De-coupled, message-based communication suits the data-centric nature of the COMPOSITION system well, where several components independently subscribe to the same information. However, some exchanges are more suited for request-response interaction, e.g. REST APIs used for querying or administration. These will be using an adapter in RabbitMQ to provide transparent request-response messaging.

The overhead and queue limitations may be a problem with the described design if there are requirements for large response payloads or sub-second response time for these interfaces. However, the current requirements are not expected to have an impact on this design. The centralized approach to communication also introduces a possible bottleneck in the system. Load-balanced clusters of RabbitMQ servers is a tried configuration to deal with scalability of the broker which is expected to be applicable in COMPOSITION. RabbitMQ is also available as highly scalable cloud services[11].

[11] https://www.cloudamqp.com/

5.3.3 Intra-factory Interoperability Layer

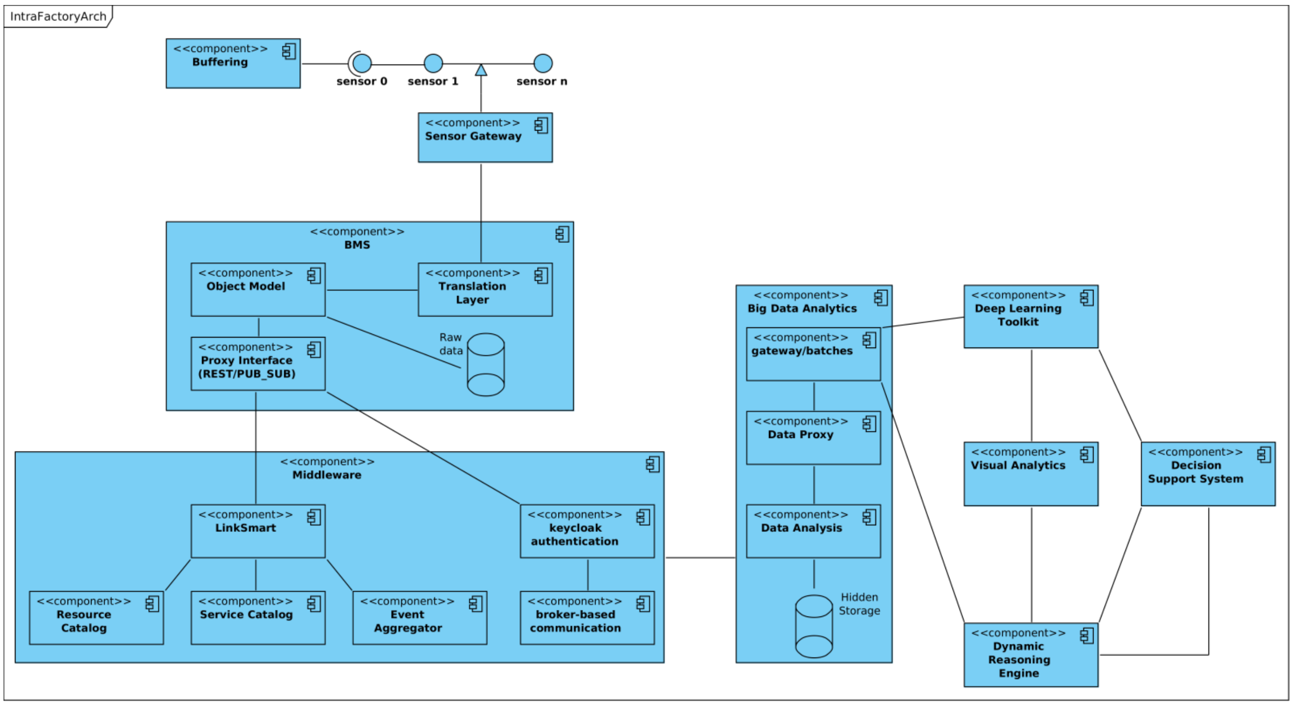

IntraFactoryArch
<<component>> Buffering
sensor 0
sensor 1
sensor n
<<component>> Sensor Gateway
<<component>> BMS
<<component>> Object Model
<<component>> Translation Layer
<<component>> Proxy Interface (REST/PUB_SUB)
Raw data
<<component>> Middleware
<<component>> LinkSmart
<<component>> keycloak authentication
<<component>> Resource Catalog
<<component>> Service Catalog
<<component>> Event Aggregator
<<component>> broker-based communication
<<component>> Big Data Analytics
<<component>> gateway/batches
<<component>> Data Proxy
<<component>> Data Analysis
Hidden Storage
<<component>> Deep Learning Toolkit
<<component>> Visual Analytics
<<component>> Decision Support System
<<component>> Dynamic Reasoning Engine

Figure 16. Intra-factory interoperability layer components and dependencies.

The intra-factory interoperability layer has two main goals: the first one is to provide a model for interconnecting the Composition ecosystem in the intra-factory scenario, the second one is to ensure the conformity between communications among interconnected components. The involved technology provided by development partners of Composition and the connectors that will be defined, developed and deployed to integrate these.

Individual partners' responsibilities and work package outputs, are highlighted in the followings:

- Sensors, Sensors Buffering and Sensors Gateways will be developed and adopted from existing technology. Consideration will be taken to Technical Objective 1.1 (see section 4.4).
- The BMS is provided by a project development stakeholder (NEXTWORKS) and is the translation layer providing shop floor connectivity from sensors to the COMPOSITION system. Raw data storage will be added for offline debug purposes.
- The middleware is the main recipient in which the interoperability single components act
    - LinkSmart is a well-known middleware solution per se and will be customized to satisfy Composition requirements. Components include
        - Resource catalogue, works as resources index
        - Service catalogue, works as service index
        - Event Aggregator, parse messages to ensure well-formed and conformity in data streams
    - Keycloak is a virtual layer that ensures authorization and authentication. Like all security related measures, it will be deployed by the Security Framework.
    - The broker-based intra-factory communication system manages all internal communication in COMPOSITION.
- The Big Data Analytics provides Complex Event Processing (CEP) capabilities for the data provided by the intra-factory integration layer

- The Hidden Storage is a storage not accessible from the outside in which aggregated data are stored for debug purposes, i.e. re-bootstrapping already trained artificial neural networks belonging to the Deep Learning Toolkit and to the Dynamic Reasoning Engine.
- The Deep Learning toolkit component for this intra-factory scenario and an example is described in section 5.3.5.
- The Visual Analytics component is the reporting interface of the Decision Support System and Simulation and Forecasting Toolkit.
- The Dynamic Reasoning Engine is part of the Simulation and Forecasting Toolkit.
- The Decision Support System uses process models to guide the production process.

Although the Human Machine Interfaces are missing in the big picture because these are still under definition, they will be most certainly be connected to this intra-factory diagram.

In order to better specify the scenario above it is worth mentioning that the main differences between the Deep Learning Toolkit and Dynamic Reasoning Engine have been highlighted during the architecture definition workshops. The former acts as a continuous learning toolkit for providing predictions on both historical and live data streams from the shop floor level based on Artificial Neural Networks models and supervised learning techniques. The latter provides simulations to needed components, such as the Decision Support System, based on both live and virtual data in a bidirectional manner, simulating possible criticalities adding hypothetical data perturbation to live streams.

LinkSmart was originally developed within the Hydra co-founded EU project for Networked Embedded Systems. It is an enabler allowing heterogeneous physical devices to be incorporated into their applications through easy-to-use web services for controlling any device. Despite at M9 being almost certain that the a reduced set of LinkSmart functionalities will be adopted it is still under discussion if the Global and Local connectors belonging to the LinkSmart architecture will be stripped down, leaving in the Inter-factory Interoperability Layer an agile tool for improving the versatility of the broker-based communication system infrastructure. The design iterations in the project will produce a LinkSmart configuration that suits the Composition ecosystem.

Finally, it is worth mentioning that the intra-factory interoperability layer is already scouting technologies in order to leverage as much as possible on precious outputs from previous EU co-founded projects. Therefore, the FI-WARE architecture components are investigated as possible relevant Generic Enablers interfaces might be adopted. Moreover, the LinkSmart middleware has been mentioned above and while its inclusion in the intra-factory layer is almost certain, there are unresolved design issues on which version and which subcomponents will complement better the brokering base system that will be deployed at the shop floor level for the intra-factory interoperability layer.

More details will be provided in D5.9 “Intrafactory interoperability layer I” due at M18 and will be reflected in this document in its next iteration.

5.3.3.1 **Iot Hub**

The IoT Hub provides an infrastructure to support continuous data collection from IoT based resources and normally is installed at companies’ premises level. Its mission is to gather the data from the shop floor IoT devices and provide it to any Data Collector Platform or Framework in this case to the Composition IIMS.

The IoT Hub is a framework built based mainly on FIWARE Generic Enablers capable to collect information from IoT devices and distribute them through a NGSI[12] broker. FIWARE exposes to developers Data Context elements or entities (JSON objects) with attributes and metadata with a uniform REST API, which allows the IoT Hub to be an open source software stack aiming to bring Data-level interoperability to the complexity of Iot standards and protocols existing today. The IoT Hub is able to expose all IoT devices information and commands using the Data Context API (NGSI) for communicating devices with NGSI brokers or any other piece which uses the NGSI protocol (Next Generation Service Interface) NGSI9/10, which is now being adopted widely in multiple domains, including manufacturing. Focusing on smart factories, the IoT Hub is dedicated for the collection of data from the shop floor acting as a middleware between the IoT data producers in the shop floor and the data consumers, which can be external services or applications through a NGSI broker or any other kind of broker.

[12] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-10_Open_RESTful_API_Specification

The following picture shows the component architecture of the IoT Hub:

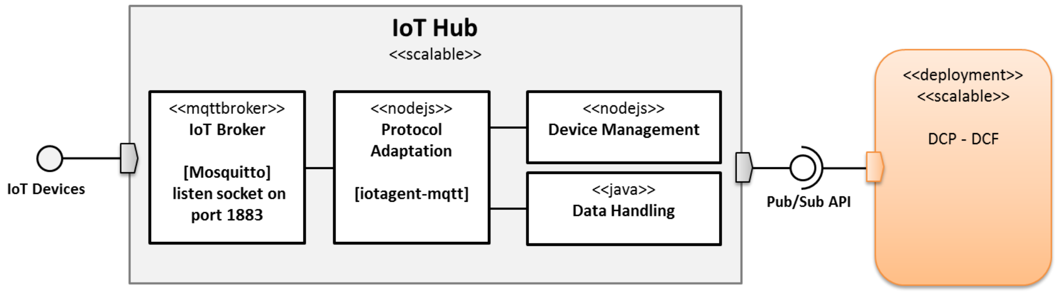


Figure 17. IoT Hub component architecture.

The components are:

- MQTT broker: The selected implementation of the MQTT Broker for the IoT Hub was Mosquitto, although it could be substituted by any other standard MQTT broker.
- FIWARE Backend Device Management GE – IDAS. This component facilitate the communication by connecting IoT devices to FIWARE-based ecosystems, as long as its IoT Agents translate IoT-specific protocols into Next Generation Service Interface (NGSI) context entities that are the FIWARE standard data exchange model. Therefore, the MQTT-JSON IoT Agent acts as a gateway for communicating devices using the MQTT protocol with NGSI brokers (or any other piece which uses the NGSI protocol). The communication is based on a series of unidirectional MQTT topics (i.e.: each topic is used to publish device information or to subscribe to entity updates, but not both).
- FIWARE IoT Data Edge Consolidation GE - Cepheus. It resides inside the IoT Hub to provide Data Handling functionalities and introduced to address the need to process data in real time by implementing features like filtering, aggregating and merging real-time sensor events thanks to the use of smart, but simple, rules described in a SQL-like language. Those rules are completely dynamic and as such they can be updated or deleted on-the-fly using its API. Once data handling techniques are applied, subscribers will be able to collect added-value and relevant data, thus externalizing the responsibility for dedicated development of asynchronous data analysis.

Therefore, the IoT hub will act as an IoT data collector complementary to the LinkSmart and the NXT BMS architecture, and at the same time as a connector for FIWARE compliant platforms, offering interoperability mechanisms between different IoT platforms, enabling the exchange of data from the IIMS with future developments intra/extra factory based on FIWARE components.

5.3.3.2 **Building Management System**

The Intra-factory Interoperability Layer (IIL) component is part of the Integrated Information Management System (IIMS) of COMPOSITION. The main purpose of this layer is allowing a seamless, homogeneous interconnection among all the cyber-physical systems in the factory and the software modules in the upper layer (data processing, decision support, etc.). The IIL has been designed considering the general principles set in the RAMI 4.0 specification, and is split into two logical sub-layers, highlighted in yellow and green in the picture below.

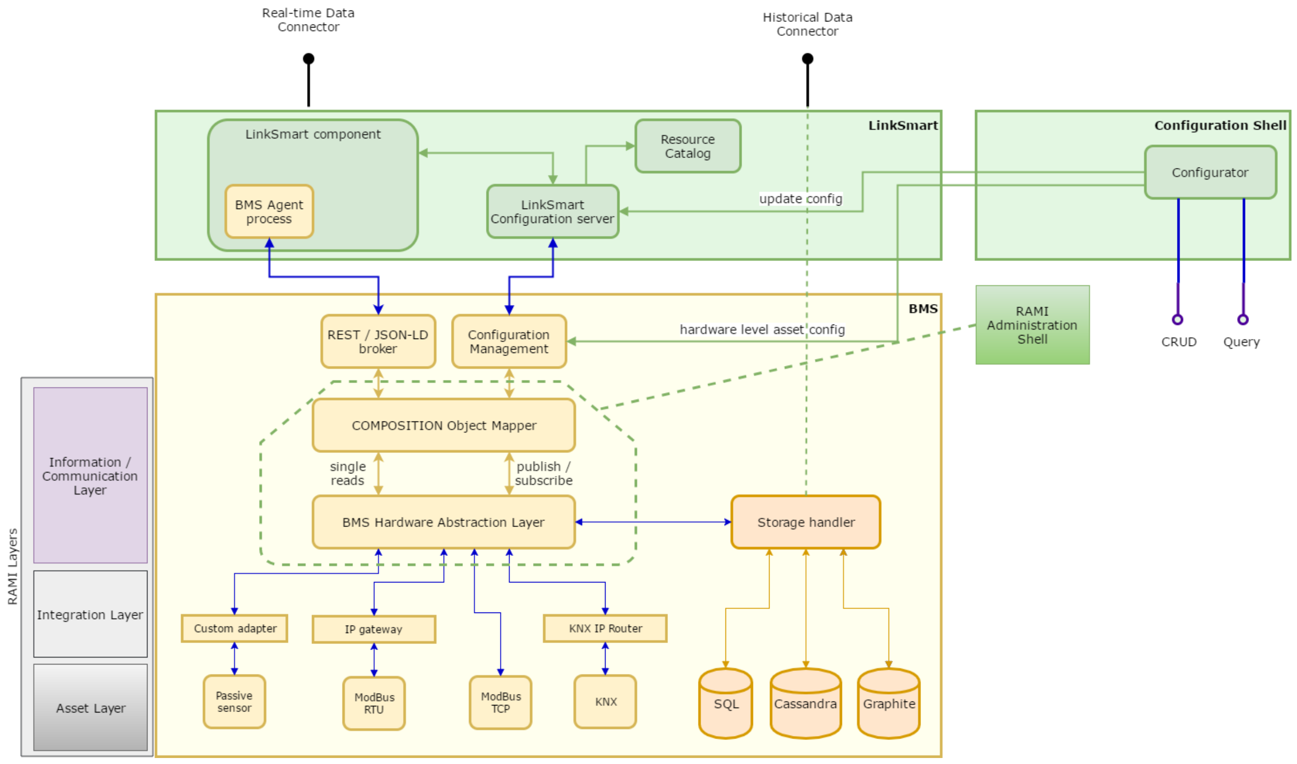


Figure 18. Components and interactions of the BMS: LinkSmart middleware, Configuration Shell, BMS (Building Management System), RAMI Administration Shell.

The lower part (yellow in picture above) is built on top of the existing BMS software modules provided by NXW, which guarantee low level interoperability with a number of different field buses (this is positioned at the Asset / Integration RAMI layers). Such modules gather data read from the sensors installed in the local environment, interconnected through different field buses (e.g. KNX, Modbus, BACnet), and organize it into a uniform Data Model. This model provides a representation of sensor and actuator data which is independent of the physical type of underlying devices (Information/Communication RAMI layers).

The BMS HAL and COMPOSITION Object Mapper expose a virtualized version of the underlying physical objects from which information can be read and actuations can be performed, thus providing the equivalent of an Administration Shell in the RAMI architecture.

The upper part (green the picture above) is made of components belonging to the LinkSmart architecture, and provides both real-time and historical data connectors for the other IIMS components. Communication between LinkSmart and the BMS components will be done through standard LinkSmart interfaces, implemented into the BMS Agent Process.

5.3.4 Big Data Analytics

Manufacturing in assembly lines consist in a set of hundreds, thousands or millions of small discrete steps aligned in a production process. Automatized production processes or production lines, they produce for each of those steps small bits of data in form of events. The events possess valuable information, but this information loses the value through time. Additionally, the data in the events usually are meaningless if they are not contextualized, either by other events, sensor data or process context. To extract most value of the data, it must be process as it's produced. In other words, in real-time and on demand. Therefore, we prose for the Big Data Analysis; the usage of Complex-Event Processing for the data management coming from the production facilities. In this manner, the data is processed at the moment when it is produced extracting the maximum value, reducing latency, providing reactivity, giving it context, and avoiding the need of archiving unnecessary data.

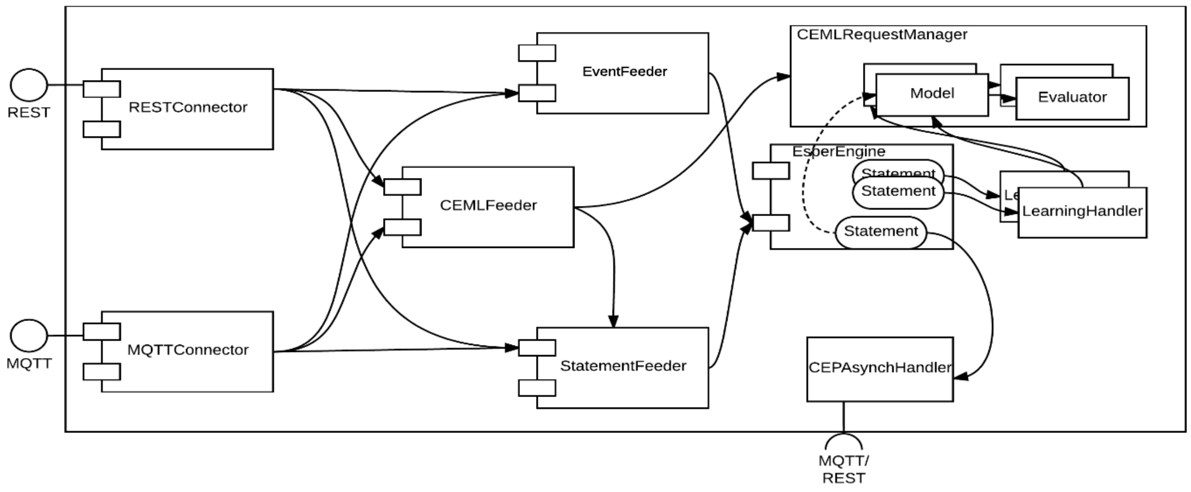


Figure 19. LinkSmart® Learning Service Architecture Sketch.

The Complex-Event Processing service is provided by the LinkSmart® Learning Service (LS). The LS is a Stream Mining service that provide means to manage real-time data for several propose. In the first place, the LS provide a set of tools for collect, annotate, filter, aggregate, or cache the real-time data incoming from the production facilities. This set of tools facilitate the possibility to build applications on top of real-time data. Secondly, the LS provide a set of APIs to manga the real-time data lifecycle for continuous learning. Thirdly, the LS can process the live data to provide complex analysis creating real-time results for alerting or informing about important conditions in the factory, that may be not be seeing at first glance. Finally, the LS allows the possibility to adapt to the productions needs during the production process. The diagram below, the use cases that the LS enable are presented.

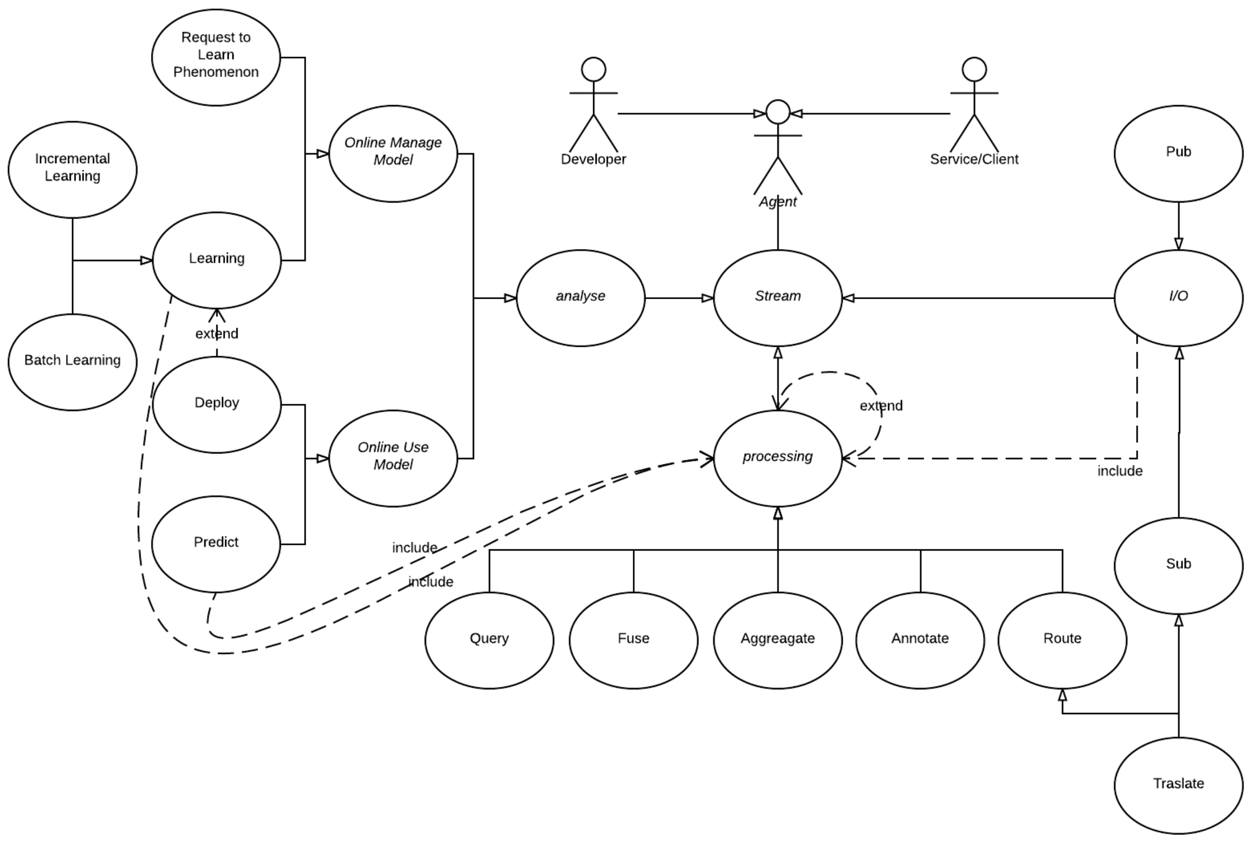


Figure 20. LinkSmart® Learning Service Enabling Use Cases.

It's worth mentioning that the LS do not learn from the data, it just facilitates the data to the models. In other words, the LS connects externally to the models for the learning process. By this the LS enables the online real-time learning process and data deliverable for training the model. In COMPOSITION, the external learning models will be provided by Deep Learning Toolkit. Nevertheless, the LS is capable do on the run analytics using less historical data intensive algorithms such as Random Forest, Gradient Boosting, Kalman Filters, Particle Filter, Hidden Markov Models, *boosted* Artificial Neural Networks. With them it may be possible to predict certain phenomenon without the need of historical data.

The LS has been developed and tested in different EU projects such as ALMANAC[13] and IMPReSS[14]. However, the use cases where in the scope of Smart Cities or Smart Buildings, and it must be tailored for more Industry 4.0 oriented use cases where the events are driven by a controlled and data intensive production process.

5.3.5 Deep Learning Toolkit

5.3.5.1 **Introduction**

The Deep Learning Toolkit delivers prediction and forecasting of relevant indicators based on machine learning models. It is a component of the Composition ecosystem and belongs to both the intra and inter-factory scenario. In the former it is in charge of analysing the shop floor parameters feed to the component by the IIMS and the BMS (section 5.3.3.2) through the middleware belonging to the Intra-factory Interoperability Layer, mediated by the pre-processing of the Big Data Analytics tool. Then, a continuous learning process produces predictions based on the analysed data, addressing use cases mainly related to predictive maintenance, and feeds these data to all components connected to the Intra-factory Interoperability Layer, especially the Decision and Support System.

The Deep Learning Toolkit component has a lifecycle of mainly four phases:

[13] http://cordis.europa.eu/project/rcn/109709_en.html
[14] http://cordis.europa.eu/project/rcn/185510_en.html

- Offline training phase
- Validation phase
- Testing phase
- Continuous learning phase

The offline training phase, as it's named after, starts with an offline analysis of the historical data and takes place outside the shop-floor.

The validation phase takes also place offline and it's the phase in which the parameters of the Artificial Neural Network are adjusted in order to reach the threshold set for an acceptable accuracy level.

The training phase is also consequent to the validation and it's the phase where the component and the Artificial Neural Network has consumed all historical data for having a robust algorithm ready to be deployed.

The continuous learning phase is the longest of the four phases and takes place at the shop-floor level, at the premises of end-user that has provided the shop floor data. In this phase, the component is online and connected to the Composition ecosystem, where it learns from data batched coming from the shop-floor in near real-time.

5.3.5.2 **Interfaces, Interactions and used standards**

The Continuous Deep Learning Toolkit module receives raw and pre-processed input data from the middleware (Adaptation Layer for Intra-factory Interoperability) and from the Big Data Analysis module respectively. The latter module also provides the Continuous Deep Learning Toolkit with rewarding signals to foster internal continuous retraining.

The Continuous Deep Learning Toolkit module publishes to the middleware reports and statistics about its estimated accuracy in addition to the predicted targets based on live data streams. It is worth noting that the middleware is not the final destination of this flow of predictions, but it only broadcast it: various IIMS modules may leverage these data. These two operations are asynchronous and independent one from another. Typically, new targets will be published as soon as new data are processed, while the retraining resulting in updated accuracy report can take place at lower frequency.

The stack diagram in Figure 21 depicts the most relevant interactions of the two aforementioned modules with the middleware.

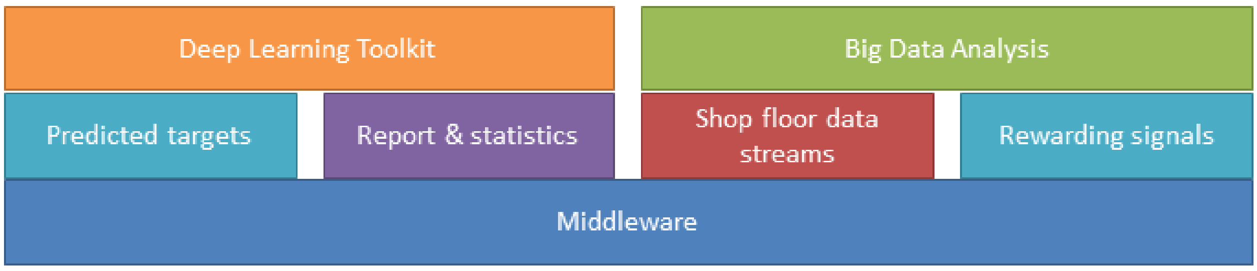


Figure 21. Deep Learning Toolkit interactions.

The Continuous Deep Learning Toolkit module will perform read and write operations to the middleware its message protocol which is likely to be MQTT based. Apart from this, the API provided to other modules to access and to provide data will be based on REST services. In both cases, the message payload will adopt common formats such as XML and JSON. Data will be formatted conforming to the commonly agreed schemas (for example, defined in XSD format or JSON Schema).

5.3.5.3 **Sequence diagrams**

The activity diagram in Figure 22 describes the offline model training based on historical data. The process is as follow:

The Continuous Deep Learning Toolkit requests historical data set to the appropriate data sources including Big Data Analysis module. The latter request, receive and pre-process the historical data set on its own and then send the output analytics to the Continuous Deep Learning Toolkit.

When the Continuous Deep Learning Toolkit receives both raw historical data and the pre-processed analytics merge them into the final dataset that is used for the training activity: this operation includes the data shuffling and splitting the dataset into three partitions dedicated to training validation and test. The most appropriate model is chosen for the prediction task to be accomplished.

Then, iteratively, multiple combinations of hyper-parameters, determining the model complexity and expressivity, are taken into account. For each combination, a model is trained over the training set in order to minimize the global prediction error and validated against the validation set.

The trained model performing at best on the validation set is retained and finally assessed against the test set, resulting in an accuracy report that marks the end of offline training and that is published on the middleware.

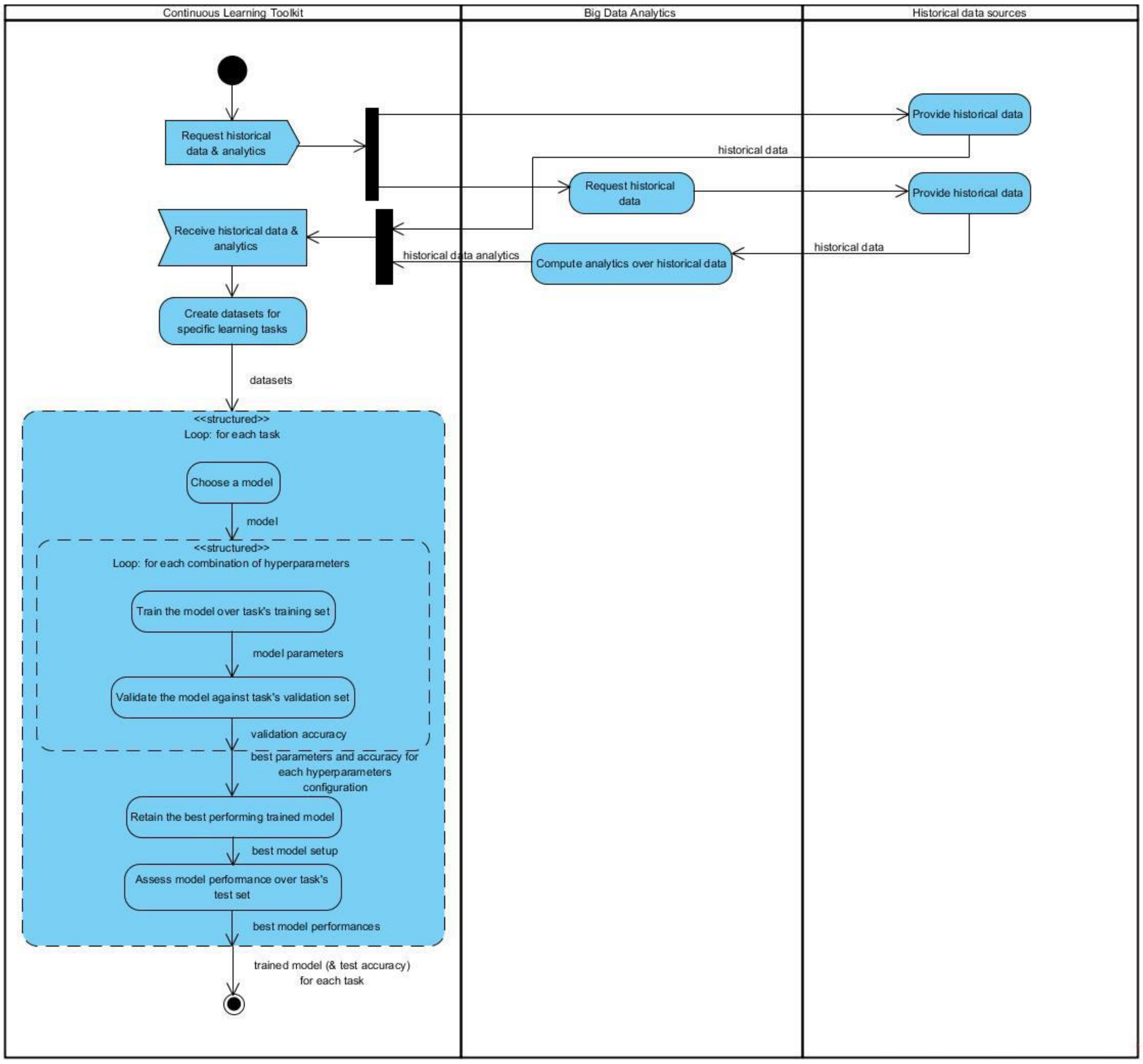

Continuous Learning Toolkit
Big Data Analytics
Historical data sources
Request historical data & analytics
Provide historical data
historical data
Request historical data
Provide historical data
Receive historical data & analytics
historical data analytics
Compute analytics over historical data
historical data
Create datasets for specific learning tasks
datasets
<<structured>>
Loop: for each task
Choose a model
model
<<structured>>
Loop: for each combination of hyperparameters
Train the model over task's training set
model parameters
Validate the model against task's validation set
validation accuracy
best parameters and accuracy for each hyperparameters configuration
Retain the best performing trained model
best model setup
Assess model performance over task's test set
best model performances
trained model (& test accuracy) for each task

Figure 22. Offline training sequence diagram.

The activity diagram in Figure 23 describes the activities the Continuous Deep Learning Toolkit executes in a production environment. The module stays idle waiting for incoming data from potentially heterogeneous (but known) sources either internal – including the Agent module, the Big Data Analytics module – or external. As soon as data are available the module uses its internal module to elaborate predictions which are made available to other modules both over the middleware and with REST APIs.

An example of the latter modality is shown in the second action where the Agent module requests to the Continuous Deep Learning Toolkit through the proper API a profiling report of other agents/or opportunities and the Continuous Deep Learning Toolkit replies with the most recent forecasting report available.

The third activity shows a very similar process where other modules of the IIMS requests forecasted indicators and stats and receives the most recent estimates.

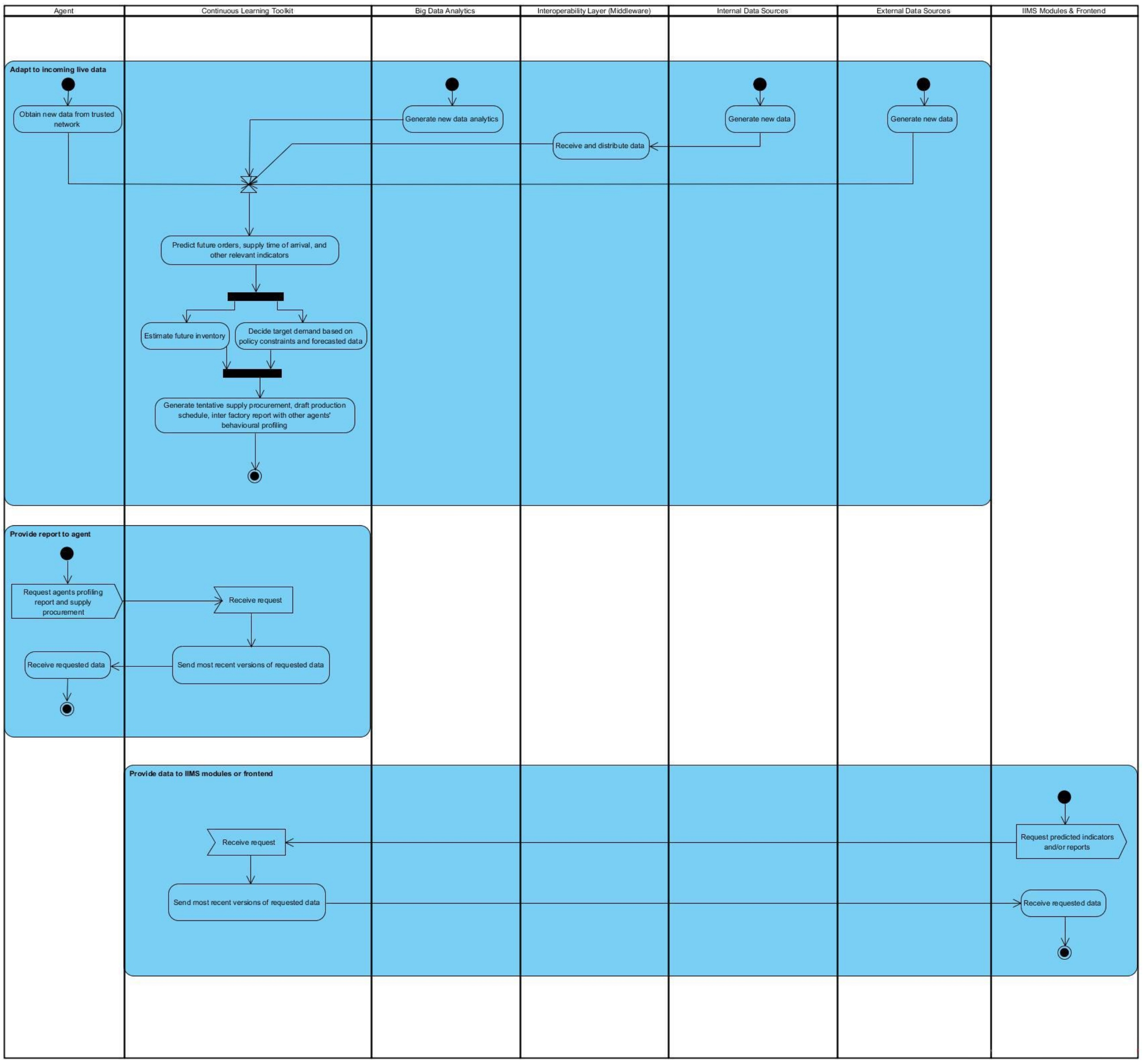

Agent
Continuous Learning Toolkit
Big Data Analytics
Interoperability Layer (Middleware)
Internal Data Sources
External Data Sources
IIMS Modules & Frontend
Adapt to incoming live data
Obtain new data from trusted network
Generate new data analytics
Receive and distribute data
Generate new data
Generate new data
Predict future orders, supply time of arrival, and other relevant indicators
Estimate future inventory
Decide target demand based on policy constraints and forecasted data
Generate tentative supply procurement, draft production schedule, inter factory report with other agents' behavioural profiling
Provide report to agent
Request agents profiling report and supply procurement
Receive request
Receive requested data
Send most recent versions of requested data
Provide data to IIMS modules or frontend
Receive request
Request predicted indicators and/or reports
Send most recent versions of requested data
Receive requested data

Figure 23. Online forecasting sequence diagram.

5.3.6 Decision Support System

The Decision Support System (DSS) component is part of the Integrated Information Management System. The IIMS is also part of the high-level platform of COMPOSITION. The main aim of the DSS will be to make a step forward towards a better understanding of the involved manufacturing processes and operations, the contribution of individual links of the supply chain, the effect of process monitoring in productivity, to facilitate communication and knowledge sharing among departments with different roles and responsibilities, the maintenance requirements and procedures and the detection of daily production details and flaws

The Components/Functional Packages of the DSS are shown in the figure below.

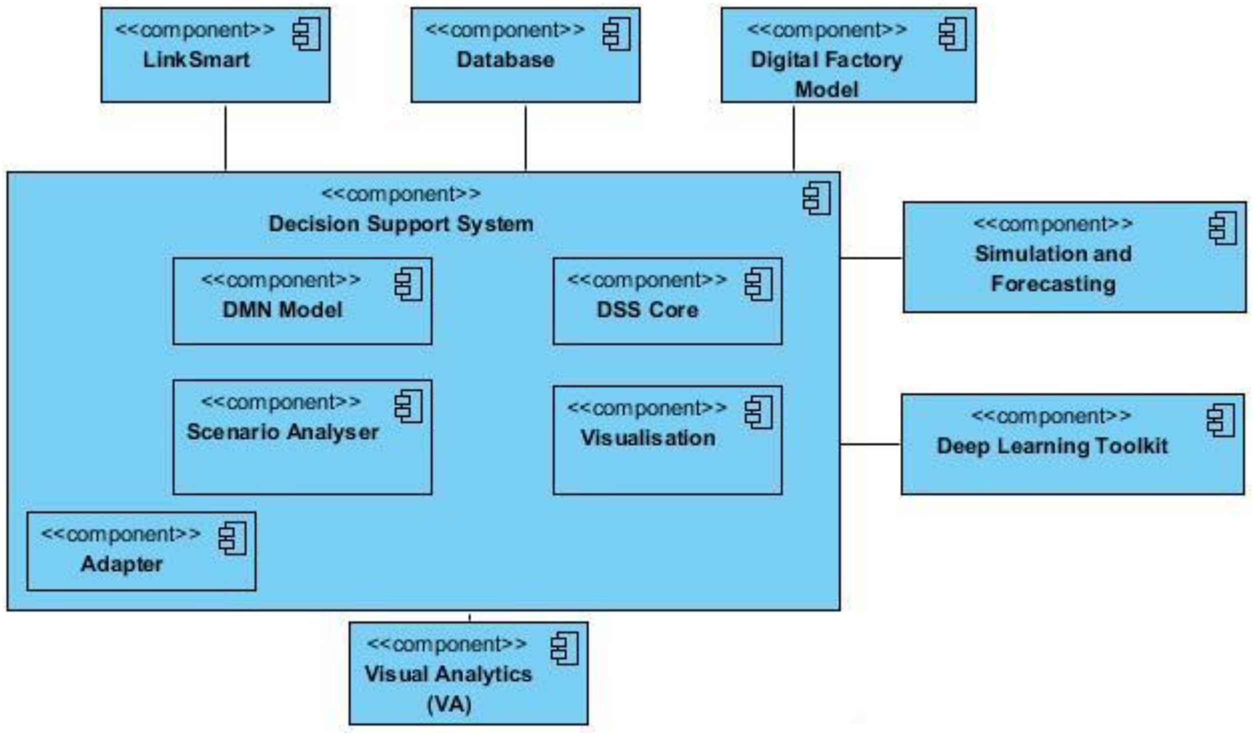

<<component>> LinkSmart
<<component>> Database
<<component>> Digital Factory Model
<<component>> Decision Support System
<<component>> DMN Model
<<component>> DSS Core
<<component>> Scenario Analyser
<<component>> Visualisation
<<component>> Adapter
<<component>> Simulation and Forecasting
<<component>> Deep Learning Toolkit
<<component>> Visual Analytics (VA)

Figure 24. Design and dependencies of the Decision Support System (DSS): LinkSmart middleware, DataBase, Digital Factory Model (DFM), Simulation and Forecasting Tool, DeepLearning Toolkit, Visual Analytics (VA).

The main inputs, outputs and functionalities follow, in order to help gain a better understanding of how this component will operate.

Main input(s) required for the DSS are:

- LinkSmart middleware (as data input)
- Deep Learning Toolkit (as a service provided)
- Simulation And Forecasting Tool (as a service provided)
- Digital Factory Model (DFM) (as a schema for internal data)

Main output(s) that the DSS gives out:

- LinkSmart (Key Performance Indicator alerts, etc)
- Visual Analytics (VA) (integrated view)

Main functionalities (high level):

- Proposes Actions based on scenarios
- Visualizes overall system behavior
- Optimizes procedures for proposed targets (restrictions)

Main functionalities of DSS toolkit are:

- Proposes Actions based on scenarios by using Combination of information provided by the Business Process Diagrams (BPB) and the Digital Factory Model (DFM) based on the produced semantic models, coming from all the involved stakeholders.
- Visualize overall system behavior and understanding between the involved manufacturing processes and operations by individual links in the supply chain, as well as the effect of process monitoring in productivity and facilitation of communication and knowledge sharing between departments with different roles and responsibilities and finally the maintenance requirements and procedures and the detection of the daily work flow helps business people to find optimal solutions

among multiple alternatives subject to different business constraints and for proposed targeted scenarios (restrictions).

It was necessary to orchestrate different components within the DSS, which would interact with each other in a specific manner, so as to achieve the described functionalities and to guarantee a smooth implementation.

The main components of DSS are:

- Decision Model and Notation (DMN)[15] Modeling: This toolkit provides models for the decision support. Its main target is to represent the processes suitable for the decision support engine.
- The scenario analyser: It uses the simulation and forecast engine to analyses scenarios, and with the modelling restrictions to provide the optimum processes.
- KPI Toolkit: It analyses the behaviour of the system and uses the Deep Learning Toolkit to provide an overall image for the system.
- Visualisation: This is a set of various components for the visualization of the results of the whole decision support system.
- Adapter: Provides a managed environment for the DSS. Translate data from various format, add watcher for several events. Provides a common command interface for the system.

In order to facilitate interactions within the different DSS components and the COMPOSITION Ecosystem, an API has been made available.

DSS API	**Description**
KPI API	The API exposes the KPI's as a service. The API exposed as a RESTful web service.
Core API	Main API of DSS for rules, models, events

The following diagrams display a typical execution from an event received via middleware. The DSS reads the data from DFM, uses the Deep Learning Toolkit as a service and visualise the result. Also provides the results back to ecosystem.

[15] http://www.omg.org/spec/DMN/

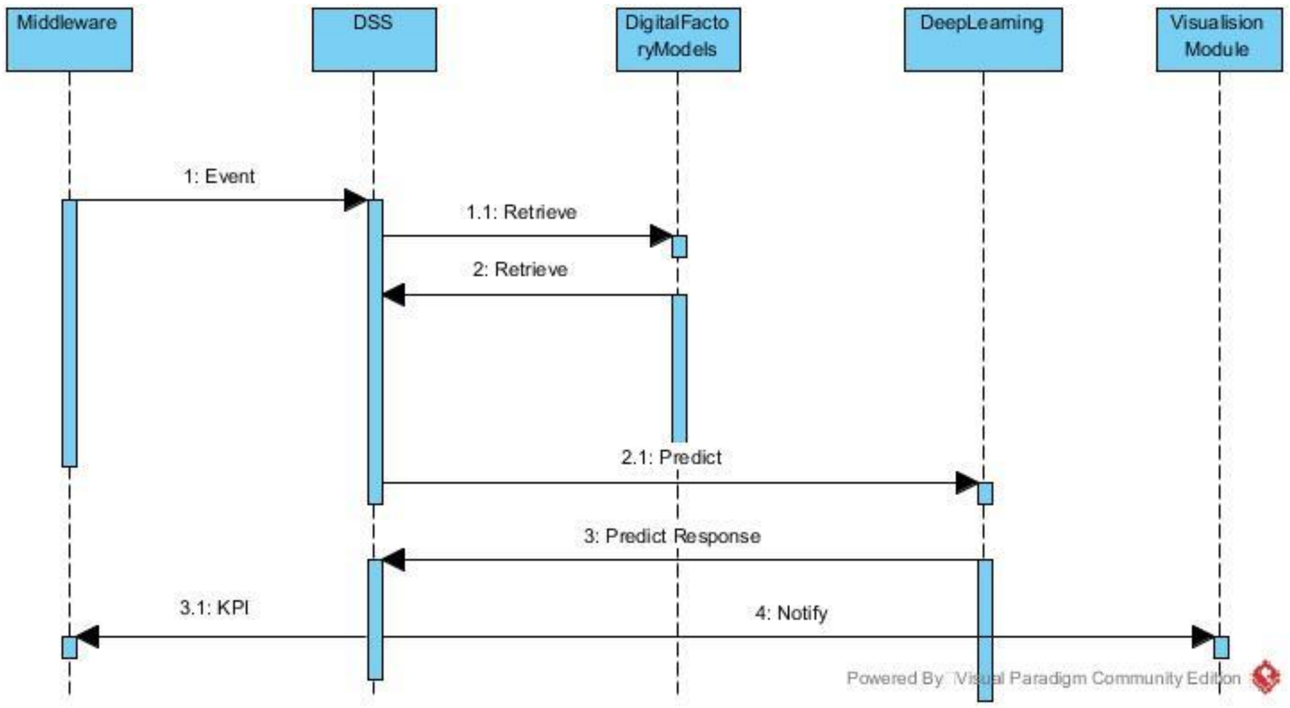


Figure 25. Typical protocol execution.

5.3.7 Simulation and Forecasting

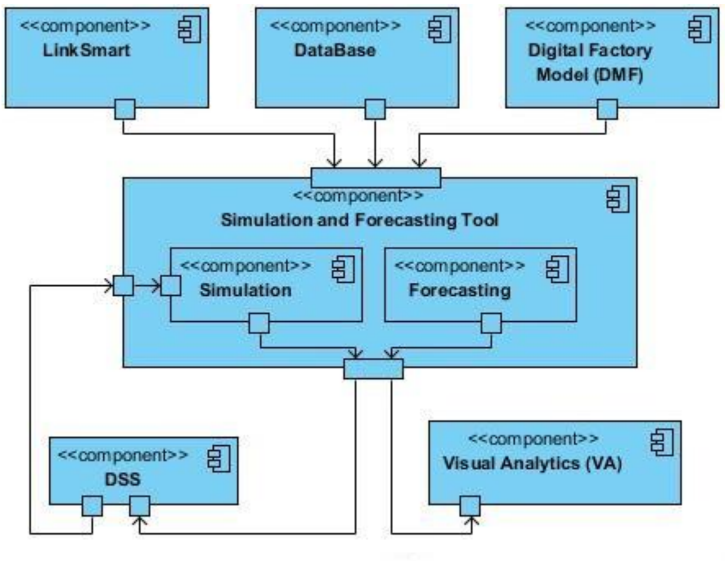

<<component>> LinkSmart
<<component>> DataBase
<<component>> Digital Factory Model (DMF)
<<component>> Simulation and Forecasting Tool
<<component>> Simulation
<<component>> Forecasting
<<component>> DSS
<<component>> Visual Analytics (VA)

Figure 26. The Simulation and Forecasting Tool and dependencies: LinkSmart middleware, DataBase, Digital Factory Model (DFM), Decision Support System (DSS) and Visual Analytics (VA).

The Simulation and Forecasting Tool component is part of the high-level platform of COMPOSITION, the Integrated Information Management System (IIMS), and its main purpose is to simulate processes models and to provide forecast of events whose actuals outcomes have not yet been observed. This component will provide a constantly updated sensing layer regarding the integration of different sensors so as to support a Dynamic Reasoning Engine (DRE) and alarming services in production and logistics.
The main inputs of Simulation and Forecasting engine component are real time data coming from Linksmart middleware, historical data coming from COMPOSITION Database and models of processes from Digital Factory Model (DFM).

Main input(s):

- LinkSmart middleware
- DataBase
- Digital Factory Model (DFM)

Main output(s):

- Decision Support System (DSS)
- Visual Analytics (VA)

Main functionalities:

- Simulation of process or logistic models
- Forecast future outcomes based on models

Simulation and Forecasting Tool component is divided in to sub-components: *Simulation* and *Forecasting*. *Simulation sub-component* will simulate models (provided by DFM component) on historical data (provided by COMPOSITION Database) or real-time data (provided by COMPOSITION LinkSmart middleware) so as to provide results on several process or logistic scenarios, according to projects' use case. The initial internal parameters of a simulation scenario will be defined be the user, accordingly. The simulation results will be fed into the COMPOSITION Decision Support System (DSS), where by an internal procedure there will be a decision(s) for more scenarios to be simulated or not, regarding the tested process. Possible models that at first fit to a process and subsequently simulated could be several approaches of regression, such as *linear*, *ridge*, *lasso*, and *elastic net* regression.
Forecasting sub-component will provide predictions of future events for the selected process model, based on the model parameters decided by the COMPOSITION Decision Support System (DSS) for the specific process when the iterations of simulation process end. The forecasting scenarios will be the presented in the most compelling way with advanced and innovative data visualization techniques utilizing an interactive human-machine interface.

5.3.8 Matchmaker

COMPOSITION Matchmaker is designed to be the core component of COMPOSITION Broker. It supports both syntactic and semantic matching in terms of manufacturing capabilities, in order to find the best possible supplier to fulfill a request for a service, raw materials or products involved in the supply chain. Different decision criteria for supplier selection according to several qualitative and quantitative factors will be considered by Matchmaker.

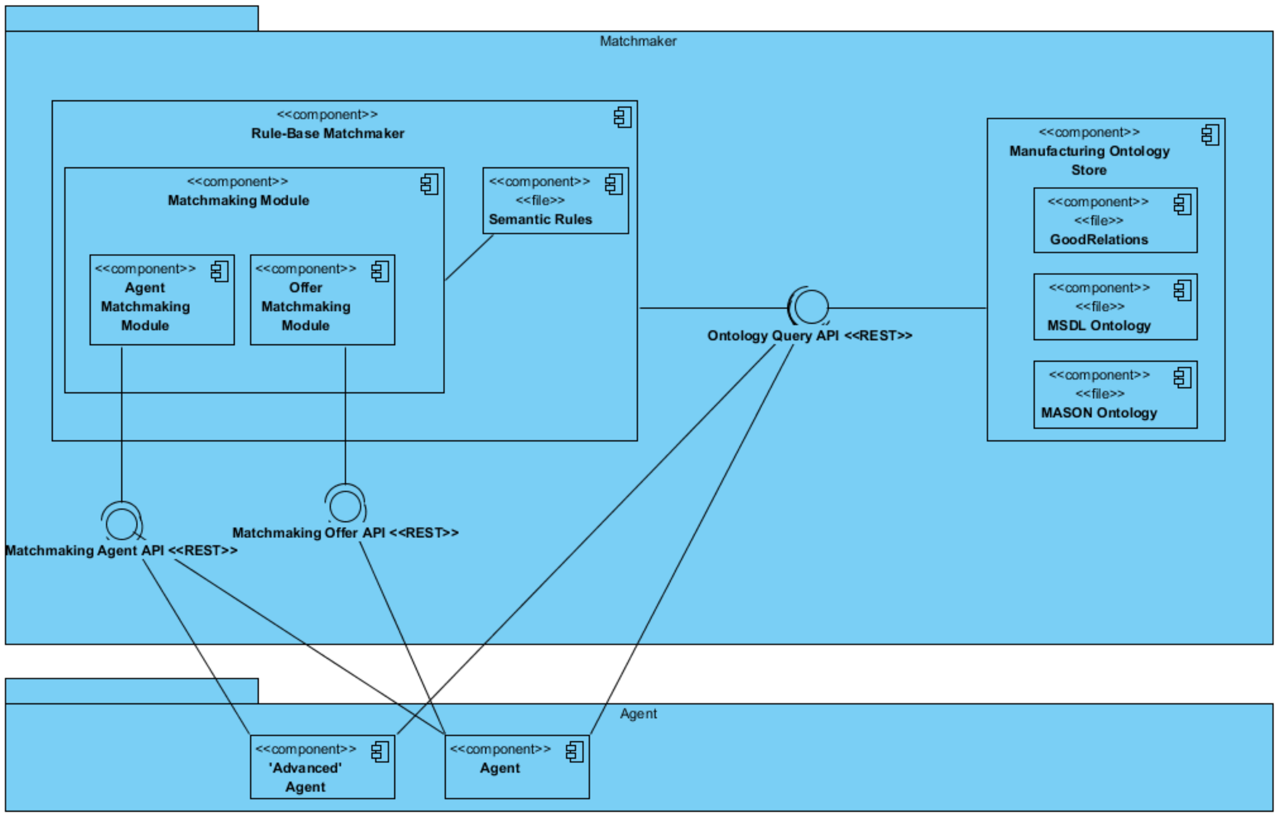

Matchmaker
<<component>> Rule-Base Matchmaker
<<component>> Matchmaking Module
<<component>> <<file>> Semantic Rules
<<component>> Agent Matchmaking Module
<<component>> Offer Matchmaking Module
<<component>> Manufacturing Ontology Store
<<component>> <<file>> GoodRelations
<<component>> <<file>> MSDL Ontology
<<component>> <<file>> MASON Ontology
Ontology Query API <<REST>>
Matchmaking Offer API <<REST>>
Matchmaking Agent API <<REST>>
Agent
<<component>> 'Advanced' Agent
<<component>> Agent

Figure 27. COMPOSITION Matchmaker component diagram.

Main Input(s):

- Agents (Matchmaker gets agents' requests as input)

Main Output(s):

- Agents (Matchmaker response to agents' requests)

The Matchmaker architecture is depicted in the figure above. As it can be observed, two main components are included: the Rule-Base Matchmaker and the Manufacturing Ontology Store.

The Rule-Base Matchmaker is developed in Java language and it is offered through RESTful web services. The Rule-Base Matchmaker will be used by Marketplace's agents in order to match requests and offers between the agents. Its core component is the Matchmaking Module which consists of the following functions:

- Agent Matchmaking Module: This module interacts with agents. An agent sends a request for a service to the Agent Matchmaking Module which applies a set of rules in collaborative manufacturing service ontology. Then the matchmaker module sends a response to the agent with a list contains the agents who support a matching offer for this request.
- Offer Matchmaking Module: This module interacts only with default agents ('thin' agents) who cannot evaluate offers by themselves (as the 'advanced' agents do) but they use the matchmaker exclusively. An agent sends a request for a service to the Offer Matchmaking Module which applies a set of rules in collaborative manufacturing service ontology. The set of rules considers several qualitative and quantitative factors to match the agent's request with the best available offer and it is not limited to match the agent with all the other agents that can support his request as the Agent Matchmaking Module does. So, the response of the Offer Matchmaking Module is the best available offer.

The Semantic Rules component is a sub-component of Rule-base Matchmaker which contains all the files with rules. These rules will be applied by Matchmaking Module to collaborative manufacturing service ontology in order to extract the requested matching. The rules are in Jena format.

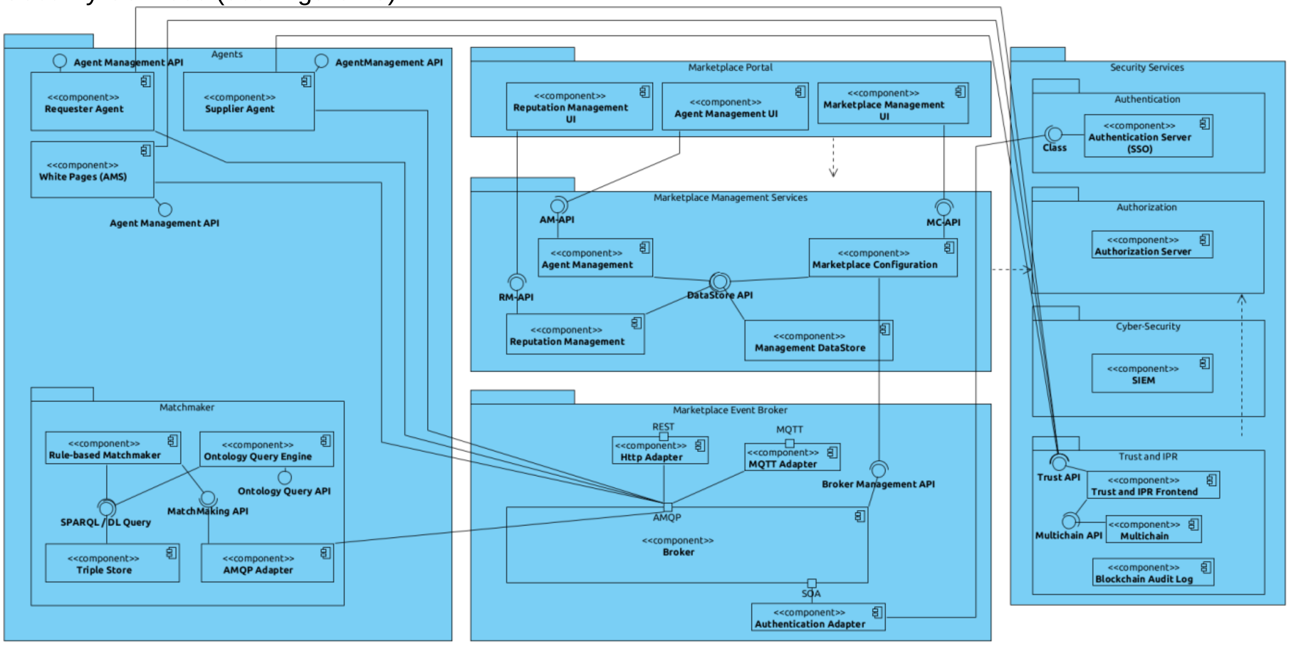
The Manufacturing Ontology Store is the knowledge base for the COMPOSITION Marketplace. Actually, it is a collaborative manufacturing services ontology which supports flexible specification and execution of manufacturing collaboration schemes. It is described in detail in the section of Marketplace Ontology.

Table 2. Matchmaker Main APIs.

Matchmaker APIs	Description
Matchmaking Agent API	The API receives as input an agent's request and sends back to the agent the Matchmaker's output which contains a list with the matching agents and their offers for this request. Input and output are in JSON format. The API exposed as a RESTful web service.
Matchmaking Offer API	The API receives as input an agent's request with a list contains the available offers which fulfil this request and sends back to the agent the Matchmaker's output which contains the best matching offer. Input and output are in JSON format. The API exposed as a RESTful web service.
Ontology Query API	This API receives as input an agent's command (e.g. import/export data) and response back to the agent with the expected command's output. Inputs and outputs are in a predefined common format (e.g. JSON). The API applies the command as a SPARQL query (e.g. Insert/Select commands) into the Manufacturing Ontology Store. This interface is defined by the REST protocol.

5.3.9 Marketplace

The COMPOSITION marketplace is composed of four main building blocks: The Agents, the Management Portal and Services, the Communication Infrastructure (namely the Marketplace Event Broker) and the Security Services (see Figure 28).



Agents
Agent Management API
AgentManagement API
<<component>> Requester Agent
<<component>> Supplier Agent
<<component>> White Pages (AMS)
Agent Management API
Matchmaker
<<component>> Rule-based Matchmaker
<<component>> Ontology Query Engine
Ontology Query API
MatchMaking API
SPARQL / DL Query
<<component>> Triple Store
<<component>> AMQP Adapter
Marketplace Portal
<<component>> Reputation Management UI
<<component>> Agent Management UI
<<component>> Marketplace Management UI
Marketplace Management Services
AM-API
MC-API
<<component>> Agent Management
<<component>> Marketplace Configuration
RM-API
DataStore API
<<component>> Reputation Management
<<component>> Management DataStore
Marketplace Event Broker
REST
MQTT
<<component>> Http Adapter
<<component>> MQTT Adapter
Broker Management API
AMQP
<<component>> Broker
SOA
<<component>> Authentication Adapter
Security Services
Authentication
Class
<<component>> Authentication Server (SSO)
Authorization
<<component>> Authorization Server
Cyber-Security
<<component>> SIEM
Trust and IPR
Trust API
<<component>> Trust and IPR Frontend
Multichain API
<<component>> Multichain
<<component>> Blockchain Audit Log

Figure 28. Marketplace components.

Agents may implement market-specific services, such as the white pages agent or the matchmaker, or they can act on behalf of industry stakeholders participating in the marketplace. Required communication infrastructure is provided by a suitable message broker (namely the Marketplace Event Broker), which

provides message delivery services to all other components through a well-known, publish-subscribe, interaction paradigm.
The set of components formed by the Marketplace Portal and the Marketplace Management Services has been designed to offer suitable means to administer marketplaces, register new market stakeholders, provide access credentials and connection parameters for agents to be deployed on the COMPOSITION market, and the like. This design choice allows stakeholders to easily manage the entire marketplace infrastructure, e.g., for defining new Closed Marketplaces.
Transactions and interactions between components in the platform are subject to a certain number of security checks and procedures aimed at ensuring a high degree of trust and reliability of exchanged information. These involve, among the others (better explained in 5.3.10), restricted access to the marketplace communication infrastructure, channel encryption, provenance assessment techniques for messages, audit logs on message trails, etc. To support marketplace components in achieving such a trusted and secure operation, a dedicated set of components is purposely part of the marketplace design: the so-called Marketplace Security Services (described in Section 5.3.11).

5.3.9.1 **Agents**

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, 2 main categories of agents can be defined a priori, depending on the kind of provided services:

- Marketplace agents
- Stakeholder agents

The former category groups all the agents providing services that are crucial for the marketplace to operate. The latter category, instead, groups agents developed and deployed by the marketplace stakeholders to take part in chain formation rounds.

5.3.9.1.1 Marketplace agents

The minimum set of marketplace agents required in COMPOSITION is composed by a single instance: the White Pages agent or AMS (Agent Management Service). This agent has the responsibility of tracing all the agents operating on the marketplace, providing information about their status as well as some very basic search capability. In order to avoid re-inventing the wheel, COMPOSITION adopts the Foundation for Intelligent Physical Agents (FIPA) definition of AMS and the corresponding set of offered services, summarized in Table 3.

Table 3. AMS services.

Service	**Description**
register	Allows registering an agent as "active" on the marketplace.
deregister	Removes an agent from the list of agents currently active on the market
modify	Modifies the registration information about a certain agent
search	Allows searching for active agents, given a set of search constraints

Every agent in the marketplace shall register and update its status on the AMS prior to any operation. This allows, for example, monitoring the overall status of the market as well as computing some metrics on the global system performances. Moreover, AMS registration avoids involvement of non-active agents in exchanges, thus simplifying the interaction management logic at the agent side.

Part of the AMS functions can be overridden by services provided by additional marketplace agents, e.g., the Directory Facilitator (or the Matchmaker) in COMPOSITION. This typically holds for search functions, as AMS-level implementations are seldom sufficient for effective search and match of agent services, especially when the required matching grammars are complex or when the number of agents in the marketplace is particularly high.

Due to its crucial role in the marketplace operation, the white pages agent is a critical point of failure in the market infrastructure. The related issues in case of breakdowns are therefore specifically addressed from the very first design phases. In particular, 2 main approaches are deployed to address possible failures of the AMS:

1. Sufficient redundancy is provided, defining mechanisms for take-over and hot-swapping of similar agents in case of failures
2. An initial design of AMS-less operation based on broadcast inquiry messages is under definition, which will permit to overcome single-point-of-failure issues at the cost of increased traffic on the marketplace communication infrastructure.

In the COMPOSITION marketplace design, advanced agent search and other complex matching functionalities are delegated to a special purpose agent, playing a superset of the roles of Directory Facilitator and Agent Broker (in FIPA jargon). Due to the inherent complexity of such an agent, a full subsection is devoted to its description: Section 5.3.8.

5.3.9.1.2 Stakeholder Agents

Stakeholder agents design is in principle out of scope for this architecture definition. In fact, every stakeholder may implement (or require a third party to implement) its own agent, following its own policies and assumptions. The only constraint set by the COMPOSITION marketplace on stakeholders' agent design is the ability to exchange messages using the Composition eXchange Language (CXL) and the capability to support a subset of well-known interaction protocols (e.g., CONTRACT-NET), mainly stemming from FIPA specifications.

Nevertheless, in the project, some reference design and implementations are provided with the aim of: (a) providing a hands-on guidance to agent development and deployment, (b) lowering the access step to the market by offering default, customizable implementations for the main agent roles envisioned in the market. The latter motivation drives the definition of the initial two types of stakeholders' agents, presented in this architecture specification: the Requester agent and the Supplier agent.

Requester agent

The Requester Agent is the agent exploited by a factory to request the execution of an existing supply chain or to initiate a new supply chain. Due to the dynamics of exchanges pursued in COMPOSITION, there is no actual distinction between the two processes, i.e., for any supply need a new chain is formed and a new execution of the chain is triggered. The Requester agent may act according to several negotiation protocols, which can possibly be supported by only a subset of the agents active on a specific marketplace instance. The baseline protocol, which must be supported by any COMPOSITION agent, is the so-called CONTRACT-NET. In such a protocol, a Requester agent plays the role of "Initiator".

While describing the CONTRACT-NET is not in the focus of this document, a sample exchange between the Requester agent and its Supplier agents counterpart is reported in Figure 29, for the sake of clarity.

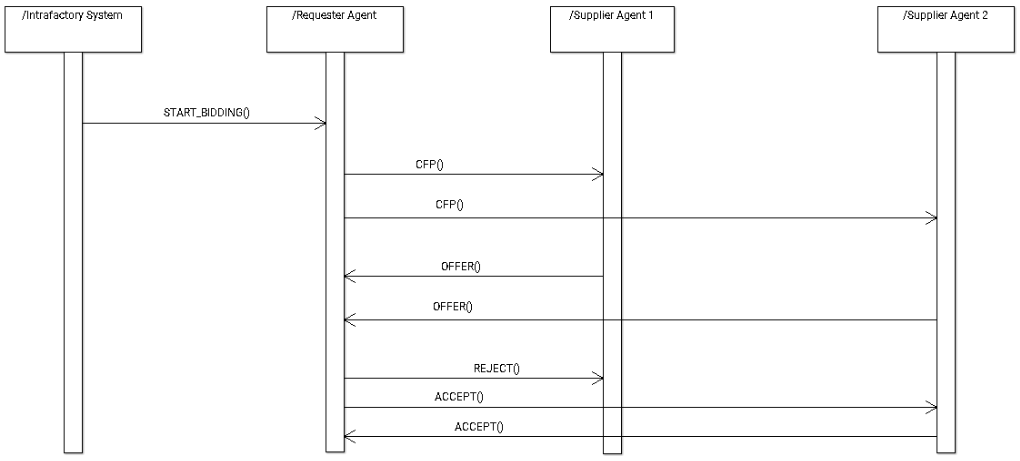


Figure 29. CONTRACT-NET protocol execution.

In the CONTRACT-NET, a requester agent generates at a certain time a new Call for Proposal (CFP) message and sends it to all the agents potentially able to fulfil the request, e.g., identified through a call to the Matchmaker agent. Supplier agents receiving the CFP may either respond with an offer or not. If they respond, the response shall be sent back to the requester within the CFP expiration times, otherwise they will not be considered in the subsequent protocol step. At the CFP expiration deadline, the requester agent collects all received offers and computes for each of them a so-called utility function. The utility function evaluates how good an offer is, with respect to the agent internal evaluation metrics. Computed values are then exploited to rank received offers in decreasing utility values and to select the best one. Upon selection of the winning offer, an ACCEPT message is sent back to the offer originator while all the agents whose offers have been discarded will receive a REJECT message.

To support the CONTRACT-NET protocol, on the requester side, a set of activities has been defined, which specifies formally the steps followed by the Requester agent in handling the negotiation. Figure 30 reports the corresponding UML activity diagram. Together with direct support to negotiation for supply chain formation and execution, the Requester agents might perform a set of parallel activities. These include, among the others, exchanging reputation data with other agents deployed on the marketplace (see Figure 31 and Figure 32) or routing relevant information from the factory (intra-factory side) to a selected set of recipients (inter-factory side), see Figure 33.

Overall, the comprehensive behaviour of a COMPOSITION request agent might be really complex, as shown by the activity diagram of the first design of default Requester agent reported in Figure 34.

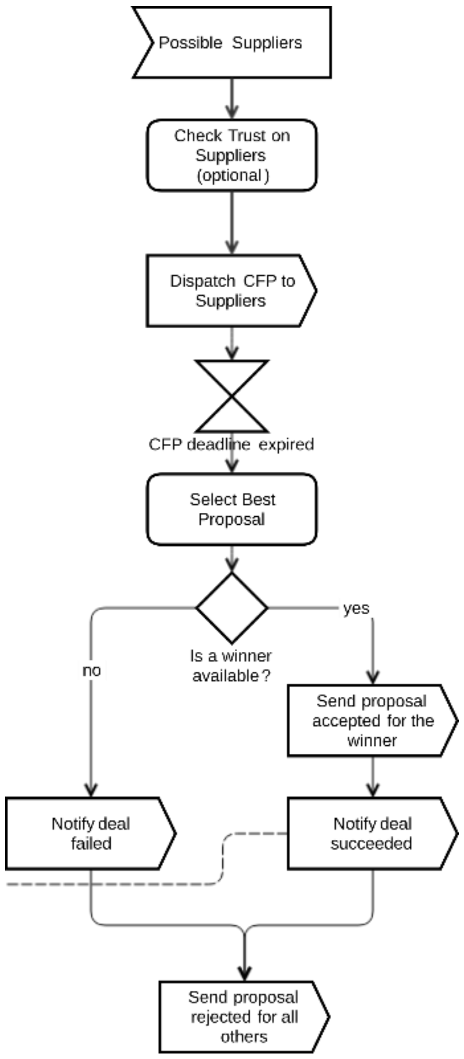

Possible Suppliers
Check Trust on Suppliers (optional)
Dispatch CFP to Suppliers
CFP deadline expired
Select Best Proposal
yes
no
Is a winner available?
Send proposal accepted for the winner
Notify deal failed
Notify deal succeeded
Send proposal rejected for all others

Figure 30. CONTRACT-NET activity diagram at the requester side.

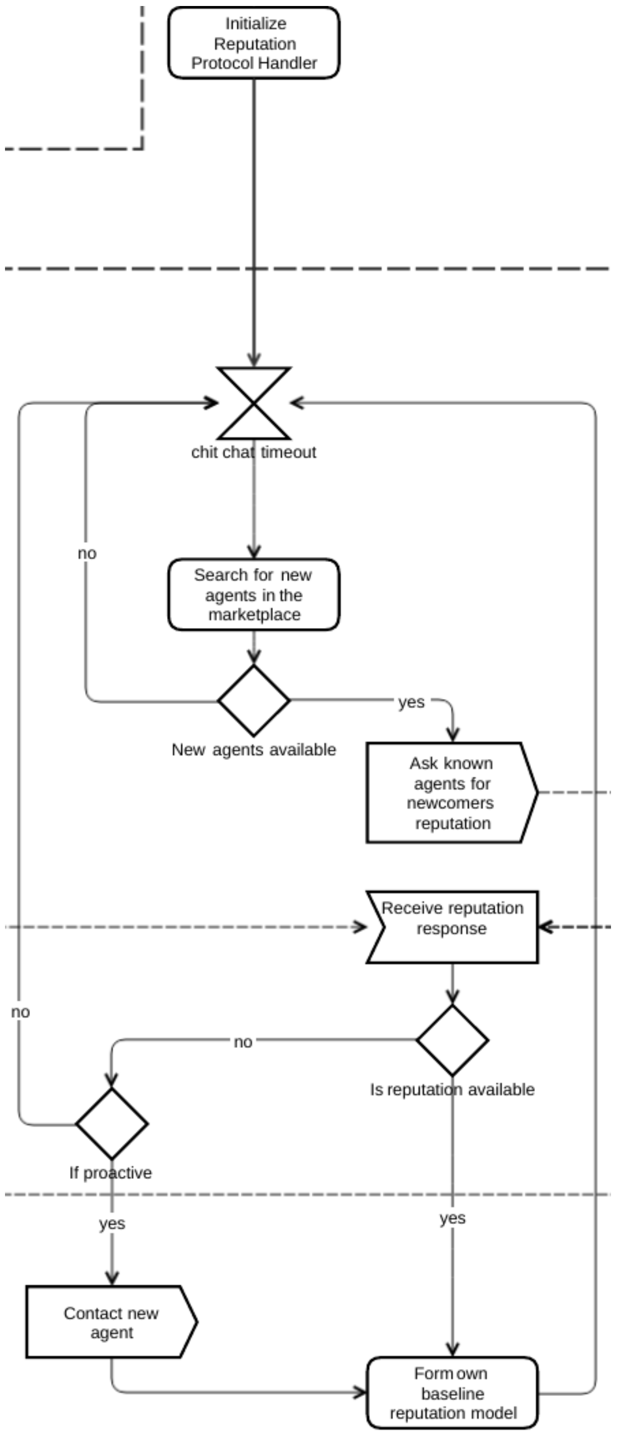

Initialize Reputation Protocol Handler
chit chat timeout
no
Search for new agents in the marketplace
yes
New agents available
Ask known agents for newcomers reputation
Receive reputation response
no
no
Is reputation available
If proactive
yes
yes
Contact new agent
Form own baseline reputation model

Figure 31. Reputation gathering activity.

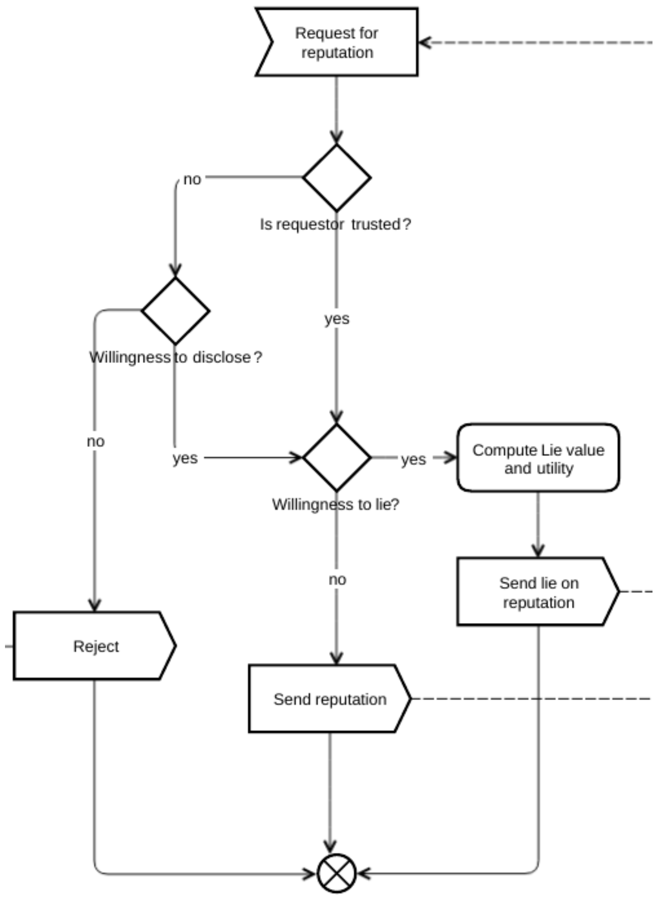


Figure 32. Reputation reporting activity, with possible lies.

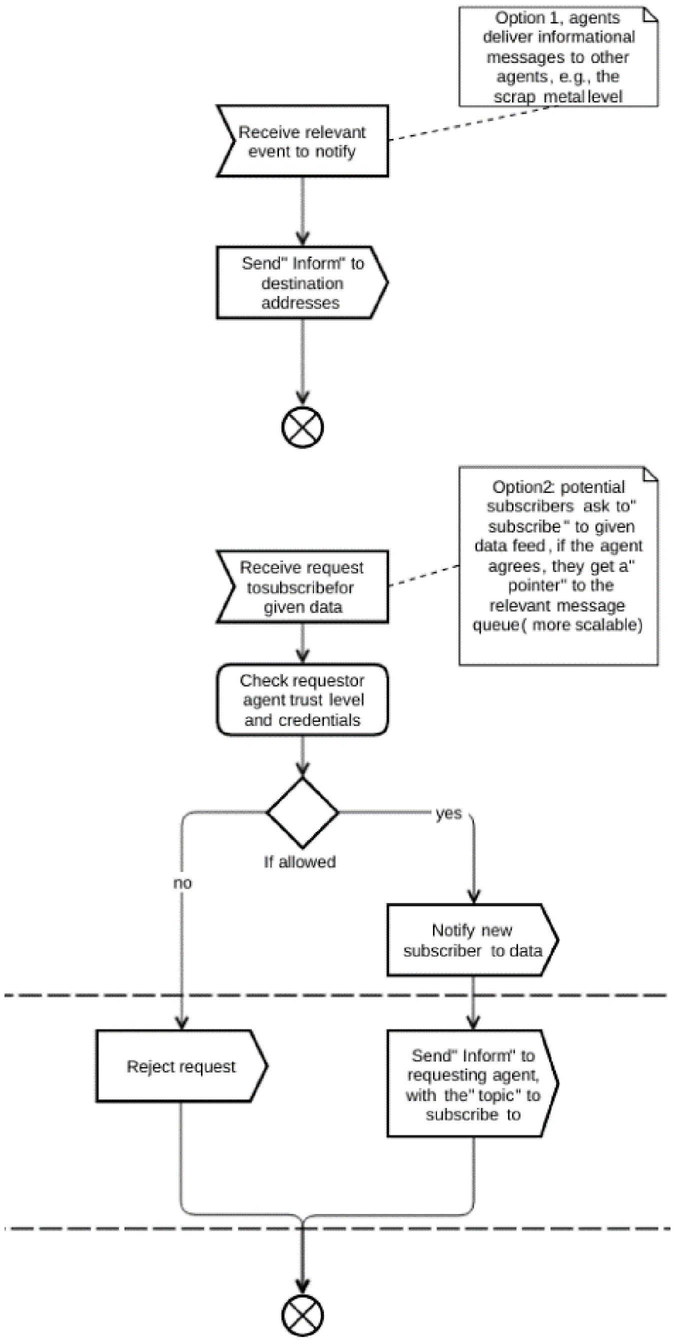

Option 1, agents deliver informational messages to other agents, e.g., the scrap metal level
Receive relevant event to notify
Send" Inform" to destination addresses
Option2: potential subscribers ask to" subscribe" to given data feed, if the agent agrees, they get a" pointer" to the relevant message queue(more scalable)
Receive request tosubscribefor given data
Check requestor agent trust level and credentials
yes
If allowed
no
Notify new subscriber to data
Reject request
Send" Inform" to requesting agent, with the" topic" to subscribe to

Figure 33. Information routing behaviour of the Requester agent.

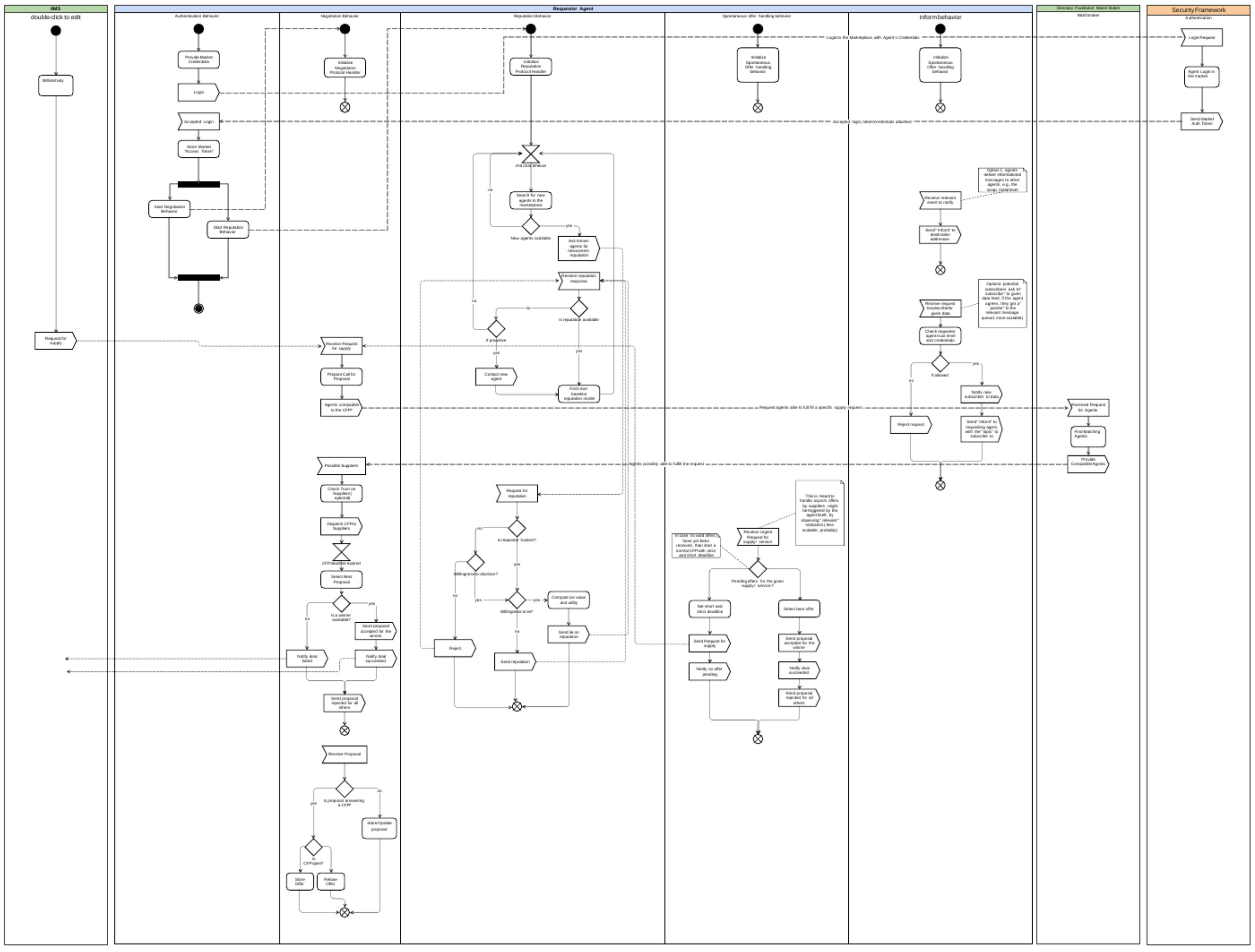


Figure 34. The initial behaviour specification for the COMPOSITION Requester agent.

Supplier Agent

The Supplier agent is the counterpart of the Requester agent on the composition marketplace. It is usually adopted by actual suppliers to respond to supply requests coming from other stakeholders in the marketplace. Factories transforming goods typically employ at least one Requester agent, to get prime goods and one supplier agent to sell intermediate products to other factories.

Supplier agents can in principle support many, different negotiation protocol, however all suppliers in COMPOSITION should at least be able to deal with CONTRACT-NET negotiations (see Figure 35). In parallel, they may carry many other activities, as in the Requester case. Figure 36 reports the initially designed behaviour of a COMPOSITION default supplier, including reputation exchange mechanisms.

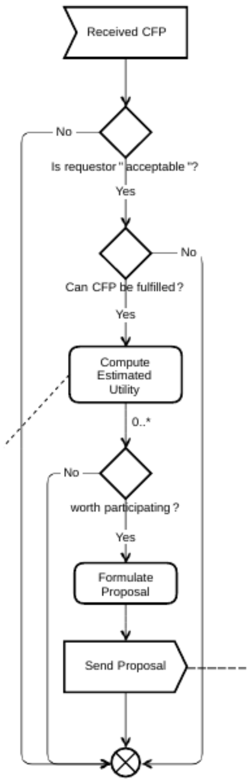


Figure 35. CONTRACT-NET activity diagram for the Supplier Agent.

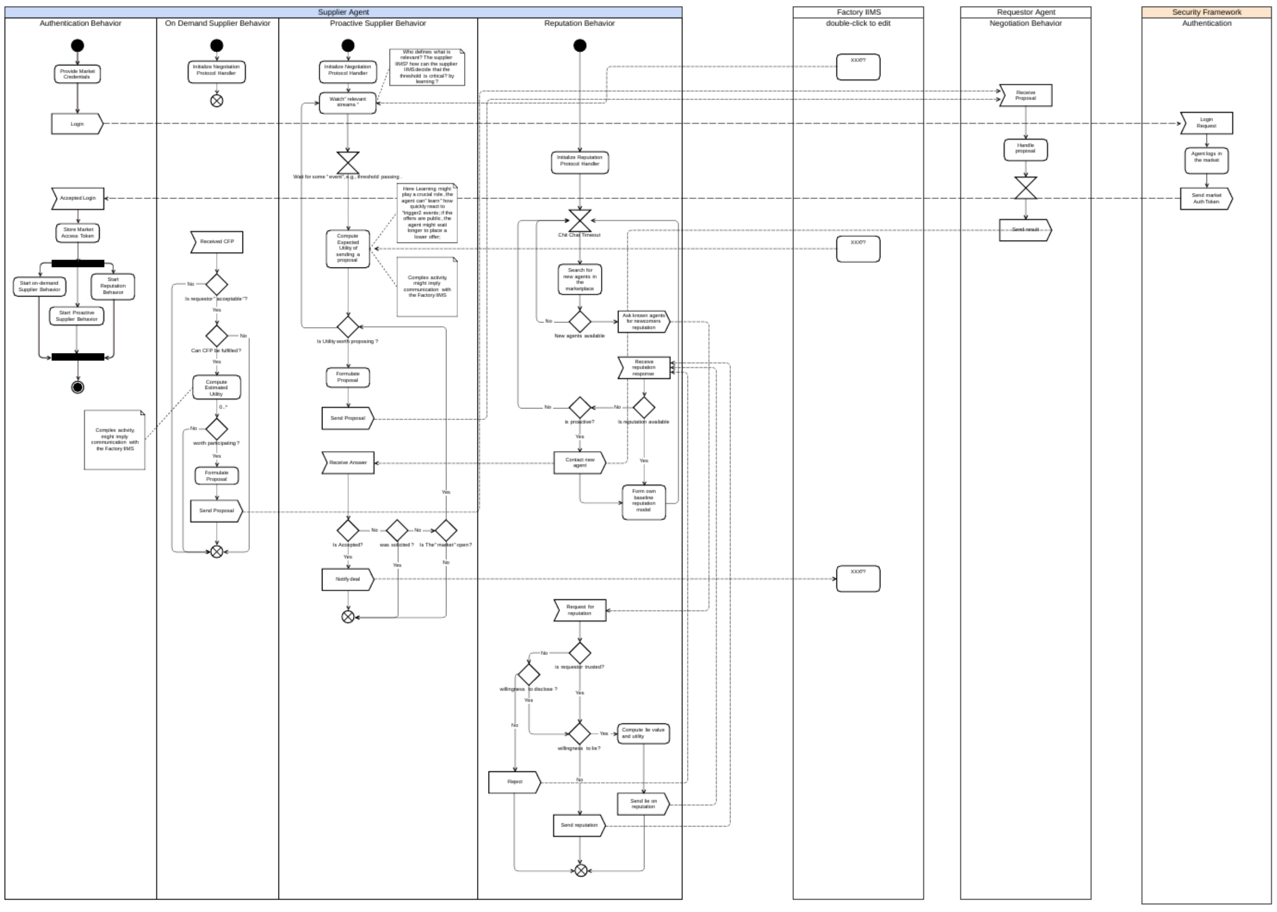

Supplier Agent
Authentication Behavior
On Demand Supplier Behavior
Proactive Supplier Behavior
Reputation Behavior
Factory IIMS
double-click to edit
Requestor Agent
Negotiation Behavior
Security Framework
Authentication
Provide Market Credentials
Login
Accepted Login
Store Market Access Token
Start on-demand Supplier Behavior
Start Reputation Behavior
Start Proactive Supplier Behavior
Initialize Negotiation Protocol Handler
Received CFP
Is requestor " acceptable "?
Can CFP be fulfilled ?
Compute Estimated Utility
worth participating ?
Formulate Proposal
Send Proposal
Complex activity, might imply communication with the Factory IIMS
Watch" relevant streams "
Wait for some " event", e.g., threshold passing
Compute Expected Utility of sending a proposal
Is Utility worth proposing ?
Receive Answer
Is Accepted?
was selected ?
Is The" market" open?
Notify deal
Initialize Reputation Protocol Handler
Chk Chat Timeout
Search for new agents in the marketplace
New agents available
Ask known agents for newcomers reputation
Receive reputation response
is proactive?
Is reputation available
Contact new agent
Form own baseline reputation model
Request for reputation
is requestor trusted?
willingness to disclose ?
willingness to lie?
Compute lie value and utility
Reject
Send lie on reputation
Send reputation
XXX??
Receive Proposal
Handle proposal
Send result
Login Request
Agent logs in the market
Send market Auth Token
Yes
No

Figure 36. The initial behaviour specification for the COMPOSITION Supplier agent.

5.3.10 Marketplace Management

The Marketplace Management component is composed of two main elements respectively named the Marketplace Management Portal and the actual Marketplace Management Services. The former provides a web-based UI for managing a set of Composition Marketplaces, whereas the latter provides the backend, empowering the UI functions and allowing direct configuration and control of the marketplace event broker.

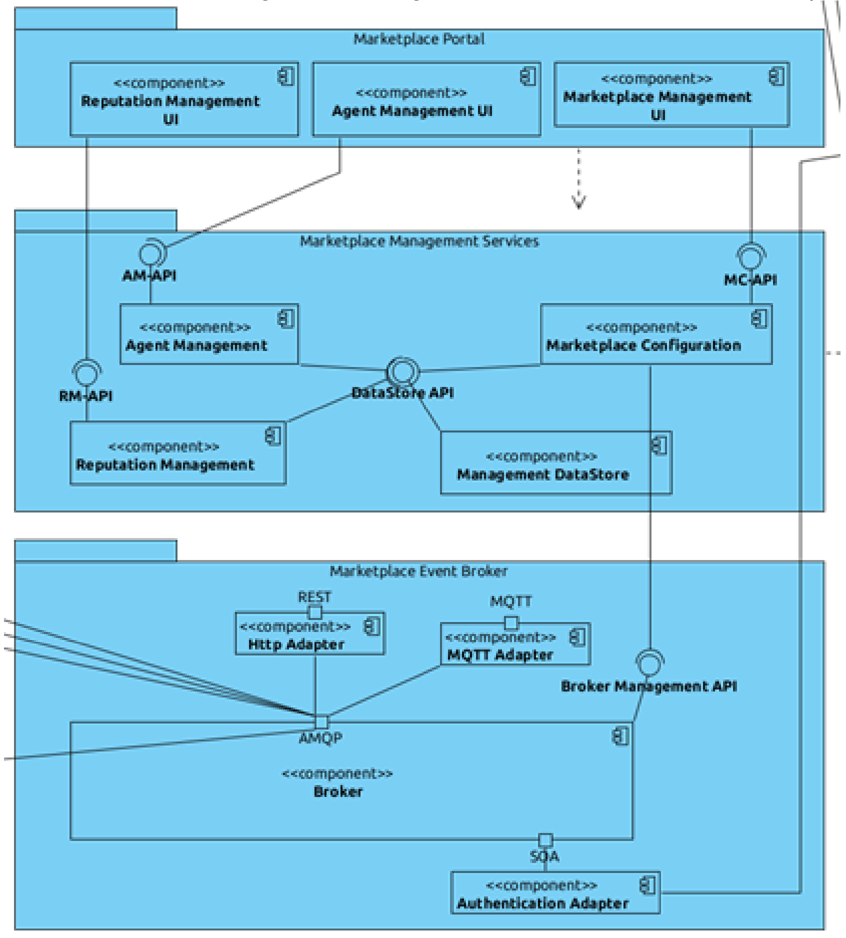

Marketplace Portal
<<component>> Reputation Management UI
<<component>> Agent Management UI
<<component>> Marketplace Management UI
Marketplace Management Services
AM-API
MC-API
<<component>> Agent Management
<<component>> Marketplace Configuration
RM-API
DataStore API
<<component>> Reputation Management
<<component>> Management DataStore
Marketplace Event Broker
REST
MQTT
<<component>> Http Adapter
<<component>> MQTT Adapter
Broker Management API
AMQP
<<component>> Broker
SOA
<<component>> Authentication Adapter

Figure 37. Marketplace Management component.

Marketplace Management components are designed to support many operations that are crucial for the COMPOSITION ecosystem. For example, the Marketplace Management allows stakeholders to join the COMPOSITION marketplaces and to receive the agent credentials and configuration parameters required for joining the corresponding agent containers.

Whenever a new stakeholder joins the COMPOSITION Open Marketplace, some specific data is required on the kind of business pursued by the stakeholder, on the category of goods provided and/or required, etc. This data is leveraged by the Management Portal to support faceted search of available stakeholders, e.g., to conduct preliminary analysis of possible targets of offers and/or possible actors to approach for selling services. Moreover, the same data is propagated to the Matchmaker agent, which can take better decisions about possible matches between supply needs and registered suppliers.

With respect to other COMPOSITION components, the Marketplace Management is not yet completely specified in this iteration of the architecture. The inner components of both the Marketplace Management Portal and the corresponding backend service are not yet settled. They might vary, with respect to Figure 37, depending on the formalization of user needs and interactions currently under development. Nevertheless, a solid set of use cases describing the main functions that need to be provided at the end user level has already been defined, and is reported in Figure 38.

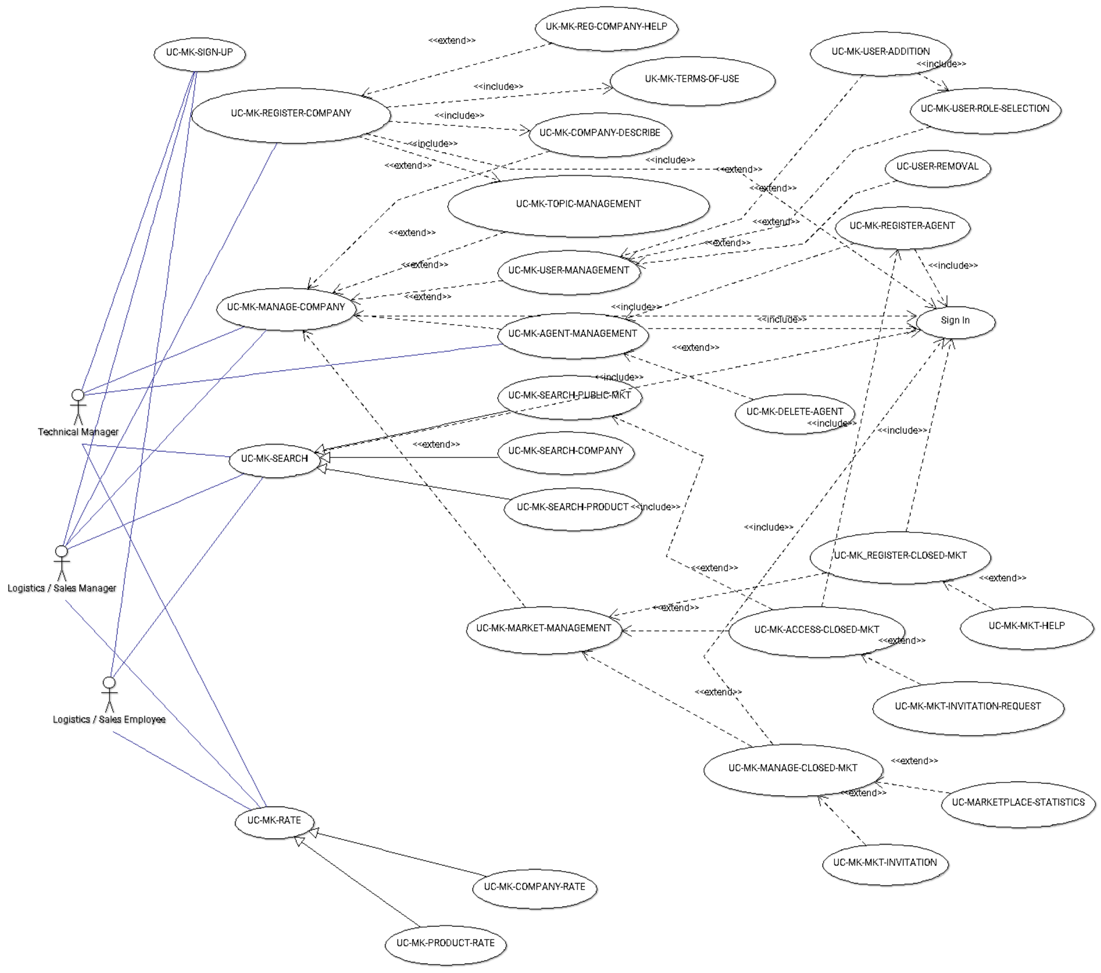


Figure 38. The Marketplace Management use cases.

These use cases are currently driving the complete specification of the management components and of the interactions between such components and the other software actors defined in the marketplace. More specifically, we are currently in the process of finalizing the initial Marketplace Management Portal mock-up to confirm design decisions and then drive the identification of needed sub-components and requirements at both the UI and the backend services level. For the sake of clarity, the currently envisioned interactions between end users and the portal, and corresponding UI wireframes are reported in Figure 39.

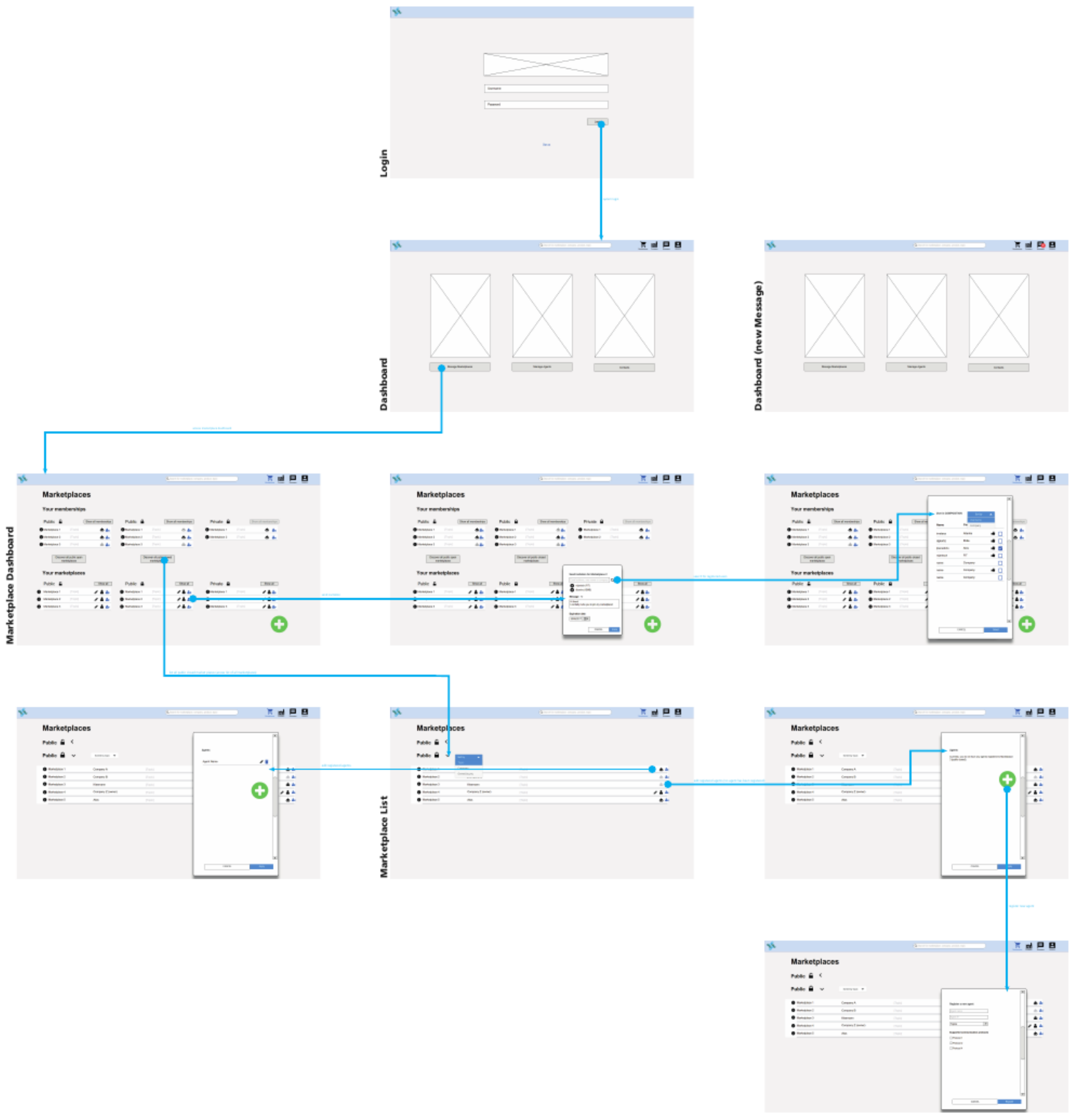


Figure 39. Initial Marketplace Portal mock-up and navigation.

5.3.11 Security Framework

5.3.11.1 **Introduction**

The Security Framework implements the security core mechanisms aiming to ensure the security, confidentiality, integrity and availability of the managed information for all authorized Composition stakeholders. Below there is an overview of the current identified components that will conform the Security Framework.

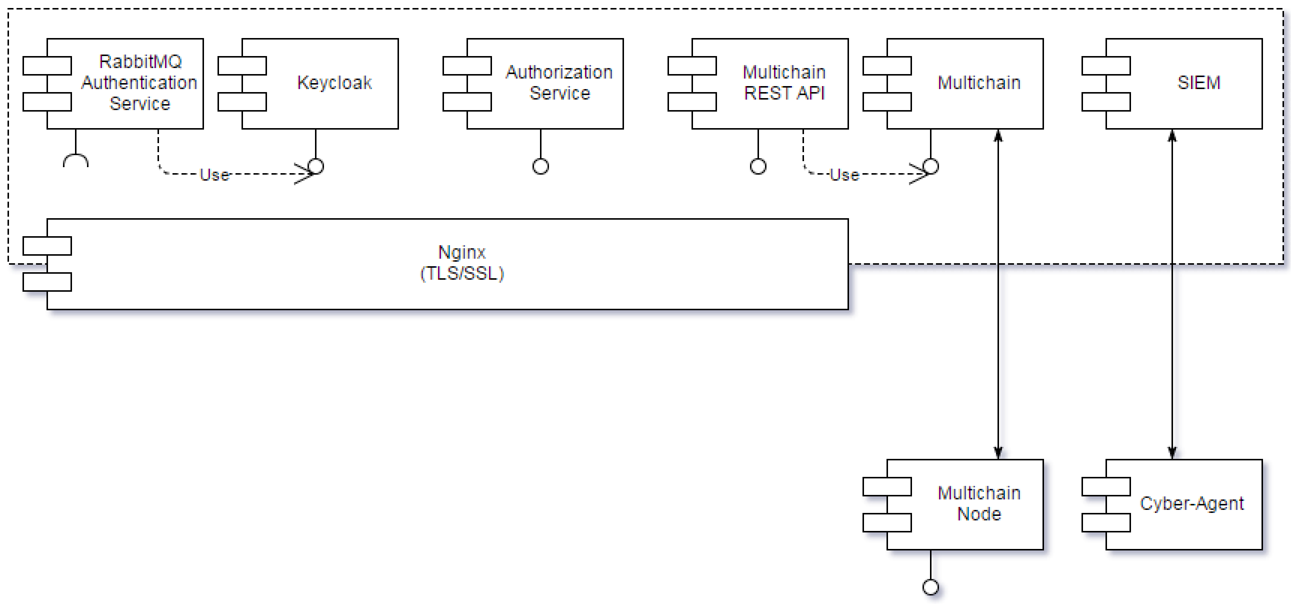


Figure 40. Components of the Security Framework.

The current components in the Security have been grouped in for main categories, each of them focusing on different security tasks:

1. Authentication:
    a. Keycloak: Open source Identity and Access Management solution.
    b. RabbitMQ Authentication Service: Service that relays in Keycloak and Authorization Service to override built-in RabbitMQ authentication mechanisms.
2. Authorization:
    a. Authorization Service: Atos tool based on XACML3.0 that provides authorization and privacy access control to resources
3. Log-Trust-IPR
    a. Multichain: Blockchain based on Bitcoin with added functionalities.
    b. Multichain REST API: Will provide functionalities based on blockchain.
4. Cybersecurity:
    a. SIEM: Atos tool that provides the capabilities of a Security Information and Event Management (SIEM) solution with the advantage of being able of handling large volumes of data and raise security alerts from a business perspective.
    b. Cyber-Agents: These components are responsible to catch the events that later will be analysed by the SIEM.

In front of all web applications and services, in this case Keycloak, RabbitMQ Authentication Service, Authorization Service and Multichain REST API; Nginx will be used as reverse proxy configured for using TLS/SSL.

The following sections will provide details on each of the components and their categories.

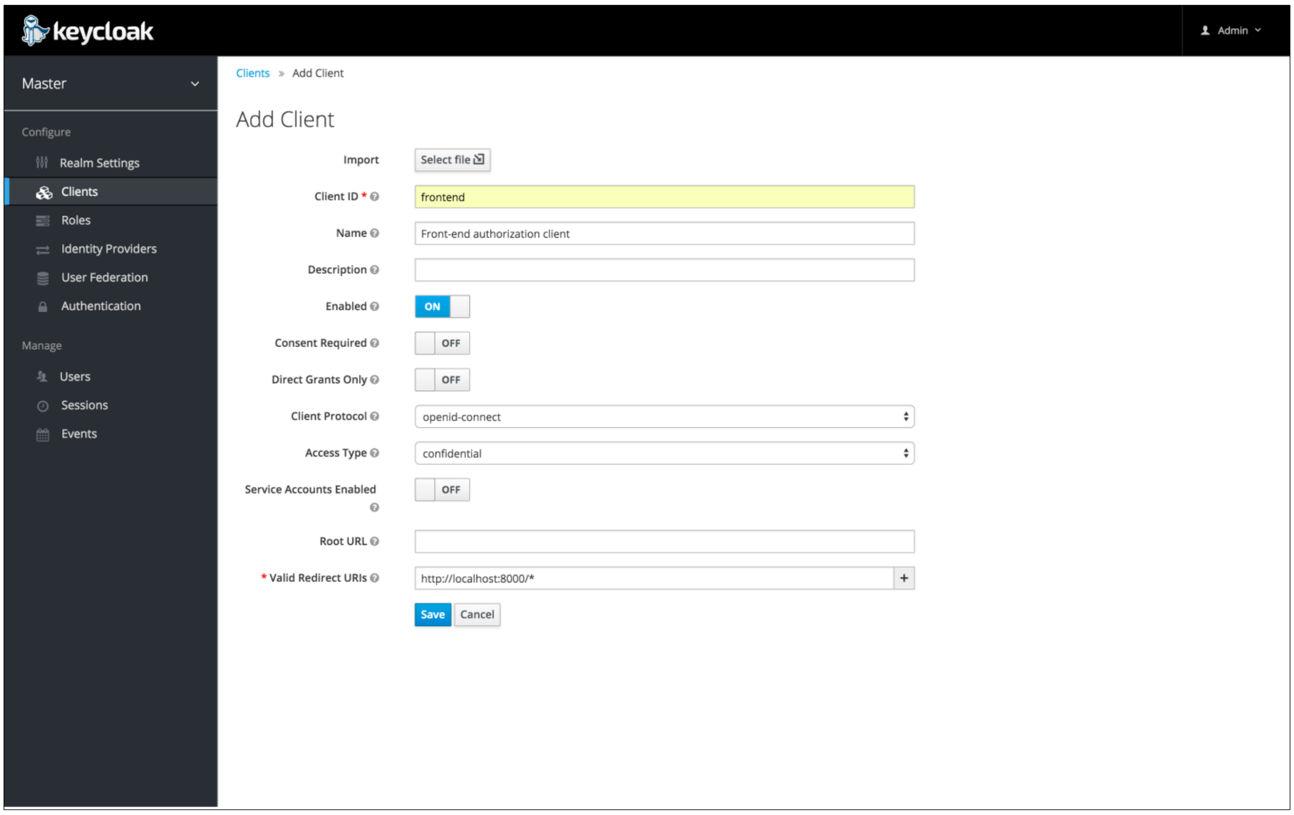
5.3.11.2 **Authentication**

The components in this category are the responsible of providing the authentication mechanisms for users, applications, services and devices.

5.3.11.2.1 Keycloak

Keycloak[16] is an open source Identity and Access Management solution. Some of the features are:

- Single-Sign On: Authenticate on Keycloak rather on different applications. One single login will allow access to multiple applications and/o services.
- Identity Brokering and Social Login: Enable login with social networks such as Google, Facebook, Twitter and GitHub.
- User Federation: Connect directly to LDAP and Active Directory servers.
- Standard Protocols: OpenID Connect OAuth 2.0 and SAML.



Add Client

Import	Select file
Client ID *	frontend
Name	Front-end authorization client
Description	
Enabled	ON
Consent Required	OFF
Direct Grants Only	OFF
Client Protocol	openid-connect
Access Type	confidential
Service Accounts Enabled	OFF
Root URL	
* Valid Redirect URIs	http://localhost:8000/*

Figure 41. Keycloak administration interface.

5.3.11.2.2 RabbitMQ Authentication Service

This component will implement the needed interfaces to override RabbitMQ built-in authentication and authorization engine and it will make use of Keycloak and Authorization Service for authentication and authorization instead.

5.3.11.3 Authorization

This category is responsible of all aspects about authorization mechanisms. Only one component has been identified under this category, the Authorization Service, which is a tool based on XACML 3.0

5.3.11.3.1 Authorization Service

The Authorization Service is a tool based on XACML 3.0 that provides authorization and privacy access control to resources. It provides two different functionalities:

[16] http://www.keycloak.org

- Policy management: Ability to manage policies. This means generating, storing, removing and modifying policies.
- Policy enforcement: Ability to enforce that a given access request for a specific resource fulfils the requirements of the policies applicable to the resource trying to be accessed.

5.3.11.4 **Log, Trust and IPR**

This category is responsible of the component that will contribute to the protection of IPR, the creation of trust and the audit trail for manufacturing and supply chain data.

5.3.11.4.1 Multichain

Multichain[17] is a private blockchain platform based on Bitcoin enhanced with added functionalities like managed permissions and data streams. Data streams are separately permissioned entities in the blockchain optimized for logging data in key-value pairs, as opposed to transactions involving assets (e.g. bitcoins). Several blockchains may be run in parallel, with managed permissions and several data streams per chain. The ability to run multichain in a consortium with a controlled set of block validators ("miners") negates the need for proof-of-work mining, making the generation of blocks, and consequently transaction validation, much faster.

5.3.11.5 **Cyber-Security**

The components on this category focuses on the analysis of the cyber security in collaborative manufacturing and logistics ecosystems, identifying the variety of attacks (such as abuse of privileges, denial of access…) that could affect and be more relevant for the availability and reliability of the platform and infrastructure and potential remediation measures to mitigate their effects.

5.3.11.5.1 SIEM

SIEM provides the capabilities of a Security Information and Event Management (SIEM) solution with the advantage of being able of handling large volumes of data and raise security alerts from a business perspective thanks to the analysis and event processing in a Storm cluster. The main SIEM functionalities can be summarized in the following points:

- Real-time collection and analysis of security events.
- Prioritization, filtering and normalization of the data gathered from different sources.
- Consolidation and correlation of the security events to carry out a risk assessment and generation of alarms and reports.

5.3.11.5.2 Cyber-Agents

These components are responsible to catch the security events and transmit them to SIEM to be analysed. They are installed on the systems that need to be secured and their configuration may differ from one installation to another depending on the events to be monitored.

5.3.11.6 **Nginx**

Nginx[18] is a free and open-source web server software, which can also be used as a reverse proxy, load balancer and HTTP cache. Currently it´s only envisioned to be used as a reverse proxy in front of the web applications and services providing an additional security layer. It will also provide Transport Layer Security (TLS) encryption capabilities to all the applications and services behind it.

[17] http://www.multichain.com
[18] https://nginx.org/

5.4 Information View

5.4.1 Data Models

5.4.1.1 Overview

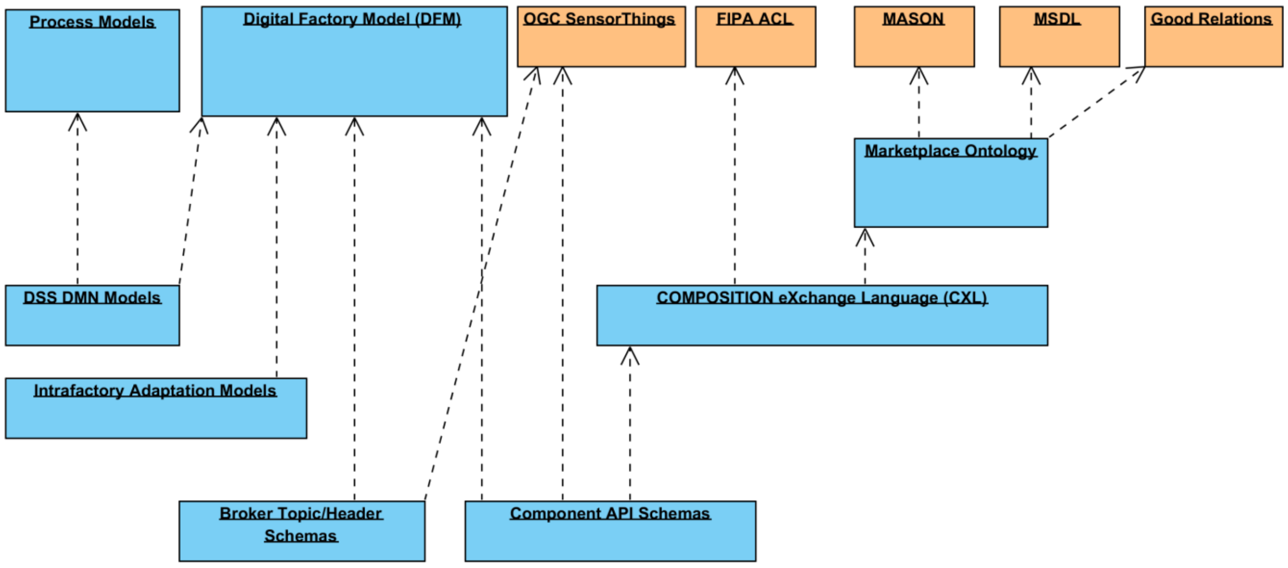

Process Models
Digital Factory Model (DFM)
OGC SensorThings
FIPA ACL
MASON
MSDL
Good Relations
Marketplace Ontology
DSS DMN Models
COMPOSITION eXchange Language (CXL)
Intrafactory Adaptation Models
Broker Topic/Header Schemas
Component API Schemas

Figure 42. Dependencies of data models used in the system.

The Digital Factory Model (DFM) contains both types and instances of the intra-factory components, e.g. production lines, products and sensors. This information will be used by e.g. the BMS to connect the physical sensors to the DFM instances and propagate this information to the LinkSmart middleware to identity the sensor data. The information is also used to build the topics in the message broker by which other components can subscribe to live data. The broker topic schemas have not yet been defined.

The process models describe the production process, linking information in the system to the process context used in the Decision Support System.

The OGC SensorThings Data Model is the used for system-generated data, e.g. data in the IIMS that has passed through the LinkSmart middleware and is exposed in inter-component communication will use the OGC SensorThings Data Model, with links to the DFM types and instances.

The inter-factory domain is modelled in the Marketplace Ontology, and expressed in the Composition Exchange Language (CXL) used for agent communication.

5.4.1.2 Process Models

The goal of process models in COMPOSITION is to use common formats or standards to describe the production process. With such process models, process-oriented monitoring is made possible. By definition, process-oriented monitoring is a monitoring strategy that builds correlation measured values from sensors to a specific process procedure and a specific product instance in a production line, so that those sensor values could be further analysed within context. For example, with process-oriented monitoring it is possible to investigate how much energy is consumed while producing a specific PCB panel in solder printing. Process-oriented monitoring opens up possibilities for different big data analysing strategy, such as real-time abnormalities detection, product quality prediction etc.

The process models of the industrial processes will follow the Business Process Model and Notation (BPMN) standard. BPMN is a standard for business process modelling that provides a graphical notation for specifying business processes in a Business Process Diagram (BPD), based on a flowcharting technique very similar to activity diagrams from Unified Modelling Language (UML). Besides the graphical representation, the standard also specifies the XML schema for describing BPMN, which makes it easy to communicate between different systems.

Different elements from BPMN are adopted to model manufacture process in process models. First of all, production procedures will be modelled as *activities* in BPMN. The procedure will be modelled according to its property, such as if it is a manual task, or if it is an automatic task finished by machines. Between activities there are intermediate message events, which matches to the corresponding sensor signals. These message events act as a transition between activities, which is triggered only when the matching sensor signals is received. With this structure, we can ensure that the BPMN virtual process is always synchronized with the real product process. Gateways are also utilized to model conditional forks during manufacture process.

During runtime, the process models will be instantiated and managed by a BPMN engine, such as the Activiti BPMN Engine. One can imagine the relation between the process model and an instantiated process as the relation between class and object in object oriented programming. Typically, each product on the line is represented by one instantiation of the model, tracking its current activity. This strategy enables a real-time matching between sensor values and the correspondent workpiece in the production line.
Figure 43 shows an example of a process model describing the production line of BSL. Notice that the process consists of many activities (rectangles in the diagram), each of which represents one step in production, such as laser marking PCB, screen printing solder, inspecting solder, etc. Between activities are intermediate message events, which will only be triggered by the matching sensor signals. Exclusive gateways are also used to model choices in process, such as if panel fails to pass *Inspect solder* test, it will be rejected to conveyor belt for either manual touch-up or touch-up in machine.

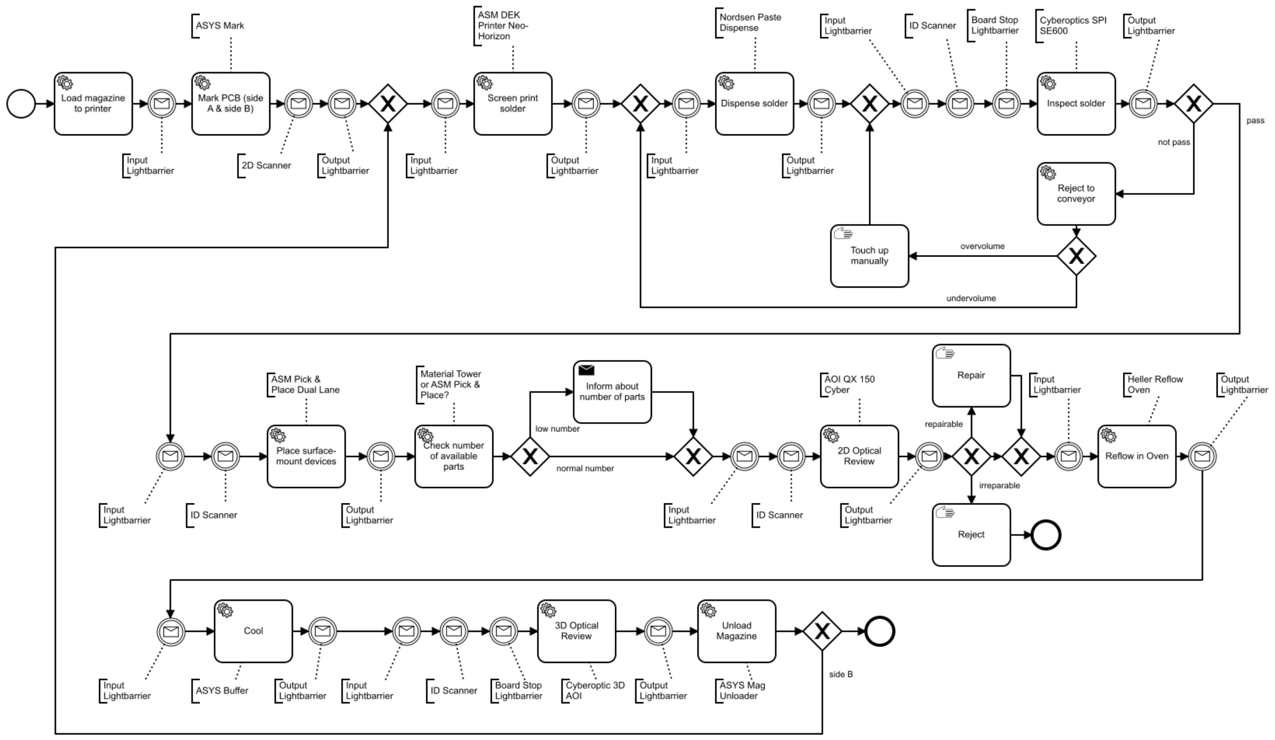


Figure 43. Initial BPMN diagram of BSL production line.

5.4.1.3 **Digital Factory Model**

COMPOSITION Digital Factory Model (DFM) aims at the digitalization of industrial aspects. It is responsible for modelling and storing both static and dynamic (real-time) information acquired from various heterogeneous sources in the factory in a common format. It offers effective use of resources and knowledge and provides all necessary means for the direct manipulation of the overall Digital Factory concept as a single, but complex entity. DFM is based on XML syntax and popular standards such as ANSI ISA-95/B2MML, gbXML, BPMN, and x3d and provides a high level of simplicity, extensibility, interoperability and openness. It is implemented as XML schema. The DFM instances will be XML files. The structure of DFM is separated in two basic components:

1. *Information Model* which describes static information: buildings, equipment, assets, actors, procedures, installed sensors and business processes related to the project.
2. *Events* describe the real-time dynamic information related to the factory including events, measurements, alerts etc.

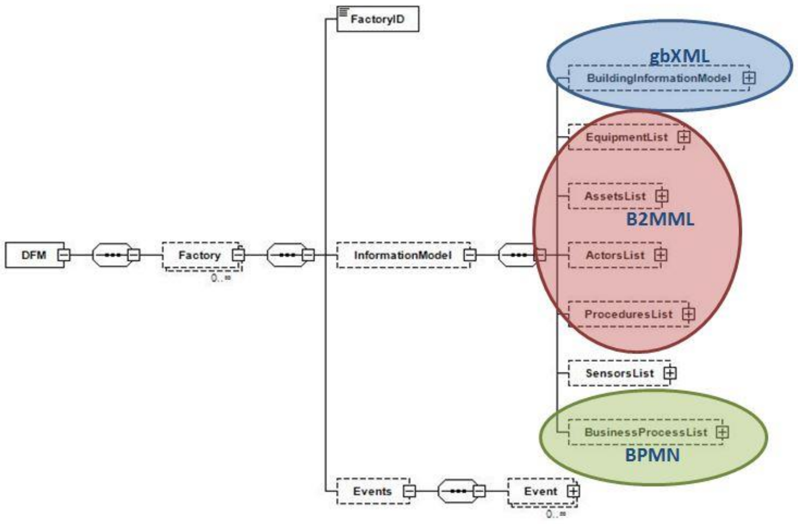


Figure 44. High level structure of COMPOSITION DFM XML schema.

Main Input(s):

- LinkSmart middleware

Main Output(s):

- LinkSmart middleware
- Simulation and Forecasting Tool
- Decision Support System (DSS)
- UI components

The DFM will interact with LinkSmart middleware in the cases a real-time data exchange will be needed. LinkSmart will collect all the information (e.g. physical equipment's real-time data) from the factory and store it to DFM. On the other hand, the LinkSmart middleware retrieves stored data from DFM and provide them to other IIMS components. Also, DFM will feed Simulation and forecasting tool and DSS with data. In order to create simulations, predictions and recommendations both simulation tool and DSS will be use information providing by DFM instances.

5.4.1.4 **Composition eXchange Language**

Agents communicate through messages encoded in a dedicated language named Composition eXchange Language (CXL). Rather than defining yet another agent communication language, the consortium decided to stick to existing standards and to extend them wherever needed. CXL has therefore been designed as a dialect of the well-known FIPA ACL language specification[19], with a dedicated syntax ("codec" in the FIPA jargon) and with reference to a well-defined set of ontologies for representing the message payload data.

A CXL message is composed of: (a) an almost fixed set of parameters, identifying the message purpose, sender and language, (b) a variable payload whose content depends on the message type, and typically is encoded according to an explicitly defined ontology. Table 4 depicts the exact fields defined in CXL. Each of them has a 1-to-1 mapping to the corresponding FIPA ACL message parameter.

Table 4. CXL message structure.

Message Parameter	Description
Act	Identifies the type of communicative act represented by the message, corresponds to the FIPA ACL performative parameter, and assumes the same possible values.
Sender	The originator of the message (an agent)
Receiver	A list of recipients of the message (i.e., a set of agents)
reply-to	The agent to which replies for this message shall be sent
Language	The language in which the content is encoded
Encoding	The encoding of the content language expressions (mime type)
Ontology	This parameter identifies the set of ontologies that describe the classes and relationships that are valid expressions of the message payload. Such ontologies are provided as a list of URIs.
Protocol	Identifies the agent-communication protocol to which this message adheres (e.g., CONTRACT-NET for basic negotiations).
Content	The actual payload of the message
conversation-id	Provides an identifier for the sequence of communicative acts (messages) that together form a conversation
Reply-with	Provides an expression that the message recipient shall include in the answer, exploiting the in-reply-to field. This allows following a conversation when multiple dialogues occur simultaneously.
In-reply-to	Denotes an expression that references and earlier action to which this message is a reply
Reply-by	Identifies the latest date/time by which the sender agent would like to receive a reply. Replies received later that the time specified by this parameter can be freely ignored.

CXL messages are encoded in JSON, supporting easy manipulation in a multitude of programming languages. Additional encodings, e.g., byte-efficient, representations may be considered for particular deployments where the message size may affect the performance of the COMPOSITION marketplace. The CXL JSON coding has been formalized by exploiting a specific JSON schema, reported below.

```
{
```

[19] http://www.fipa.org/specs/fipa00061/SC00061G.pdf

```
 "description":"The JSON syntax specification of the COMPOSITION CXL language, mainly focus on the
message envelope",
 "type":"object",
 "properties":{
  "act":{
   "type":"string",
   "enum":["accept-proposal","agree","cancel","cfp","confirm","disconfirm","failure","inform","not-
understood","query-if","query-ref","refuse","reject-proposal","request","request-when","request-
whenever","subscribe","inform-if","inform-ref","proxy","propagate","propose"]
  },
  "sender":{
   "type":"object",
   "description":"the message originator",
   "properties":{
    "name":{
     "type":"string"
    },
    "addresses":{
     "type":"array",
     "items": {
      "type":"string"
     }
    },
    "user-defined":{
     "type":"object"
    }
   }
  },
  "receiver":{
   "type":"array",
   "description":"The set of recipients for this message",
   "items":{
    "type":"object",
    "description":"the message recipient",
    "properties":{
     "name":{
      "type":"string"
     },
     "addresses":{
      "type":"array",
      "items": {
       "type":"string"
      }
     },
     "user-defined":{
      "type":"object"
     }
    }
   }
  },
  "reply-to":{
   "type":"object",
   "description":"The agent to which replies for this message shall be sent",
   "properties":{
    "name":{
     "type":"string"
    },
    "addresses":{
     "type":"array",
     "items": {
      "type":"string"
     }
    },
    "user-defined":{
     "type":"object"
    }
   }
  },
  "language":{
   "type":"string",
   "description":"the language used for encoding the message content"
  },
  "encoding":{
   "type":"string",
   "description":"the specific encoding used for language expressions, typically a mime type"
  },
  "ontology":{
   "type":"array",
   "description":"The set of ontologies defining the primitives that are valid within the message content",
   "items":{
    "type":"string",
    "format":"url"
   }
  },
  "protocol":{
   "type":"string",
   "description":"Identifies the agent communication protocol to which the message adheres"
  },
  "content":{
   "type":"object",
   "description":"The actual payload of the message"
  },
  "conversation-id":{
   "type":"string",
   "description":"Provides an identifier for the sequence of communicative acts (messages) that together form a conversation"
  },
  "reply-with":{
   "type":"string",
   "description":"Provides an expression that the message recipient shall include in the answer, exploiting the in-reply-to field. This allows following a conversation when multiple dialogues occur simultaneously."
  },
  "in-reply-to":{
   "type":"string",
   "description":"Denotes an expression that references and earlier action to which this message is a reply"
  },
  "reply-by":{
   "type":"string",
   "format":"date-time"
  }
 },
 "additionalProperties":false
}
```

This design choice supports syntax validation of messages exchanged between agents, thus preventing elaboration of messages that are not conforming to the language primitives.

While the CXL-JSON format specifies the envelope of agent messages, the message content depends on both the communication protocol adopted by the agents and on the reference ontology specified through the ontology parameter. Ontology expressions in the content field shall be encoded in JSON-LD for the message to be validated according to the CXL-JSON schema.

At the current stage of development 2 main vocabularies, i.e., ontologies, have been identified and catalogued for use in the composition CXL. The first vocabulary is called FIPA-Agent-Management and it stems from the FIPA ACL specifications. The FIPA-Agent-Management ontology defines the primitives used by agents to register themselves and interact with the AMS agent. The second vocabulary, instead, is a full-blown ontology defined in the context of the COMPOSITION project and thoroughly described in Section 5.4.1.5.

To exemplify how messages are encoded in CXL, and how difficult is handling the corresponding syntax, let us consider the CONTRACT-NET exchanges depicted in Figure 35. At the beginning of a CONTRACT-NET conversation, the Requester agent sends a CFP message to a selected set of suppliers. A sample CFP message encoded in CXL-JSON is reported below.

```
{
 "act":"cfp",
 "sender":{
  "name":"RequesterAgent1",
  "address":["amqp://reqagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"cd9cf485-c7cc-42f8-b209-1ac9c6faa2e6"
  }
 },
 "receiver":[{
  "name":"SupplierAgent1",
  "address":["amqp://suppagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"428a96d7-ad9a-499f-a13e-cf59e02ba469"
  }
 }],
 "reply-to":{
  "name":"RequesterAgent1",
  "address":["amqp://reqagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"cd9cf485-c7cc-42f8-b209-1ac9c6faa2e6"
  }
 },
 "language":"x-cxl-json",
 "encoding":"x-application/json",
 "ontology":[
  "http://www.composition-project.eu/cxl/ontology#"
 ],
 "protocol":"x-contract-net",
 "content":{
  "@context":"http://www.composition-project.eu/cxl/ontology#",
  "offering":{
    "hasBusinessFunction":"Sell",
    "includes":"scrap-metal-collection"
  },
  "scrap-metal-collection":{
   "minimumWeigth":"10",
   "weightUnit":"t",
   "collectionMean":"truck"
  }
 },
 "conversation-id":"cd9cf485-c7xk-42f8-b209-1ac9c6faa2e6",
 "reply-with":"propose-scrap-1",
 "reply-by":"2017-05-30T00:00:00+01:00"
}
```

Suppliers, may either provide back an offer or not. In case an offer is provided, a possible CXL encoding of such an offer could be as follows.

```
{
 "act":"propose",
 "sender":{
  "name":"SupplierAgent1",
  "address":["amqp://suppagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"428a96d7-ad9a-499f-a13e-cf59e02ba469"
  }
 },
 "receiver":[{
  "name":"RequesterAgent1",
  "address":["amqp://reqagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"cd9cf485-c7cc-42f8-b209-1ac9c6faa2e6"
  }
 }],
 "reply-to":{
  "name":"SupplierAgent1",
  "address":["amqp://suppagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"428a96d7-ad9a-499f-a13e-cf59e02ba469"
  }
 },
 "language":"x-cxl-json",
 "encoding":"x-application/json",
 "ontology":[
  "http://www.composition-project.eu/cxl/ontology#"
 ],
 "protocol":"x-contract-net",
 "content":{
  "@context":"http://www.composition-project.eu/cxl/ontology#",
  "offering":{
    "hasBusinessFunction":"Sell",
    "includes":"scrap-metal-collection",
    "hasPriceSpecification":{
     "hasCurrency":"USD",
     "hasCurrencyValue":"65.5",
     "validThrough":"2017-11-30T23:59:59"
    }
  },
  "scrap-metal-collection":{
   "minimumWeigth":"5",
   "weightUnit":"t",
   "collectionMean":"truck"
  }
 },
 "conversation-id":"cd9cf485-c7xk-42f8-b209-1ac9c6faa2e6",
 "in-reply-to":"propose-scrap-1"
}
```

Finally, when the negotiation deadline expires the winner receives an ACCEPT message, while the other involved suppliers will receive a REJECT message. Corresponding CXL messages are reported below, in the same order (i.e.,ACCEPT followed by REJECT).

```
{
 "act":"accept-proposal",
 "sender":{
  "name":"RequesterAgent1",
  "address":["amqp://reqagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"cd9cf485-c7cc-42f8-b209-1ac9c6faa2e6"
  }
 },
 "receiver":[{
  "name":"SupplierAgent1",
  "address":["amqp://suppagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"428a96d7-ad9a-499f-a13e-cf59e02ba469"
  }
 }],
 "reply-to":{
  "name":"RequesterAgent1",
  "address":["amqp://reqagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"cd9cf485-c7cc-42f8-b209-1ac9c6faa2e6"
  }
 },
 "language":"x-cxl-json",
 "encoding":"x-application/json",
 "ontology":[
  "http://www.composition-project.eu/cxl/ontology#"
 ],
 "protocol":"x-contract-net",
 "content":{
  "@context":"http://www.composition-project.eu/cxl/ontology#",
  "offering":{
    "hasBusinessFunction":"Sell",
    "includes":"scrap-metal-collection",
    "hasPriceSpecification":{
     "hasCurrency":"USD",
     "hasCurrencyValue":"65.5",
     "validThrough":"2017-11-30T23:59:59"
    }
  },
  "scrap-metal-collection":{
   "minimumWeigth":"5",
   "maximumWeight":"15",
   "weightUnit":"t",
   "collectionMean":"truck"
  }
 },
 "conversation-id":"cd9cf485-c7xk-42f8-b209-1ac9c6faa2e6",
 "in-reply-to":"propose-scrap-1",
 "reply-with":"propose-scrap-1-confirmed"
}
```

```
{
 "act":"reject-proposal",
 "sender":{
  "name":"RequesterAgent1",
  "address":["amqp://reqagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"cd9cf485-c7cc-42f8-b209-1ac9c6faa2e6"
  }
 },
 "receiver":[{
  "name":"SupplierAgent1",
  "address":["amqp://suppagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"428a96d7-ad9a-499f-k734-cf59e02ba469"
  }
 }],
 "reply-to":{
  "name":"RequesterAgent1",
  "address":["amqp://reqagent1@market1/test"],
  "user-defined":{
   "composition-agent-uuid":"cd9cf485-c7cc-42f8-b209-1ac9c6faa2e6"
  }
 },
 "language":"x-cxl-json",
 "encoding":"x-application/json",
 "ontology":[
  "http://www.composition-project.eu/cxl/ontology#"
 ],
 "protocol":"x-contract-net",
 "content":{
  "@context":"http://www.composition-project.eu/cxl/ontology#",
  "offering":{
    "hasBusinessFunction":"Sell",
    "includes":"scrap-metal-collection",
    "hasPriceSpecification":{
     "hasCurrency":"USD",
     "hasCurrencyValue":"65.5",
     "validThrough":"2017-11-30T23:59:59"
    }
  },
  "scrap-metal-collection":{
   "minimumWeigth":"5",
   "maximumWeight":"5",
   "weightUnit":"t",
   "collectionMean":"truck"
  }
 },
 "conversation-id":"cd9cf485-c7xk-42f8-b209-1ac9c6faa2e6",
 "in-reply-to":"propose-scrap-2"
}
```

As future step, the CXL will be extensively documented and additional supported ontologies will be defined with the relative syntax and primitives, with particular focus on data-forward and reputation exchange communications currently under design.

5.4.1.5 **Marketplace Ontology**

COMPOSITION's collaborative manufacturing services ontology and language supports flexible specification and execution of manufacturing collaboration schemes by describing both relations between business entities and manufacturing services/ resources. It is developed in OWL format and well-known ontologies for e-commerce and manufacturing domain modelling are imported. More precisely Manufacturing Service Description Language or MSDL (Ameri, 2006) and MAnufacturing's Semantics Ontology or MASON (Lemaignan, 2006) are imported into COMPOSITION ontology and provide sufficient description of manufacturing services, capabilities and resources(machine-tools, tools, human resources, geographic resources like plants and workshops). In order to describe relations between business entities the GoodRelations Language Reference (GoodRelations, 2017) ontology is also imported to COMPOSITION one. GoodRelations ontology provides all the necessary classes and properties for the description of offers and requests among business entities.

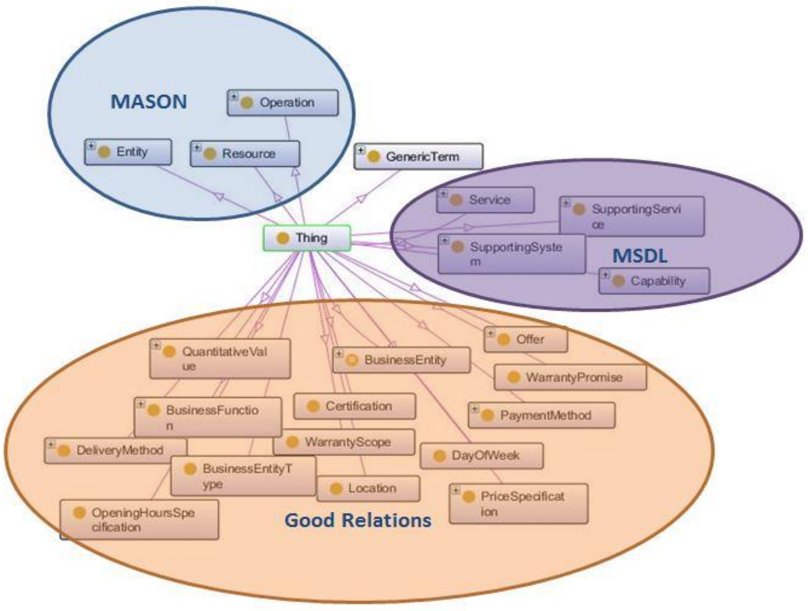


Figure 45. High level COMPOSITION Ontology class diagram.

COMPOSITION Ontology will be used as the knowledge base for the Marketplace. No user interface will be needed. The Marketplace's agents will be able to export/import data from/to ontology by applying queries (e.g. SPARQL queries) using the Ontology Query API. COMPOSITION Matchmaker will infer knowledge by applying semantic rules in the ontology and then it will be able to match offers and requests for manufacturing services, raw materials and products between the involved business entities.

5.4.1.6 **OGC SensorThings**

The SensorThings API[20] is an OGC[21] standard specification, part of the OGC Sensor Web Enablement standards[22]. This standard has been selected as the generic representation of data managed by the COMPOSITION system (see Figure 46 for the SensorThings data model). It is also used in the LinkSmart platform.[23]

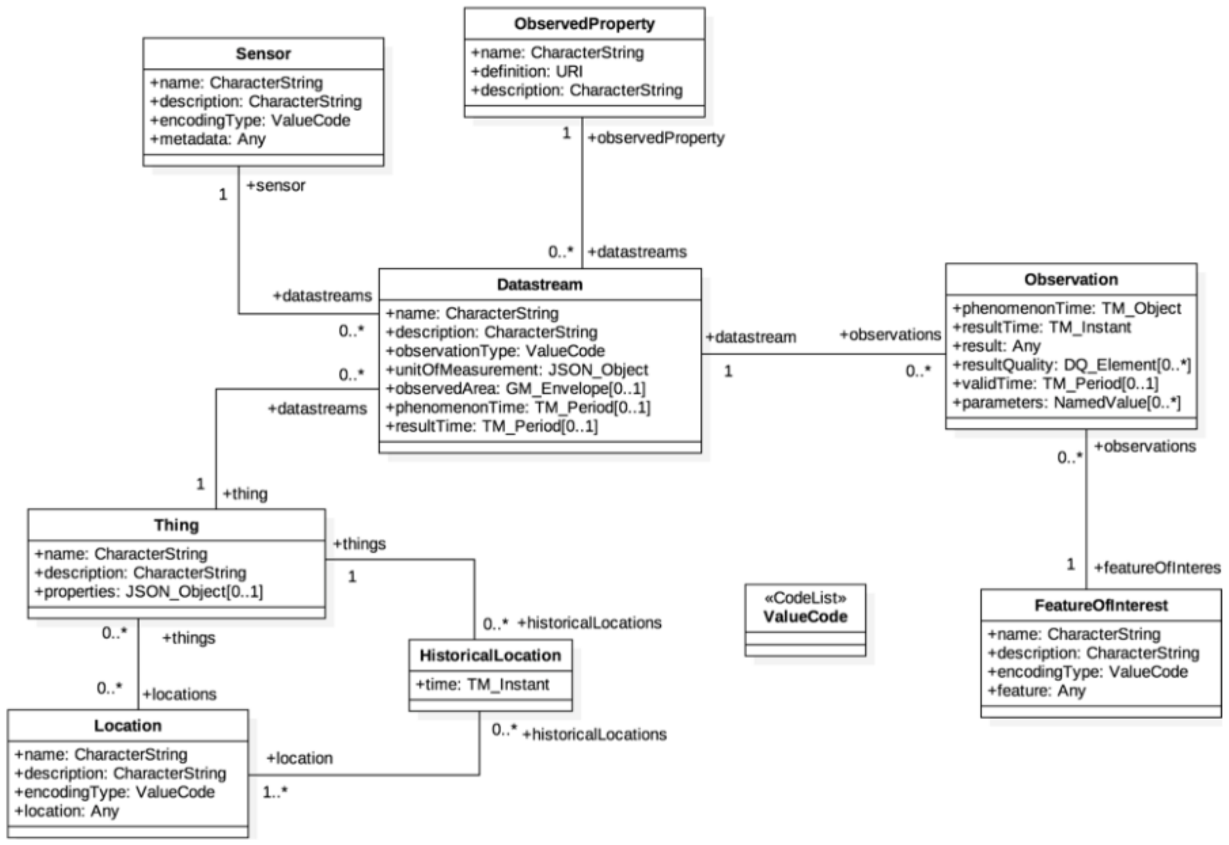


Figure 46. OGC SensorThings Data Model.

The OGC SensorThings API consists of the Sensing and Tasking profiles.

The Sensing profile allows IoT devices and applications to CREATE, READ, UPDATE, and DELETE (i.e., HTTP POST, GET, PATCH, and DELETE) IoT data and metadata in a *Thing* service. Managing and retrieving observations and metadata from IoT sensor systems is one of the most common use cases. As a result, the Sensing profile is designed based on the ISO/OGC Observation and Measurement (O&M) model (OGC and ISO 19156:2011).

The key to the model is that an *Observation* is modelled as an act that produces a result whose value is an estimation of a property of the observation target or *FeatureOfInterest*. An Observation instance is classified by its event time (e.g., *resultTime* and *phenomenonTime*), FeatureOfInterest, ObservedProperty, and the procedure used (often corresponding to a *Sensor*). Things are also modeled in the SensorThings API, together with the historical set of their geographical positions

[20] http://docs.opengeospatial.org/is/15-078r6/15-078r6.html
[21] http://www.opengeospatial.org/
[22] http://www.opengeospatial.org/ogc/markets-technologies/swe
[23] https://linksmart.eu/redmine/projects/iot-data-processing-agent/wiki/Usage_IoT_Data-Processing_Agent_

More specifically, in the Sensing profile, a *Thing* has *Location*s and *HistoricalLocation*s. It can also have multiple *Datastream*s associated. A Datastream is a collection of Observations grouped by the same *ObservedProperty* and *Sensor*. An Observation is an event performed by a Sensor that produces a result whose value is an estimate of an ObservedProperty of the FeatureOfInterest.

Following subsections better detail the single data model entries.

5.4.1.6.1 Thing

The OGC SensorThings API follows the ITU-T definition, i.e., with regard to the Internet of Things, a thing is an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks (Y.2060, 2012).

5.4.1.6.2 Location

The Location entity locates the Thing or the Things it is associated with. A Thing's Location entity is defined as the last known location of the Thing.

5.4.1.6.3 HistoricalLocation

A Thing's HistoricalLocation entity set provides the current (i.e. last known) and previous locations of the Thing with their time.

5.4.1.6.4 Datastream

A Datastream groups a collection of Observations and the Observations in a Datastream measure the same ObservedProperty and are produced by the same Sensor.

5.4.1.6.5 Sensor

A Sensor is an instrument that observes a property or phenomenon with the goal of producing an estimate of the value of the property.

5.4.1.6.6 ObservedProperty

An ObservedProperty specifies the phenomenon of an Observation.

5.4.1.6.7 Observation

An Observation is an act of measuring or otherwise determining the value of a property (ISO19156, 2011).

5.4.1.6.8 FeatureOfInterest

An Observation results in a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the FeatureOfInterest of the Observation (ISO19156, 2011). In the context of the Internet of Things, many Observations' FeatureOfInterest can be the Location of the Thing. For example, the FeatureOfInterest of a wifi-connect thermostat can be the Location of the thermostat (i.e. the living room where the thermostat is located in). In the case of remote sensing, the FeatureOfInterest can be the geographical area or volume that is being sensed.

5.4.2 Data Persistence

The Deep Learning Toolkit needs to have historical data available to train the artificial neural networks, although this only has to be available as unstructured bulk data, without query capabilities. The Intrafactory Adaptation Layer has built-in storage for unprocessed shop-floor level data which can be used in this capacity and LinkSmart also provides capabilities for storing historical observation data.

Data persistence will to a significant extent be handled internal to the components and exposed through the component interfaces, in the case of component-specific data. However, there will still be a need to record and query both shop-floor data and data generated by the COMPOSITION system, common to all components. The Decision Support System and the Simulation and Forecasting Tool both need access to structured historical data generated by the system, with query capabilities.

As mentioned above, this data is transported in OGC SensorThings format inside the system. The OGC SensorThings API specifies how to store and retrieve data conforming to this model using a REST API or MQTT. Any implementation of this API would thus be a natural choice of persistence mechanism. There are open source implementations of the OGC SensorThings API available, e.g. GOST[24] or SensorThingsServer[25], where observation data can be stored. These can be deployed in Docker containers and may be used interchangeably in the system.

5.4.3 Data Flow

5.4.3.1 **Deep Learning and Big Data Analysis**

As described in section 5.3.5, the Deep Learning Toolkit is deployed after the first three phases that take place offline and relies on conspicuous historical datasets provided by the end user partners: the training, validation and test datasets specific for each learning task are extracted from randomized data. During the offline training activity, for each learning task, the most appropriate models are selected and configured for optimal predictions. This process consists in iterating multiple times through model training and model validation steps. During a training step, the optimal parameters are determined for a given combination of model hyper-parameters, by minimizing prediction error over the training set. Subsequently, during the validation phase, the trained model (as specified by its hyper-parameters and parameters values) is validated against the validation dataset. In the end, the trained model with best validation score is selected and its performances are assessed one more time against the test dataset, in the testing phase, to ensure proper generalization. The model is then deployed inside the deep learning toolkit for online prediction and continuous learning. The continuous online training process is represented as an activity diagram in the figure below.

[24] https://github.com/FraunhoferIOSB/SensorThingsServer
[25] https://github.com/Geodan/gost

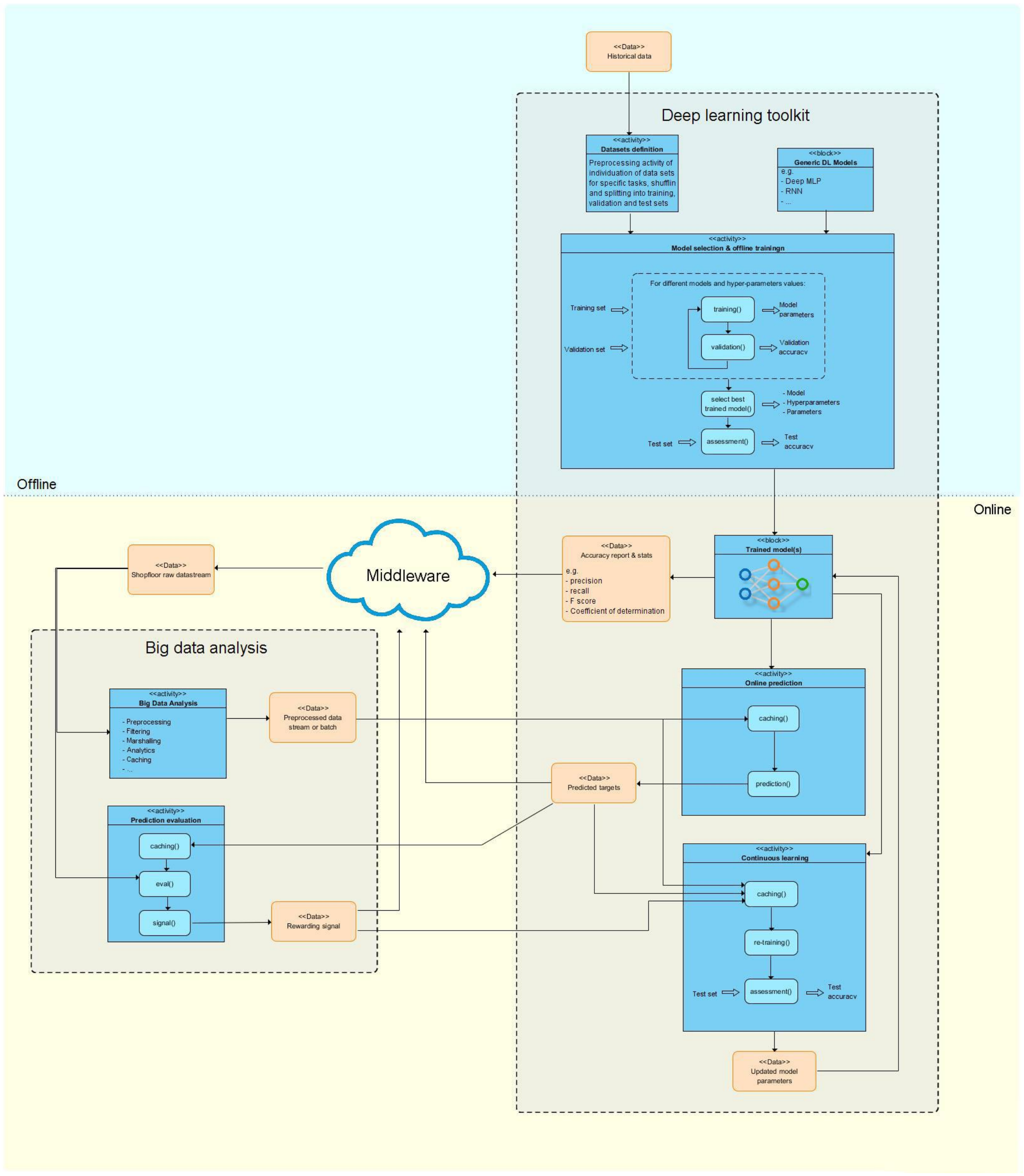


Figure 47. Deep Learning Toolkit: online continuous learning model as activity diagram.

The online interactions and data flows between modules are described in the following. Raw data streams from the shopfloor (IoT, sensors, machine data) are distributed through the middleware. The Big Data Analysis module is responsible for preprocessing these data streams, which for instance may encompass functionalities such as aggregation, marshalling, filtering, analyzing and caching. The output is a preprocessed data stream (or a periodic emission of batches of buffered data) which is take as an input in the prediction activity of the Deep Learning Toolkit module. Here – depending on specific prediction task – the data might be further cached, finally data are processed by the trained model that output the predicted targets. Considered the high relevance to decision making, the predicted targets are broadcasted over the middleware for other IIMS modules to use or present to the user. Additionally, the Big Data Analysis module can use these data as well, aiming to evaluate the actual accuracy Deep Learning Toolkit by comparing previous predictions with current shop floor data. This leads Big Data Analysis module to asynchronously send rewarding signals to the Deep Learning Toolkit which use them as additional input to its continuous learning activity. The latter aims at refining the training of the internal model exploiting data, predictions and feedbacks from the Big Data Analysis module. This process results in updating the model so to achieve better accuracy in pair with new report and statistics about the model itself. This model is published on the middleware as well. The training process is represented as an activity diagram in the figure below.

The data flow of both processes are summarized by the component model in Figure 47, the upper part referring to offline training and the lower one to online forecasting and retraining.

5.4.3.2 **Marketplace**

Several kinds of information are exchanged at the marketplace level, some directly regarding supply-chain formation and execution and some related to other activities such as reputation exchange, data transfer, etc. While each kind of information flow deserves a dedicated description, reported in the following, it is important to highlight that all such flows are based on common basic assumptions.

In particular, every information exchanged between software entities connected to the marketplace takes form of a CXL message stream flowing from one originator agent to one or more recipient agents. Such a design choice preserves independence of data flows from the actual agent implementation. Furthermore, it does not impose a priori any particular stateful and/or stateless assumption such as, for example in REST-based communication.

5.4.3.2.1 Agent-to-agent communication

Since agents are the company representatives on the marketplace, they have full control on information crossing the company boundaries, and full control on (expected) recipients of such an information, thus limiting the possibility for unknown subscribers to eavesdrop “confidential” agent-to-agent conversations (see Section 6.1 for the corresponding security measures devised in the project).

From a very high-level standpoint, information exchanges over the marketplace can be of three main types:

- Supply-chain formation and execution data
- Monitoring data transferred to selected listeners, e.g., for supporting offer pushing scenarios,
- Reputation data exchanged between agents

These main flows involve several communication acts, between the different marketplace components.

Supply chain formation and execution

Supply chain formation, and execution, may be deployed differently depending on the actual complexity of the involved agents, on the adopted discovery mechanisms and on the adopted negotiation protocol. In this initial release of the COMPOSITION architecture, we envision 2 main cases respectively related to fully fledged agents, able to autonomously match and rank requests and offers, and to “dumb” agents (e.g., base or default implementations) requiring “help” to the platform Matchmaker agent to effectively discriminate between acceptable and not acceptable proposals.

In the former case, i.e., when agents have enough intelligence to handle the matching and selection of offers, the high-level information flow between “core” entities is deployed as in Figure 48.

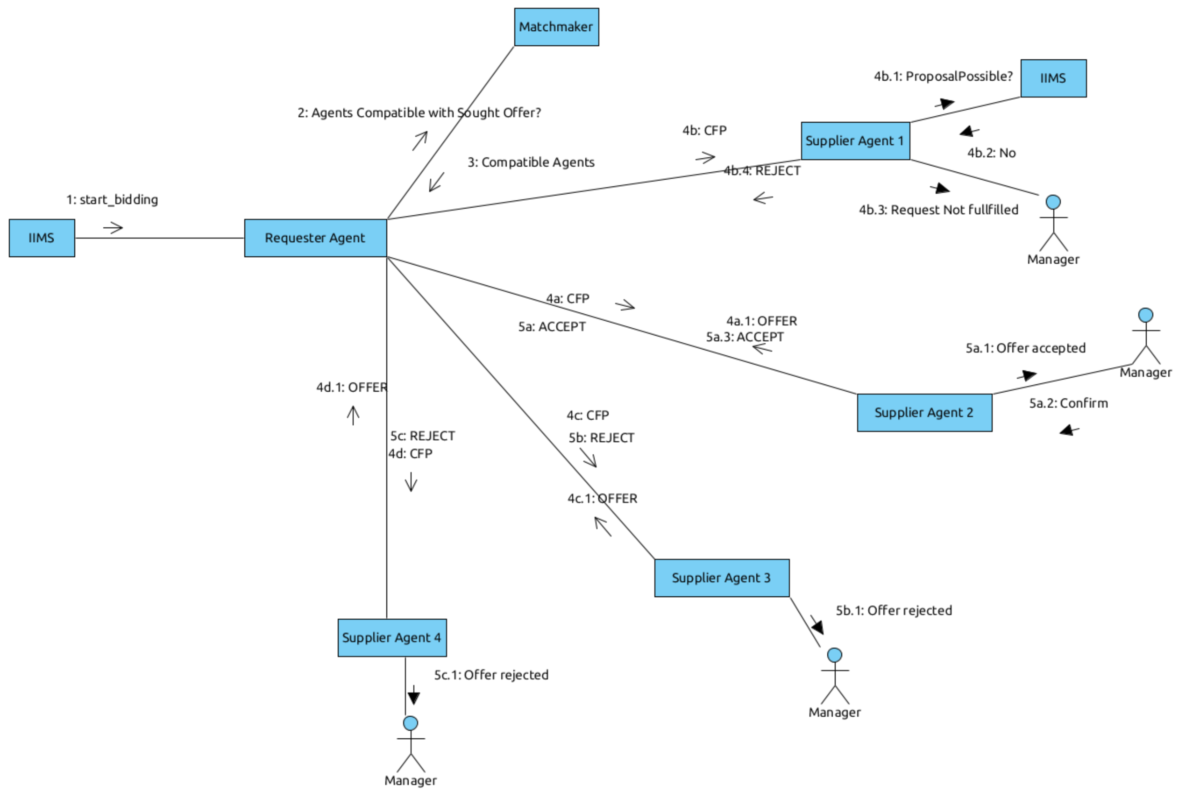


Figure 48. High-level data flow (UML Interaction Diagram) between marketplace actors in a simple contract-net exchange.

Following the message sequence, at some point in time an event in the intra-factory IIMS system generates a request to start a bidding process for procuring a certain kind of material (1). The Requester Agent contacts the Matchmaker agent to identify the most suited set of stakeholder agents currently active on the market (2). Logic inference and rule-based reasoning on the business descriptions associated to agents currently active on the market is performed and a list of candidates for negotiation is provided back (3).Once retrieved the possible candidates, the requester agent creates and delivers Call For Proposal (CFP) messages (4a,4b,4c,4d) to the agents identified by the matchmaker. Among these agents, some may belong to a company that is currently not producing the required material, e.g., due to a programmed maintenance. The corresponding agent thus refuses to participate in the negotiation and replies with a REJECT CXL message (4b.4), while at the same time informs the company commercial manager that an order was missed due to a downtime (4b.3). The other agents are instead able to participate to the negotiation and respond to the CFP with a valid CXL OFFER message (4a.1, 4c.1, 4d.1). Once the bid deadline expires, the Requester agent evaluates the offers and finds that the one coming from Supplier Agent 2 is maximizing its internal utility function (evaluating how well a deal matches the IIMS-specified needs). It reacts to this situation by accepting the offer from Supplier Agent 2 (5a) and by rejecting all the others (5c, 5b). In order to finalize the negotiation the Supplier Agent 2 asks for explicit confirmation from a human counterpart (manager) in company 2 (5a.1) and upon approval (5a.2), agrees to finalize the current deal (5a.3).

It is important to notice that for the sake of clearness, several intermediate passages have been omitted. In particular, each message exchange relies on the marketplace message broker for the actual delivery and exploits the marketplace security framework for checking message provenance. If we consider a single agent-to-agent exchange, e.g., the one identified by the 4a, 5a sequence, the actual information exchanged between agents, and their human counterparts) can be detailed as in the sequence diagram reported in Figure 49.

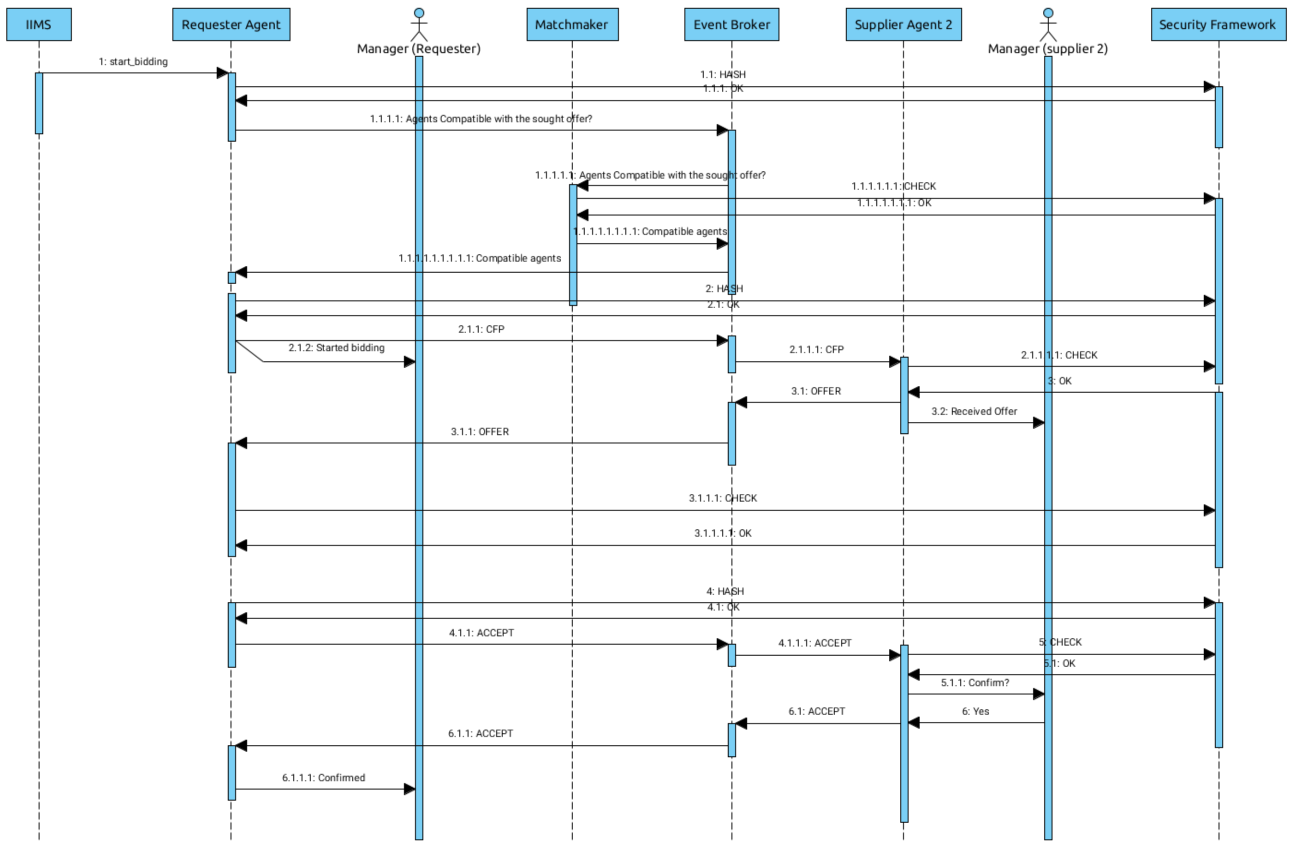


Figure 49. Sequence diagram showing the information flow between to agents during a successful negotiation.

The second type of negotiation analysed in this first period of the project refers to cases where agents have not enough computational power, or intelligence, to neither discriminate between acceptable and not acceptable proposals nor rank such proposals in terms of achieved utility values. For similar scenarios, the matchmaker agent can be exploited as external matcher (see Figure 50) and agents can safely delegate evaluation of offers to such an agent.

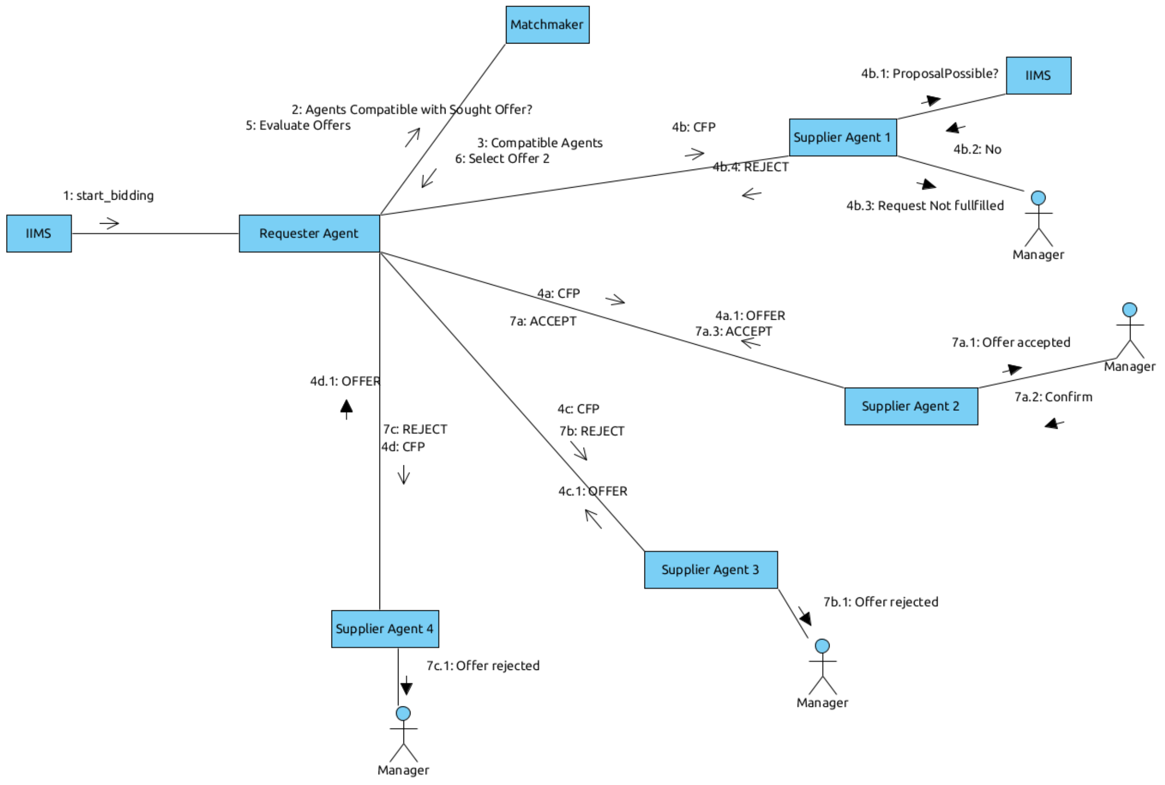


Figure 50. Data flow for "dumb" or default agents.

Monitoring data transfer

During the marketplace operation, agents might require other agents to provide certain streams of “live” data for performing activities that may be part of a previously negotiated deal or that might trigger new deals in a push-modality. In COMPOSITION, for example, scrap metal management scenarios are adopting this paradigm in the context of Closed Marketplaces, where waste management companies on the marketplace are “trusted” by the market owner[26]. In these cases, the data owner agent is responsible to route required information to the right recipient agents, through dedicated CXL messages (of type “inform”). Recipients, on the other hand, shall require subscription to data streams needed for performing their own tasks, e.g., monitoring the scrap container levels and timely offer to collect wasted metal. Such a subscription process exploits again a set of CXL messages, with a performative identifier equal to “subscribe”. The corresponding sequence diagram is reported in Figure 51.

[26] Typically corresponding to the Requester in a pull interaction such as the one firstly discussed.

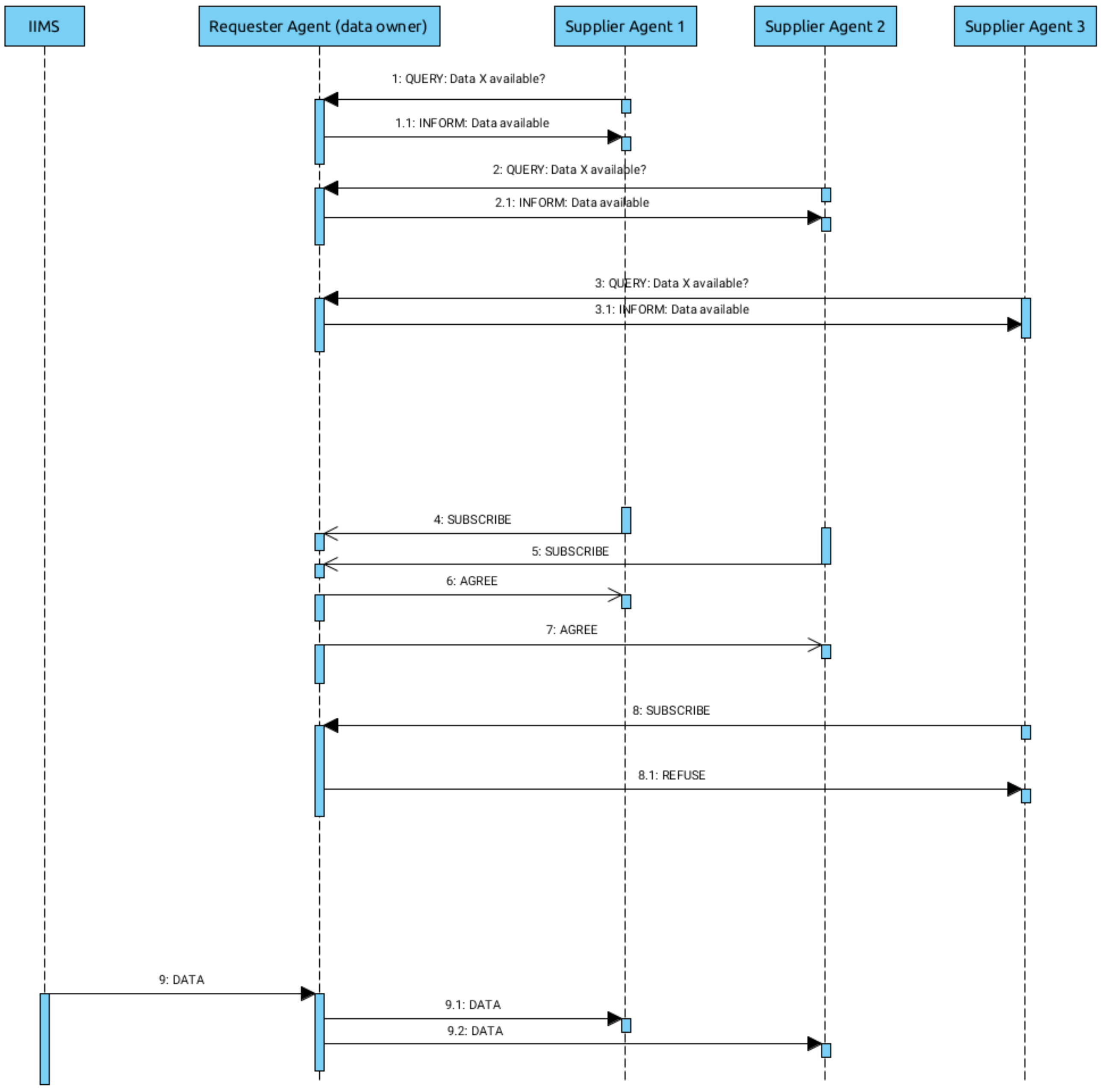

IIMS
Requester Agent (data owner)
Supplier Agent 1
Supplier Agent 2
Supplier Agent 3
1: QUERY: Data X available?
1.1: INFORM: Data available
2: QUERY: Data X available?
2.1: INFORM: Data available
3: QUERY: Data X available?
3.1: INFORM: Data available
4: SUBSCRIBE
5: SUBSCRIBE
6: AGREE
7: AGREE
8: SUBSCRIBE
8.1: REFUSE
9: DATA
9.1: DATA
9.2: DATA

Figure 51. Data routing information flow.

In push interaction, exchanged follow a well-defined sequence. First, all interested agents contact the data owner agent to verify that relevant data is actually available an potentially accessible (1,2,3). To perform such an action, a CXL QUERY message is issued. The data owner may reply or not to the query, and in case it decides to reply it might confirm or deny the existence or required data (1.1, 2.1, 3.1). In the example reported in Figure 51 all supplier agents receive a confirmation response. They can therefore activate the subscription process by sending a dedicated CXL SUBSCRIBE message to the data owner. At this point, the data owner replies positively to 2 out of 3 agents (using a CXL AGREE message – 6,7), while due to internal policies refuses subscription from the Supplier Agent 3 (using a CXL REFUSE message – 8.1). Sometimes later, data messages matching the agent requests appear on the IIMS broker (to which the data owner agent is subscribed) and the agent forwards such messages to the right recipients using CXL INFORM messages, carrying required data as payload.

Reputation data exchange

As witnessed by currently available marketplace solutions, such as the Google Play or the Amazon market, one of the key factor for successful adoption of the market is the availability of a solid reputation system. In scenarios where the amount of stakeholders possibly involved in setting-up deals, and in dynamically forming supply chains, is huge, the capability to discriminate reliable sellers from lower quality providers is of utmost importance. The COMPOSITION agent marketplace is designed to "mimic" the human-driven reputation systems currently in use by letting agents exchange reputation information about other agents. The approach, at this stage, is not yet completely defined; nevertheless, two main operation modes are foreseen: a central, collaborative agent ranking system and an agent-level personal ranking system.

The first is implemented by the Matchmaker, which collects evaluations about concluded deals. Every agent on the marketplace may decide to provide an evaluation for a certain deal (possibly driven by an underlying human-level assessment of the deal results), for a certain agent. When new negotiations start, the Matchmaker can exploit such reputation data to compute or refine the set of agents that are "compatible" with a given offer. Additionally, reputation data might be inserted in the provided list of agents in order to let the Requester decide to involve them or not, depending on some local reputation policy.

The second operation mode might be implemented by each agent. It consists of periodical "opinion" exchanges between collaborating agents where the "opinion-seeking" agent contacts known stakeholders, e.g., the one with which it had successful negotiations in the past, for gathering their opinion about a third party agent (see Figure 52). Inquired agents might decide to reply to such a request or not, and they might decide to provide false information attempting to reach monopolistic conditions[27]. Whatever decision they take, at some point in time the opinion requester agent will receive back a "biased" view of the agent marketplace, in terms of reputation values, that merged with its internal view will contribute to the "personal" reputation system of the agent itself. Such a system will then be exploited to filter or re-rank "compatible" agents received from the Matchmaker during negotiation, or to, e.g., refuse subscription requests from low ranked agents.

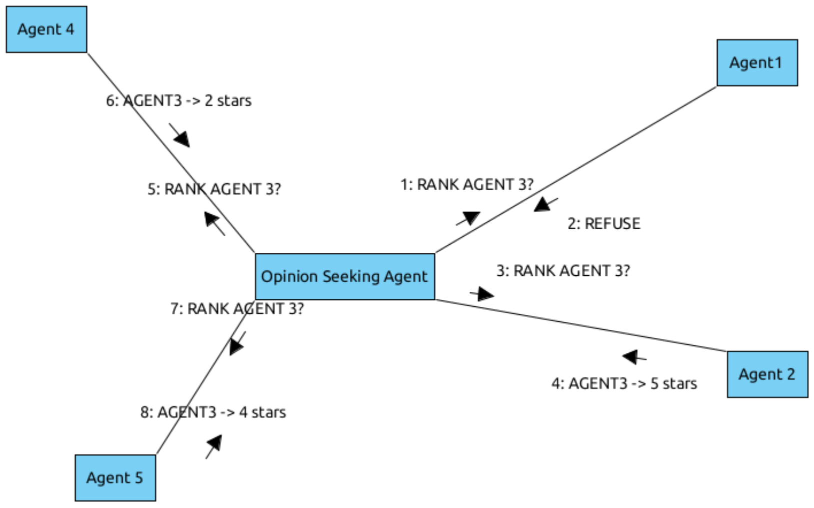


Figure 52. Personal opinion sharing between agents.

5.4.3.2.2 IIMS to Agent communication

Agents and corresponding IIMS have a privileged relationship that is reflected in a particular set of communication interfaces and interactions between the two systems. In a sense, a company agent is lying at the boundary between the company IIMS and the COMPOSITION marketplace and acts as interface to the latter. The agent has therefore a mixed nature by being at the same time an IIMS and a marketplace component.

Typical communication paradigms employed between a factory IIMS and the factory agent are subject to customization and/or adaptation commanded by the stakeholder needs / requirements. Nevertheless, in COMPOSITION we design and support a certain number of communication means and paradigms, depending on the task actually being carried by the IIMS. At the time of writing a precise specification is still lacking, however 3 main communication means (and interaction patterns) are under consideration: REST-based operation, queue-based messaging and publish-subscribe.

REST communication between the IIMS and the agent is envisioned for all cases in which the stakeholder does not run the COMPOSITION IIMS on premises, but prefers to exploit its own management suite (even

[27] This option is currently investigated in research and requires the "liar" agent to find the best deal between telling lies and gaining advantageous market positions and being spotted as a "liar" and get a very bad ranking.

manual management is allowed). In such a scenario, agents exposing their main functions through REST endpoints are likely to be easier to integrate and might therefore promote early adoption of the COMPOSITION marketplace.

When companies joining the COMPOSITION ecosystem decide to exploit the project IIMS, a privileged connection between the IIMS and the company agent(s) can be foreseen. In such cases, the IIMS can exploit the native communication infrastructure of the agent, i.e., its capability to handle distributed message queues (e.g., using RabbitMQ). Such an option may leverage CXL (or extended versions of it) to encode operating directives thus enabling a much easier and stricter interaction with the agent that natively supports such a message-based interaction paradigm.

Finally, both in companies exploiting the COMPOSITION IIMS and in companies selecting other platforms, a publish/subscribe interaction between the IIMS and the agent(s) is foreseen to effectively handle asynchronous data delivery from the factory shop floor sensors, and management systems, to interested agents on the marketplace. The publish-subscribe pattern, in fact, lets the agent subscribe to topics corresponding to relevant data only, and to duly forward it to the final recipients over the marketplace.

It is important to notice that the brokering systems currently targeted in COMPOSITION can operate on both distributed queue and publish-subscribe modes, thus gracefully supporting the last 2 operation modes, which are expected to be active at the same time (see Figure 53 for a comprehensive overview).

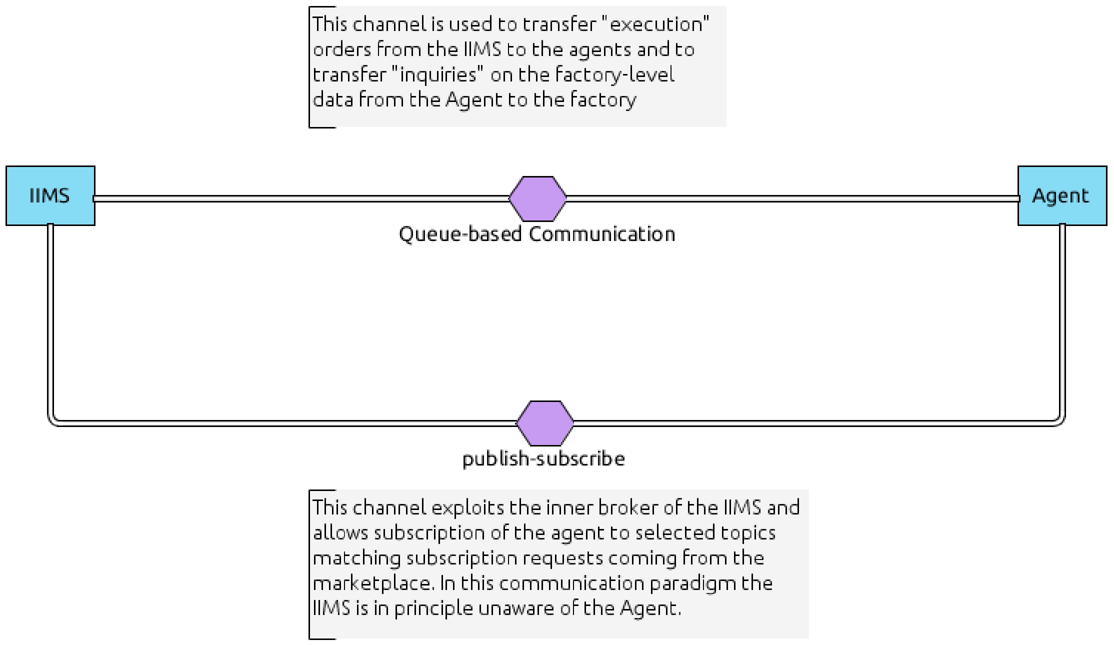

This channel is used to transfer "execution" orders from the IIMS to the agents and to transfer "inquiries" on the factory-level data from the Agent to the factory
IIMS
Agent
Queue-based Communication
publish-subscribe
This channel exploits the inner broker of the IIMS and allows subscription of the agent to selected topics matching subscription requests coming from the marketplace. In this communication paradigm the IIMS is in principle unaware of the Agent.

Figure 53. Communication channels between the IIMS and the factory agent.

5.5 Deployment View

5.5.1 Docker

One of the critical points in adopting new systems in productive contexts is the need to perform specific hardware and software set-up, which are typically difficult to deploy, as companies have precise software deployment policies, rather strict options on operating systems and public access to company IT services. These restrictions are strongly dependent on company-level decisions and are the result of years of operation in real business.

In COMPOSITION, we have clear in mind that any particular technological requirement for the COMPOSITION IIMS and Marketplace may hamper or slow down adoption of the platforms in the real world. Therefore, after a careful evaluation of possible solutions, included PAAS and SAAS solutions (which on the other hand could be difficult to handle due to data ownership issues), the technical partners, in accordance with industrial stakeholders, identified Docker as a viable deployment infrastructure.

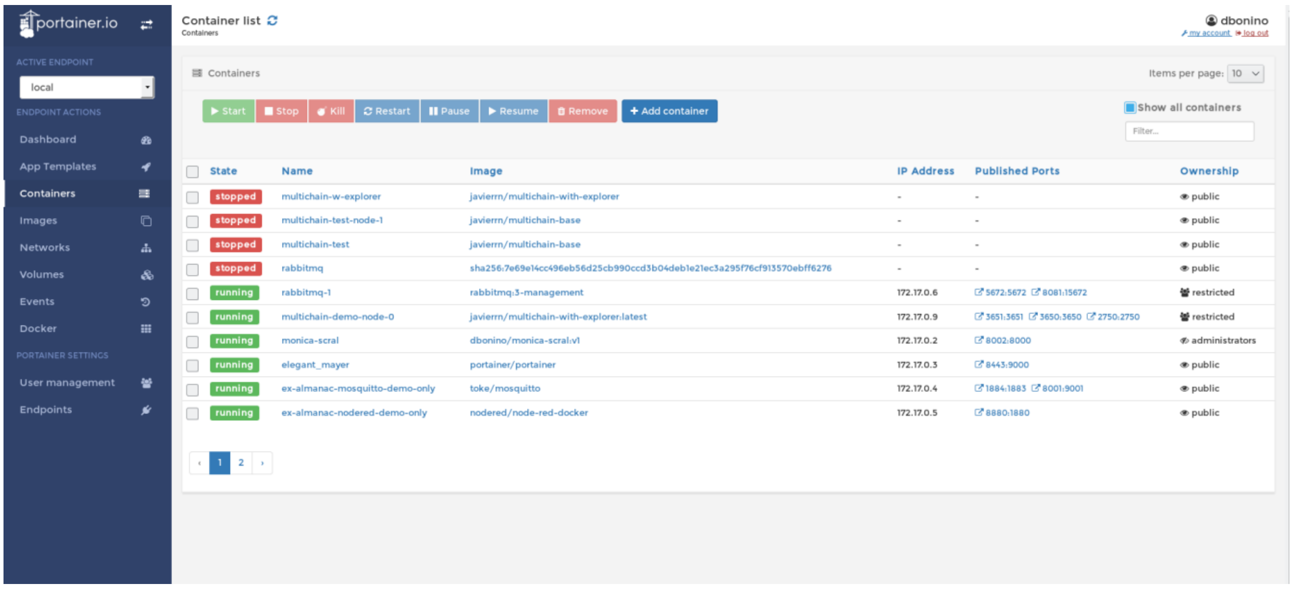
Docker is an open-source project aiming at automating the deployment of applications as portable, self-sufficient containers that can run virtually anywhere, on any kind of server. It can be considered as a lightweight alternative to full machine virtualization provided by hypervisors such as ESXi, Xen or KVM. 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.

Docker containers are designed to run on a wide range of platforms ranging from physical computers to bare-metal servers and up to cloud clusters, e.g., based on OpenStack. Technically speaking Docker extends the LinuX Containers (LXC) format designed to provide an isolated environment for applications, by enabling image management and deployment services. Among supported platforms, we can cite:

- Mac, Windows and Linux desktops
- AWS and Azure cloud services
- Windows, CentOS, Debian, Fedora, Oracle Linux, RHEL, SLES and Ubuntu servers.

This ensures the ability to deploy Docker-based COMPOSITION components on virtually all possible IT infrastructure available on site. Since deployment is a crucial part of the agile development process adopted in COMPOSITION, components are wrapped into Docker images since the very beginning. All continuous integration and testing processes in the project rely on Docker and act upon Docker images. This ensures full compatibility of systems under development with the targeted deployment tools.

In order to spread the Docker approach among the consortium, and to support such a container-based integration approach, a dedicated "test" server has been configured and made available to all technical partners for benchmarking and testing solutions under development (see Figure 54).

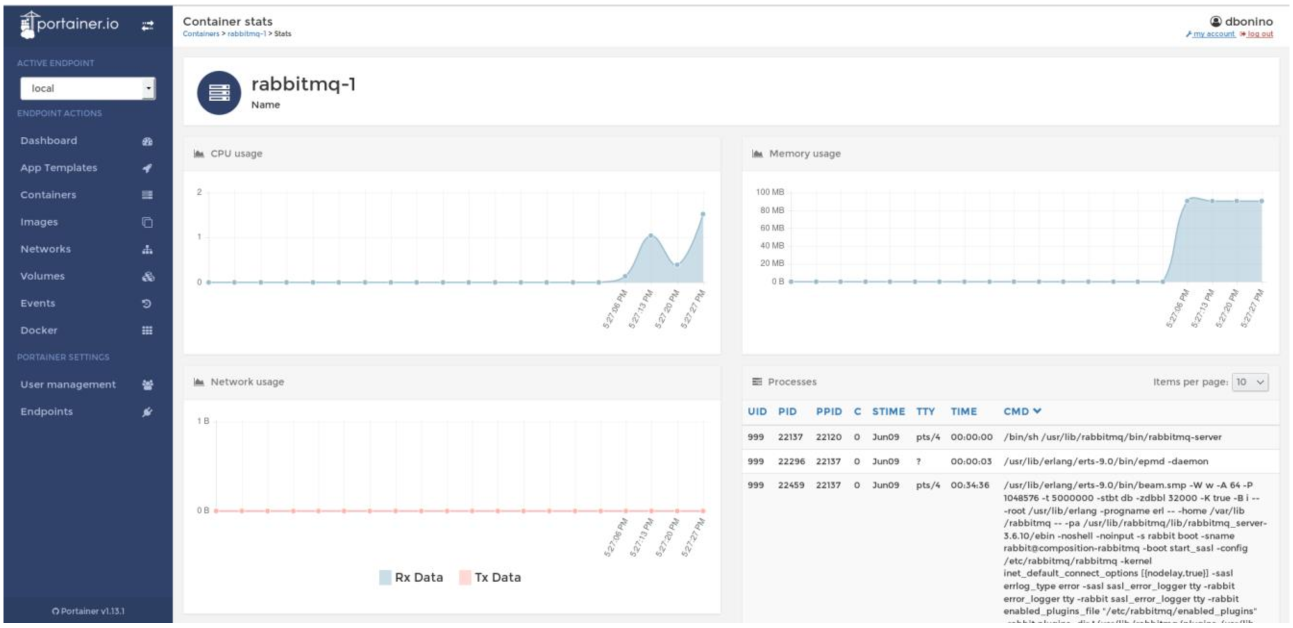


State	Name	Image	IP Address	Published Ports	Ownership
stopped	multichain-w-explorer	javierrn/multichain-with-explorer	-	-	public
stopped	multichain-test-node-1	javierrn/multichain-base	-	-	public
stopped	multichain-test	javierrn/multichain-base	-	-	public
stopped	rabbitmq	sha256:7e69e14cc496eb56d25cb990ccd3b04deb1e21ec3a295f76cf913570ebff6276	-	-	public
running	rabbitmq-1	rabbitmq:3-management	172.17.0.6	5672:5672 8081:15672	restricted
running	multichain-demo-node-0	javierrn/multichain-with-explorer:latest	172.17.0.9	3651:3651 3650:3650 2750:2750	restricted
running	monica-scral	dbonino/monica-scral:v1	172.17.0.2	8002:8000	administrators
running	elegant_mayer	portainer/portainer	172.17.0.3	8443:9000	public
running	ex-almanac-mosquitto-demo-only	toke/mosquitto	172.17.0.4	1884:1883 8001:9001	public
running	ex-almanac-nodered-demo-only	nodered/node-red-docker	172.17.0.5	8880:1880	public

Figure 54. Example of Docker Management platform showing a set of deployed containers.

Thanks to a dedicated web management tool, namely Portainer[28], also deployed as a Docker container, partners and in general all technical stakeholders have the ability to publish, run and test the single COMPOSITION components under their respective responsibility. Continuous monitoring and logging infrastructure allow deep analysis of the performances of deployed software that can both be carried before the final deployment inside factories and during real-world operation (see Figure 55).

[28] http://portainer.io/



UID	PID	PPID	C	STIME	TTY	TIME	CMD
999	22137	22120	0	Jun09	pts/4	00:00:00	/bin/sh /usr/lib/rabbitmq/bin/rabbitmq-server
999	22296	22137	0	Jun09	?	00:00:03	/usr/lib/erlang/erts-9.0/bin/epmd -daemon
999	22459	22137	0	Jun09	pts/4	00:34:36	/usr/lib/erlang/erts-9.0/bin/beam.smp -W w -A 64 -P 1048576 -t 5000000 -stbt db -zdbbl 32000 -K true -B i -- -root /usr/lib/erlang -progname erl -- -home /var/lib /rabbitmq -- -pa /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.10/ebin -noshell -noinput -s rabbit boot -sname rabbit@composition-rabbitmq -boot start_sasl -config /etc/rabbitmq/rabbitmq -kernel inet_default_connect_options [{nodelay,true}] -sasl errlog_type error -sasl sasl_error_logger tty -rabbit error_logger tty -rabbit sasl_error_logger tty -rabbit enabled_plugins_file "/etc/rabbitmq/enabled_plugins"

Figure 55. Example of single container monitoring.

Docker natively supports distribution and replication of services. Moreover, it can easily be deployed on cloud-based platforms. This flexibility is a strong hint to the fact that such a deployment design choice will not generate issues when upscaling of performance will be required.

We expect to gather the first feedbacks from end-users and some confirmation of the viability of this approach after the completion of the first development cycle, expected to happen around the end of the first year of the project.

6 System Quality Perspectives

6.1 Security Perspective

6.1.1 Authentication and Authorization

One of the key aspects of the security in COMPOSITION are the authentication and authorization mechanisms. These tasks fall on two components: Keycloak responsible for identity and access management, and the Authorization Service responsible for authorization and access control to data and resources. To keep Keycloak and Authorization Service as the core components for authentication and authorization there is the need add other components like the RabbitMQ Authentication Service which is a service that will allow overriding RabbitMQ built-in mechanisms for authentication and authorization. In case there is the need to do the same as with RabbitMQ for another COMPOSITION platform component the same procedure will be used if possible.

The next figure shows an overview of the architecture of the Authentication and Authorization framework:

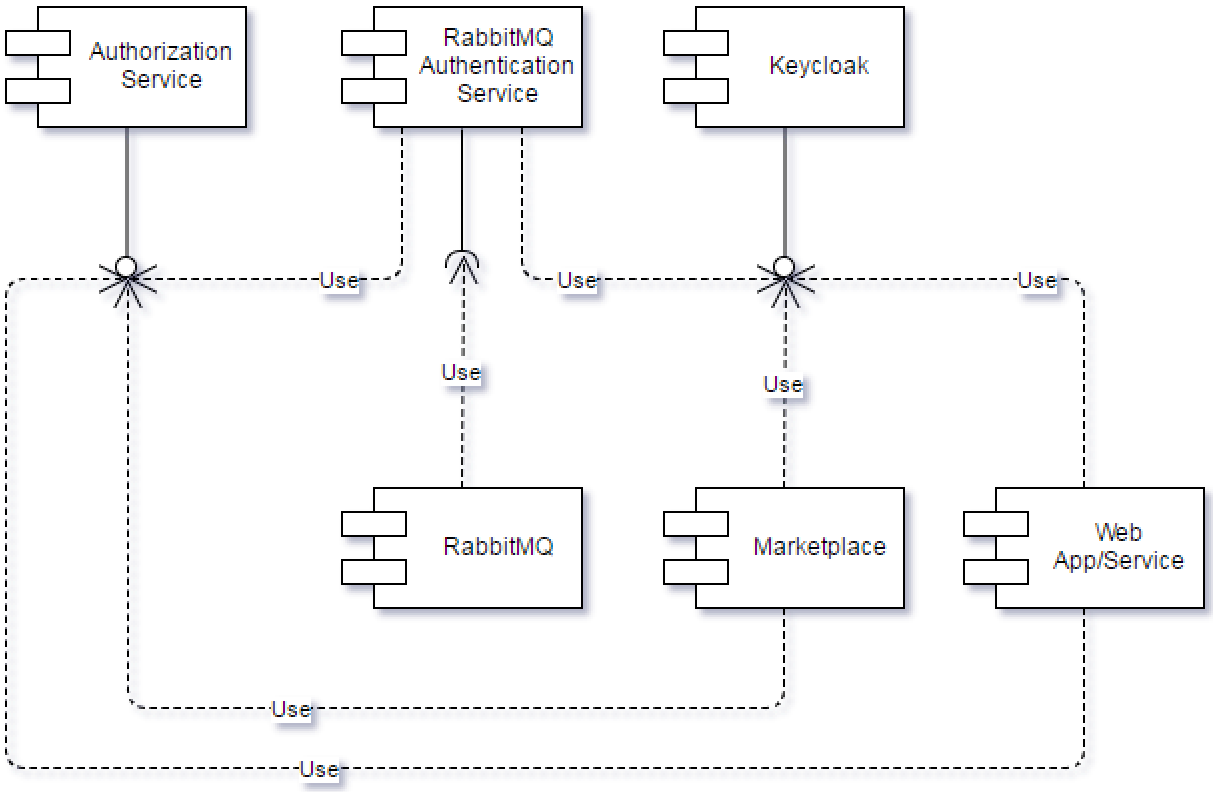


Figure 56. The Authentication and Authorization framework.

Any user or application needs to be identified through Keycloak before having access to the secured COMPOSITION applications and services. Once identification is successful Keycloak issue a token which is valid for a limited time and should be renewed after it expires. This token is also the one used by the Authorization Service to allow or deny access to data and resources based on the rights of the user/application and the rules stored in the Authorization Service.

6.1.2 Blockchain Uses

Other areas where COMPOSITION aims to offer a high level of security are:

- IPR, Confidentiality and Data integrity
- Log and Traceability

For that it has been considered the use of a blockchain implementation; in this case Multichain, which is a private blockchain based on Bitcoin with interesting new features implemented like data streams and managed permissions. Since it´s a private blockchain there is no need for mining which is an important aspect to have into account, as transactions will have no cost if desired.

6.1.2.1 **IPR, Confidentiality and Data integrity**

In the case of protecting IPR, COMPOSITION proposal is to use the blockchain to get a digital certificate of authentication for any kind of digital document without storing the document itself in anyway in the blockchain. The next figure is an overview of the architecture.

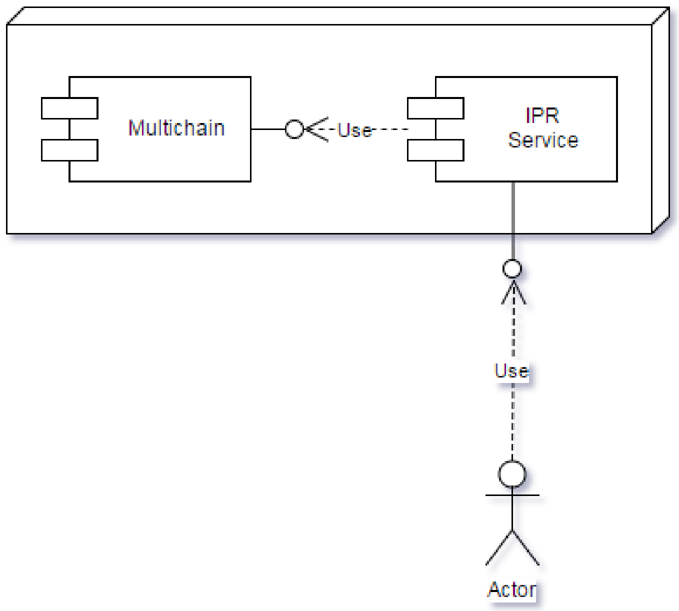


Figure 57. IPR Service.

The method to obtain a certificate for a document is pretty simple:

1. Upload document
2. IPR service calculate hash and store in blockchain
3. Return hash

The following figure depicts the process in detail:

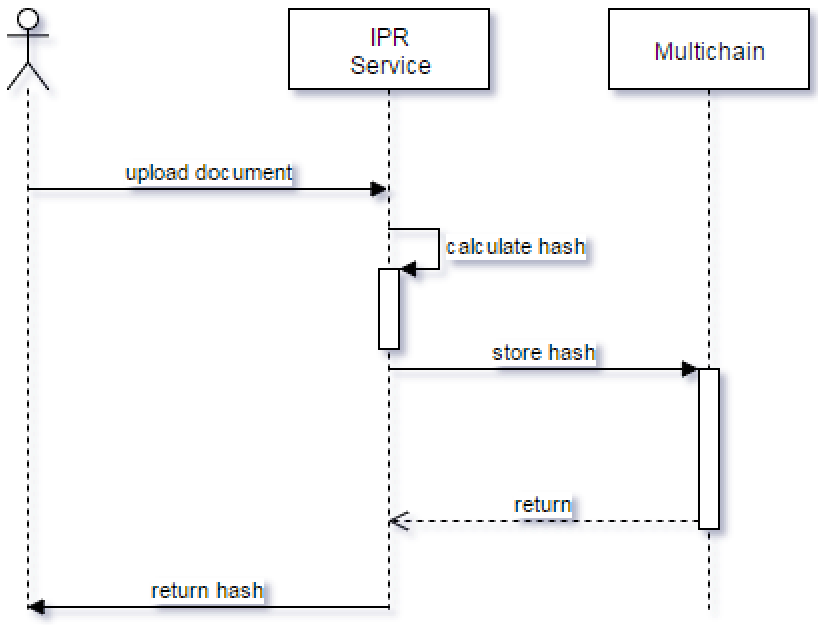


Figure 58. IPR Service sequence diagram.

The method to check if a document existed at any given time is fairly simple also. The steps are the following:

1. Upload document
2. IPR service calculate hash
3. IPR service checks hash in block chain
4. Return date if found

To ensure Confidentiality, Data Integrity and also IPR of the messages/data sent across the platform using RabbitMQ message broker Multichain will be used in a similar way as with the certificate of authentication explained before. It´s important to note that, as in the previous case the message or data itself it´s not stored in the blockchain. The architecture can be depicted in the following figure:

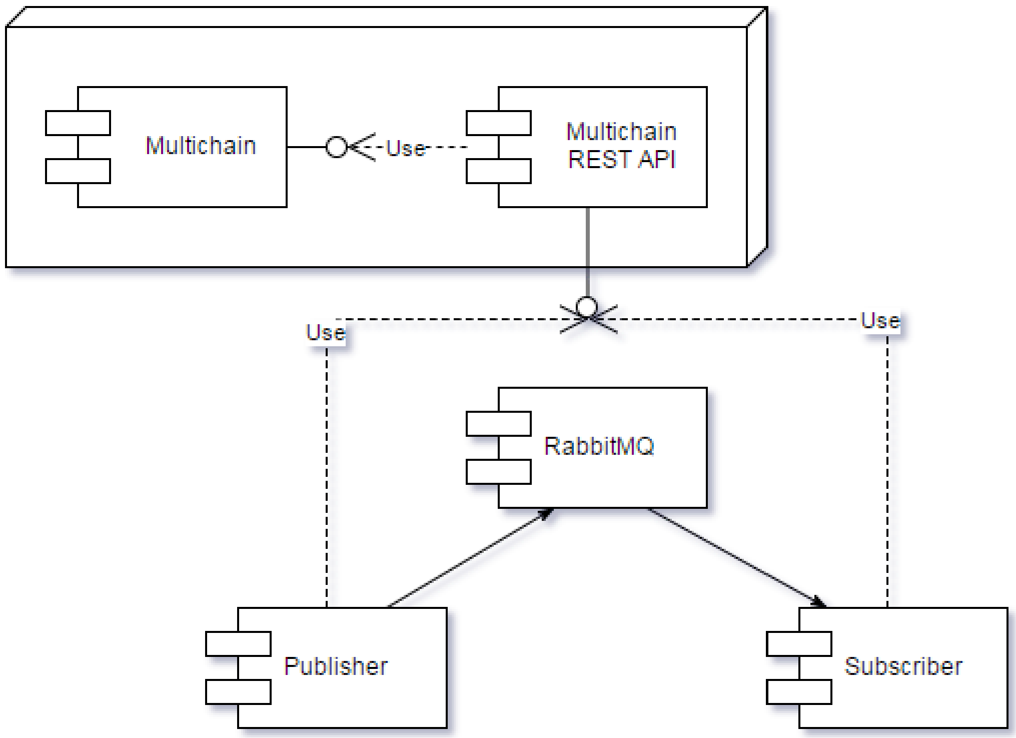


Figure 59. Blockchain used for distributed trust in messaging.

Before sending any message/data a publisher must first sign the message/data using a service created for that purpose. Afterwards it can send the message using RabbitMQ message broker. Any subscriber receiving the message can check if the data has been modified in any way and ensure that is coming from where it is assumed. This is done by uploading the message/data in the service which will calculate the hash and will check if the same hash it´s already in the blockchain. The following figure depicts the whole procedure in detail:

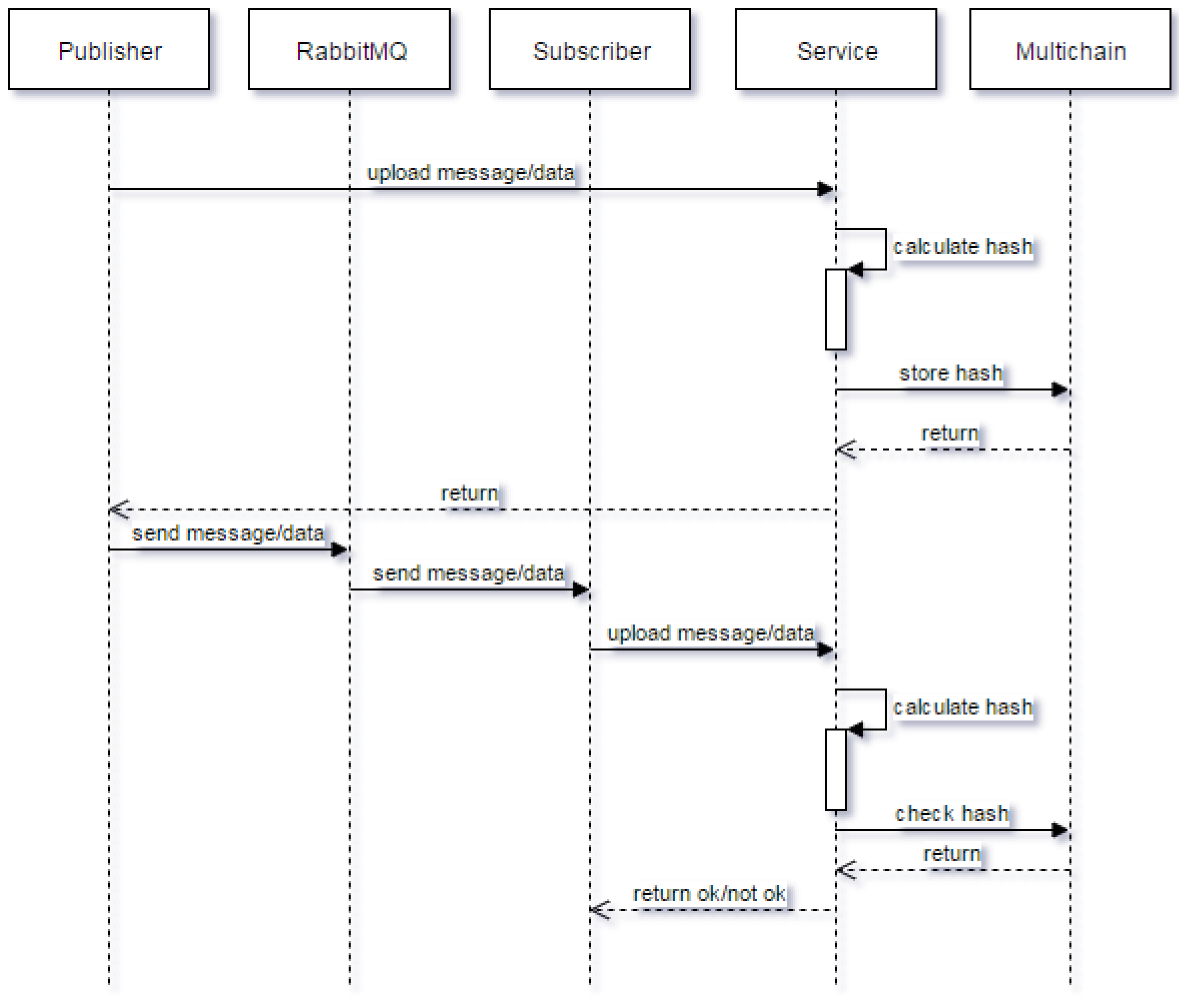


Figure 60. Sequence diagram of integrating blockchain in message sending.

6.1.2.2 **Log and Traceability**

Multichain will also be used to provide an audit trail for manufacturing and supply chain data enabling both product data traceability and secure access for stakeholders. An approach of the architecture to be used is shown in the following figure.

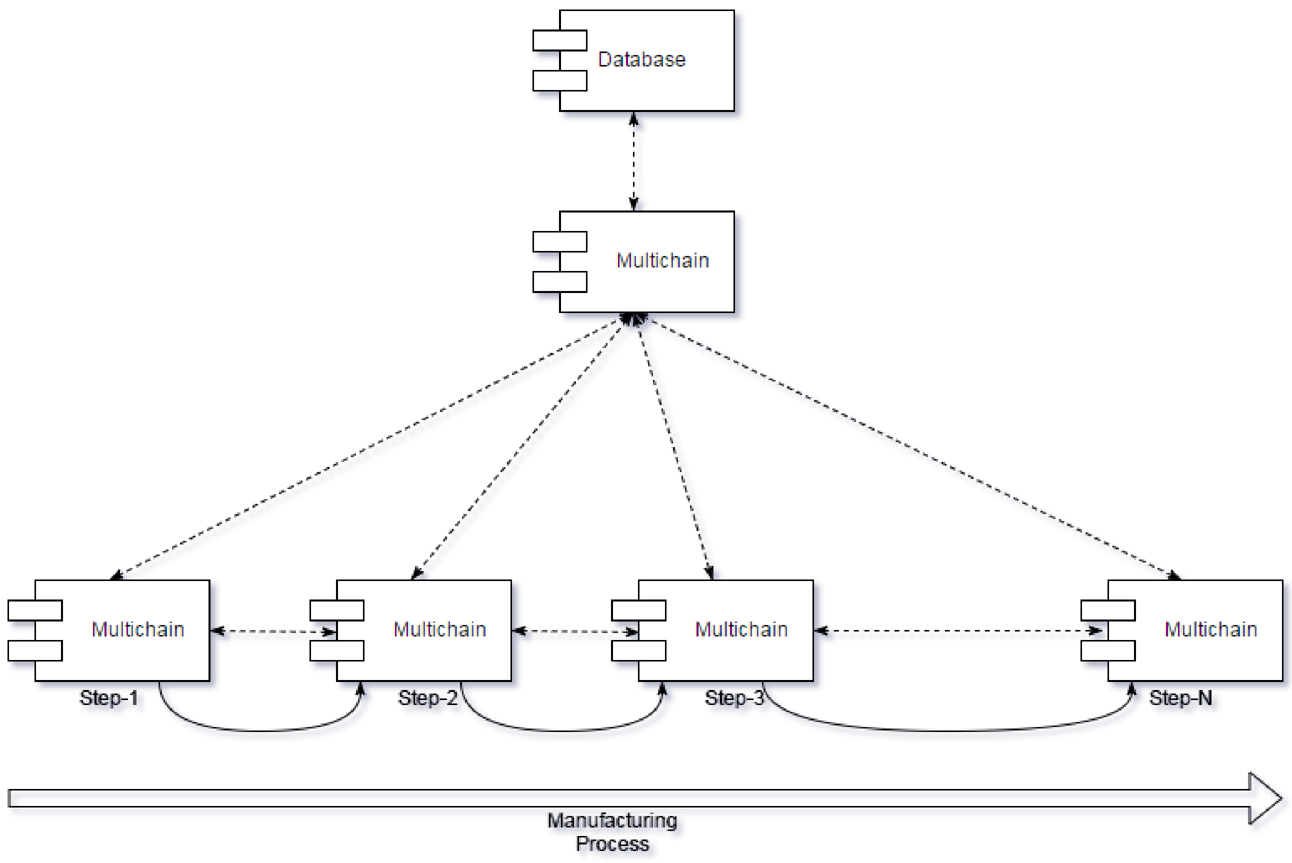


Figure 61. Blockchain in manufacturing process.

The idea is to have multiple blockchain nodes along the whole manufacturing process with a central node. Each node in the chain will make a transaction to the next node with the data available at each stage of the process. Each node will add its own data to the one received from the previous node. As each transaction is stored in the blockchain by the end of the manufacturing process it will be possible to have a clear overview of what happened on each of the steps.

An advantage of this approach is that since the blockchain acts like a network of replicated databases, this means all nodes have exactly the same information; it's very difficult that a problem in the system may cause the loss of data. The failure of a node it´s not a big problem either, as replacing a node it´s really easy and as soon as it is connected to the network all data will be replicated on it.

It should be also considered only to store relevant data in each transaction while all other data is stored on an external database and linked to the data in the blockchain.

6.1.3 Cyber-Security

Cybersecurity is another key aspect on COMPOSITION. The deployment of Atos SIEM and the Cyber-Agents it will ensure the security monitoring and protection against potential threats such as abuse of privileges or denial of access among others. A Cyber-Agent is deployed on each system to be monitored. All events are sent to the SIEM which will collect and analyze the security events and will raise an alert in case of a potential risk. The SIEM supports several ways of communication to obtain the events from the Cyber-Agents, in case of COMPOSITION as the system already uses RabbitMQ it has been decided to use it as the communication channel. The next figure is an overview of the architecture:

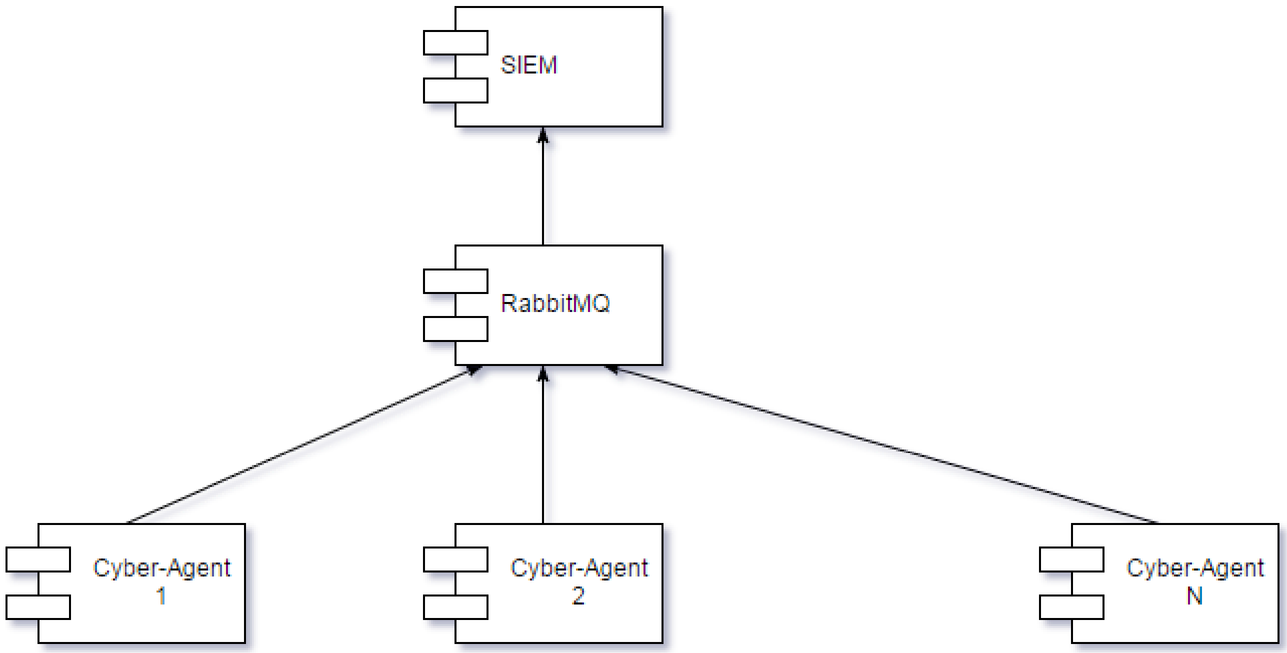


Figure 62. Cyber-security components.

6.1.4 Transport Layer

All communication between COMPOSITION components should be encrypted using TLS/SSL where possible. In case of web applications and services its planned to use Nginx as a reverse proxy, with this approach all applications and services can run their own web servers and do not need to implement TLS/SSL on their own ad Nginx will take care of it. In the case of RabbitMQ message broker it needs to be configured to allow encrypted message transactions.

6.2 Scalability Perspective

This section describes scalability concerns and how the chosen design decisions and mechanisms can adopt measures to address these concerns. At the time of writing this report, scalability concerns and scenarios has not yet been addressed for all components. The described approach will be applied to the components and to the system in future work and reported in the updated version of this document M24.

6.2.1 Basic Concepts and Terminology

6.2.1.1 Nodes, Resources and Scalability

As described in the Deployment View, each component will run in a Docker container; a virtual computational resource (node) with a certain specified computing and/or storage *capacity*. Other examples of nodes are physical servers, cloud services and execution containers in the cloud.

Computational resources are thus constrained by the amount allocated to the node with the limitations of the docker host being the upper limit, which means the physical specification of the hardware if this is a locally hosted deployment or in the case of cloud based provisioning by the corresponding SLAs (Service Level Agreements).

For the sake of clarity, we would like to differentiate between performance and scalability. By performance we refer to the capability of a system to provide a certain response time *with a given set of nodes and resources*, e.g., to serve a defined number of users or processes a certain amount of data from a server with a certain capacity specification. Although no standard definition is available for these terms (Lehrig, Eikerling, & Becker, 2015), most of the available literature uses a similar definition for performance, e.g. (Wilder, 2012) where it is defined as "… an indication of the responsiveness of a system to execute any action within a given time interval".

Scalability we would like to define in analogous to the definition in (Lehrig, Eikerling, & Becker, 2015) as the ability of a system to increase the maximum workload it can handle by expanding its quantity of consumed resources. Similar definitions are “the ability of a system either to handle increases in load without impact on performance or for the available resources to be readily increased” (Wilder, 2012) or “the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth” (Bondi, 2000).

Scaling is thus about allocating more *resources* for an application, i.e., resource *provisioning*. In this discussion, we assume that the system has been designed to use the available resources as efficiently as possible i.e., by maximizing the performance with a given set of resources. Examples of resources needed by an application usually include CPU, memory, disk (capacity and throughput), and network bandwidth. An application or service is said to be scalable if when we increase the resources in a system, it results in increased performance in a manner proportional to resources added. Resources can be handled in *scalability units,* i.e., groups of resources that could be scaled together.

6.2.1.2 **Vertical/Horizontal scaling**

The scaling discussed here concerns the steps that may be taken when the available resources run out and the application does not fulfil its functional or non-functional requirements - the maximum workload of the system with the given resources is reached. We may then scale the system to increase the maximum workload it can handle by expanding its quantity of available resources. We can increase the quantity of consumed resources by increasing the amount of resources within existing nodes, or by adding more nodes.

To *scale up (or scale vertically)* is to increase overall application capacity by increasing the resources within existing nodes. In COMPOSITION, e.g., increasing the capacity of the node running the message broker in the IIMS. (For a Docker container, this can be achieved by using options such as “--cpus” and “--memory-reservation”.).

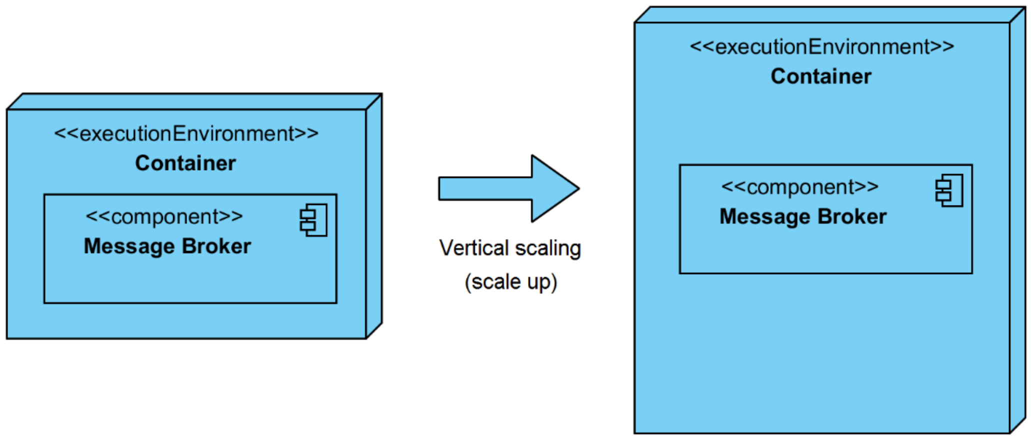


Figure 63. Boost capacity of node, scale up.

Scaling up is usually the simplest and cheapest solution, as is does not require any changes to the design, code details or deployment of the application. While less complex (and sometimes cheaper compared to re-design or code improvements to increase performance) there are limitations to this approach compared to scaling out.

To *scale out (or scale horizontally)* is to increase overall application capacity by adding nodes, e.g., adding an additional message broker to the IIMS. (For Docker, this can be achieved by using options such as “--scale” or using docker swarm.)

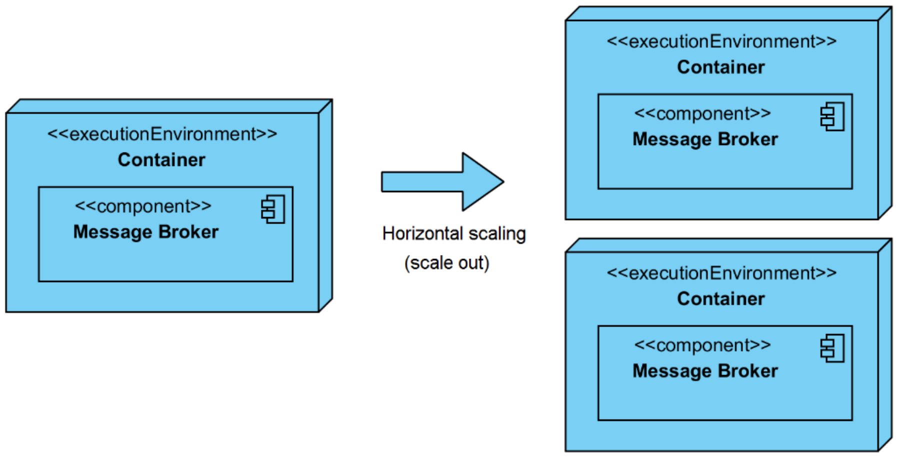


Figure 64. Scale out by adding nodes for component.

Scaling out increases the overall application capacity by adding entire new computational nodes. Scaling out tends to be more complex than scaling up, and has more impact on the application architecture. We may scale out a COMPOSITION system instance by adding nodes for specific components (e.g., a Match Maker) and implement support for this at the component level. In the case of horizontal scaling, the system should also be able to adapt to shrinking demand for resources, to *scale in*. This property is often referred to as *elasticity* (Lehrig, Eikerling, & Becker, 2015).

When all the nodes supporting a specific function are configured identically - same hardware resources, same operating system, same function-specific software - we say these nodes are *homogeneous*[29]. We would add that components executing on different nodes may be homogenous with regards to functionality – all nodes support the same functions – and data or state – all nodes share the same data. This has implications on the design of horizontal scaling.

An *autonomous* node does not know about other nodes of the same type, similarly the same term can also be used for components.

In COMPOSITION, we will not test scalability by creating a model of the system and performing simulations. The approach we will take is to identify the scalability issues by analysis of deployed capacity, against application performance requirements, identifying scenarios where the maximum workload may exceed the capability of the system or components, investigate common design patterns for how these scenarios may be addressed, and, determine how the design of components deals with scaling up and out.

6.2.2 Issue identification and analysis

In this section, we list a number of scalability quality attribute scenarios where a high value of the attribute may cause the workload to exceed the maximum that the system or individual components can handle. Common design patterns to address these problems are described. The component designs and architectural decisions are described from a scalability viewpoint; the possible bottlenecks of each component, the possibility of scaling up or out, and the design implications. The work on this has just started at the time of writing and the section will be revised in the updated version of this document M24.

6.2.3 Scenarios for scalability requirements of the system

6.2.3.1 Attributes that may affect workload of system or components

- Factory IIMS
    - The number of concurrently reporting sensors/field devices
    - The number of concurrently reporting BDA and ANN generated data streams
    - The number of generated data that should be persistently stored in the system (for future deep learning network training or in the blockchain
        - Number of generated data streams

[29] Wilder, Bill. Cloud Architecture Patterns: Using Microsoft Azure. Sebastapol, CA: O'Reilly Media, Inc., 2012

        - Number of observations
    - The number of concurrent queries for stored data
- Marketplace
    - The number of concurrent agent negotiations.
    - The number of concurrent requests for Matchmaker services
    - The number of concurrent requests for block chain storage

6.2.3.2 **Performance attributes affected**

- Response time
    - Service time - how long it takes to do the work requested.
    - Wait time - how long the request has to wait for requests queued ahead of it before it gets to run.
    - Transmission time – How long it takes to move the request to the computer doing the work and the response back to the requestor.
- Throughput
    - The amount of work accomplished in a given amount of time.
- Resource usage
    - CPU usage
    - Memory usage
    - Storage usage
    - Network usage - data sent and received

6.2.4 Performance and Scalability Design

Some examples of common design patterns used for performance and scalability are summarized here for convenience so that they may be referenced in the component scalability design section. More comprehensive descriptions may be found in e.g. (Wilder, 2012), (Homer, Sharp, Brader, & Swanson, 2014) or (Fowler, 2002).

6.2.4.1 **Caching**

When certain sets of data are frequently accessed, these may be copied to fast storage located close to the requesting application. E.g. since REST interfaces are employed for request response communication, HTTP caching may be used to avoid unnecessary load on the system by caching data at the HTTP client. Caching can also be performed in the Intra-factory Interoperability Layer or the Broker.

6.2.4.2 **Materialized Views**

HMI and other components may have need for views on data that is not stored or formatted in a way optimal for the query required to produce this view. The system may then generate prepopulated views over the data, possibly cached locally at the requesting node.

6.2.4.3 **Throttling**

To avoid that a single application or input source degrades the entire system, the services provided by the system may be temporarily limited. E.g. an agent sending a lot of requests may get a “503 Service Unavailable” response telling it to wait, or some functionality of the Marketplace or IIMS may be prioritized in case of insufficient resources.

6.2.4.3.1 Data partitioning

Data stored or processed in the system may be physically divided into separate nodes, so that they are not homogenous with respect to the data they manage. Using horizontal partitioning, the nodes may use the same schema but hold different parts of the data (e.g. different big data analytics nodes may process the same type of data but from different sources). With vertical partitioning, nodes will hold different parts of the schema, e.g. a broker instance may process only request-response type messaging or a storage node may only hold observation data. When different parts of the schema are handled by different nodes based on business or usage context, the term functional partitioning is sometimes used.

6.2.4.4 **Load balancing**

When the maximum workload of a single component is reached, redundant deployments of the component are created and a load balancing system dynamically distributes workloads. If the component works without state between calls and function calls are idempotent, this strategy is easier to implement.

6.2.4.5 **Queue based load levelling and competing consumers**

Instead of passing requests directly on to other components, a message queue can be used to implement the communication channel between the components. The sender component(s) post requests in the form of messages to the queue, and the consumer component(s) receive messages from the queue and process them, each at its own pace. This way, fluctuations in workload and differences in throughput between various parts of the system can be balanced, and individual components can be scaled out to optimize throughput.

6.2.4.6 **Local hosting vs cloud hosting**

The COMPOSITION system may be hosted, in whole or in parts, on physical or virtual hardware, in an environment owned and operated by a business (e.g. for a private marketplace) or in a cloud environment (e.g. Amazon, Azure).

Depending of the choice of hosting it may be possible to scale up or out automatically. In any case and at the very least, the components need to be able to indicate, when queried or by events, that the capacity limit is being reached. The systems administrator or an auto-scale component may use this information to start provisioning new resources. When automatically scaling out, the component should also be able to be elastic and scale in if there is less demand for resources.

6.2.5 COMPOSITION Scalability Design

As a result of scalability analysis, we will describe how the system components and the architectural mechanisms may be affected by the scenarios and how the design of the component or the choice of design and implementation mechanism makes use of added resources to make the system scalable. Scalability is primarily discussed at component level. The scalability analysis is still in progress at the time of writing and will be updated in the future revision of this document.

Some short preliminary results may be reported:

Blockchains that use proof-of-work mechanisms and use a global chain are hard to scale. The chosen implementation mechanism (Multichain), however, is configurable for consortium or private use, where proof-of-work is not needed. This greatly speeds up the generation of new block and thus the time to accept new transactions. Multichain also allows for the creation of several parallel chains, which allows for horizontal or vertical partitioning of the log.

For the data persistence solution (section 5.4.2), horizontal partitioning may be used to reduce the disk size of storage instances. If the number of concurrent requests for stored data is high, redundant storage behind a load-balancer may be used.

The centralized communication design using the message broker risks introducing a bottle neck in the design. However, clustering of brokers behind a load-balancing system is a triad approach to this problem. The chosen implementation (RabbitMQ) is also available as dynamically scalable cloud service. This approach is also applicable to the Matchmaker, which also may become a frequently requested resource in the agent marketplace.

Docker supports control of both horizontal and vertical scaling of containers. It also makes migration of containers to more capable hardware and re-configuration components to implement strategies such as queue based load levelling or load-balancing easy compared to installations on virtual machines.

7 Summary and future work

The architecture work has so far been performed in a bottom-up approach, extending the inception work presented in the project specification. Components have been identified, mechanisms evaluated and selected, the general functional decomposition has been set and the use and dependencies of models in the system investigated. A unified end-to-end security framework has been designed which aligns with the architecture. The architecture work is expected to move to a top-down approach as the component designs mature. Workshops evaluating architecture perspectives such as security, scalability and evolution will be held, possibly also architecture evaluations. The RAMI4.0 alignment, important to the dissemination of the project, and FIWARE and FITMAN extension and integration will also be further developed.

8 References

Bondi, A. (2000). *Characteristics of scalability and their impact on performance.* Proceedings of the second international workshop on Software and performance - WOSP '00.

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

Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Addison Wesley.

Homer, A., Sharp, J., Brader, L. N., & Swanson, T. (2014). *Cloud Design Patterns.* Microsoft patterns & practices.

IEC. (2013). *IEC 62890: IEC Project: Life Cycle Management for Systems and Products used in Industrial-Process Measurement, Control, and Automation.* IEC.

IEC62264. (2013). *IEC 62264-1: Enterprise-control system integration Part 1: Models and Terminology.* IEC.

IEEE. (2000). *IEEE 1471 Recommended Practice for Architectural Description for Software Intensive Systems.* IEEE.

ISO/IEC/IEEE42010. (2011). *ISO/IEC 42010: Systems Engineering – Architecture description.* ISO/IEC/IEEE.

ISO19156. (2011). *Geographic information -- Observations and measurements.* ISO.

Kruchten, P. (2004). *The Rational Unified Process: An Introduction.* Addison-Wesley Professional.

Lehrig, S., Eikerling, H., & Becker, S. (2015). *Scalability, Elasticity, and Efficiency in Cloud Computing: a Systematic Literature Review of Definitions and Metrics.* Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA '15), Montreal, QC, Canada, May 4–7.

Rozanski, N., & Woods, E. (2012). *Software Systems Architecture,: working with stakeholders using viewpoints and perspectives.* Addison-Wesley.

(2015). *Status Report Reference Architecture Model Industrie 4.0 (RAMI4.0).* Düsseldorf: VDI e.V.

Wilder, B. (2012). *Cloud Architecture Patterns.* O'Reilly.

Y.2060, I.-T. (2012). *ITU-T Y.2060 : Overview of the Internet of things.* ITU.

9 List of Figures and Tables

9.1 Figures

9.2 Tables