EFPF: European Connected Factory Platform for Agile Manufacturing

European Factory Platform

WP3: EFPF Architecture

D3.1: EFPF Architecture-I
Vs: 1.0

Deliverable Lead and Editor: Rohit Deshmukh, FIT

Contributing Partners: FIT, ICE, SRFG, ASC, VLC, CNET, CERTH, C2K, NXW, ALM, AID, FOR, SRDC, LINKS

Date: 2019-09

Dissemination: Public

Status: <~~Draft~~ | Consortium Approved | ~~EU Approved~~>

Short Abstract

This deliverable presents the baseline architecture of the EFPF ecosystem with focus on the EFPF Platform and the Data Spine, the interoperability backbone of the EFPF ecosystem. The architecture is both modular and extensible to meet the need for incorporating new tools in the EFPF platform, new platforms in the EFPF ecosystem and also to accommodate the needs of users and experimenters. The baseline architecture presented in this deliverable will be revised in D3.2 at M18 of EFPF project.

Grant Agreement: 825075

Document Status

Deliverable Lead	Rohit Deshmukh, FIT
Internal Reviewer 1	Norman Wessel, ASC
Internal Reviewer 2	Violeta Damjanovic-Behrendt, SRFG
Type	Deliverable
Work Package	WP3: EFPF Architecture
ID	D3.1: EFPF Architecture-I
Due Date	2019-09
Delivery Date	2019-09
Status	<~~Draft ¦~~ Consortium Approved ~~¦ EU Approved~~>

History

See Annex A.

Status

This deliverable is subject to final acceptance by the European Commission.

Further Information

www.efpf.org

Disclaimer

Project Partners:

Executive Summary

This deliverable presents the baseline architecture of the EFPF ecosystem that is composed of the Data Spine, the EFPF platform, the four base platforms (from H2020 FoF-11-2016 cluster: NIMBLE[1], COMPOSITION[2], DIGICOR[3], and vf-OS[4]) and external platforms. The deliverable primarily focuses on the architectural specification of the two major components of the EFPF ecosystem: the EFPF platform and the Data Spine. It details the architectural design of the subcomponents of the Data Spine and the high-level architecture of the tools and services that together constitute the EFPF platform. However, the detailed or internal architecture of the EFPF platform tools and services is not the focus in this deliverable. Rather, how these tools and services connect or interplay with the EFPF Data Spine or within the EFPF platform is the main concern.

The architecture of the EFPF Data Spine and Platform has been designed with modularity and extensibility in mind to meet the need for incorporating new tools in the EFPF platform and external platforms in the EFPF ecosystem, with minimum effort, and also the needs of users and experimenters.

The architecture described in this deliverable is fundamentally aimed at enabling the provision of an operational technology (e.g. digital manufacturing) platform. To achieve this, the architecture definition considers the resources (i.e. tools, services) provided by the EFPF partners, their interconnections, dependencies (data, access, communications) and interfaces (both internal and external) to Data Spine and the EFPF platform.

One important objective of this deliverable is to provide necessary information to the EFPF project participants as well as external entities who might be interested in interlinking their tools/services through the Data Spine and making them part of the EFPF ecosystem.

The baseline architecture presented in this deliverable will be subsequently developed and enhanced as the project progresses and the updated architectural specification will be included in the next version of this deliverable at M18 of the EFPF project.

[1] https://www.nimble-project.org/
[2] https://www.composition-project.eu/
[3] https://www.digicor-project.eu/
[4] https://www.vf-os.eu/

Table of Contents

0 Introduction

0.1 EFPF Project Overview

EFPF – European Connected Factory Platform for Agile Manufacturing – is a project funded by the H2020 Framework Programme of the European Commission under Grant Agreement 825075 and conducted from January 2019 until December 2022. It engages 30 partners (Users, Technology Providers, Consultants and Research Institutes) from 11 countries with a total budget of circa 16M€. Further information: efpf.org

In order to foster the growth of a pan-European platform ecosystem that enables the transition from "analogue-first" mass production, to "digital twins" and lot-size-one manufacturing, the EFPF project will design, build and operate a federated digital manufacturing platform. The Platform will be bootstrapped by interlinking the four base platforms from FoF-11-2016 cluster funded by the European Commission, early on. This will set the foundation for the development of EFPF Data Spine and the associated toolsets to fully connect the existing platforms, toolsets and user communities of the 4 base platforms. The federated EFPF platform will also be offered to new users through a unified Portal with value-added features such as single sign-on (SSO), user access management functionalities to hide the complexity of dealing with different platform and solution providers.

0.2 Deliverable Purpose and Scope

The purpose of this document, "D3.1 EFPF Architecture-I", is to present an overview of the architecture of the EFPF ecosystem that is composed of the Data Spine, the EFPF platform, the four base platforms and external platforms. This document focuses on the architecture specification of two major components of the EFPF ecosystem: the Data Spine and the EFPF platform. It describes the architecture of the subcomponents of the Data Spine and of the tools and services that together constitute the EFPF platform. This deliverable presents the key concepts related to the methodology used to develop the architectural design of the EFPF ecosystem and its components. Finally the deliverable describes the components of the Data Spine and EFPF platform along with their responsibilities, interfaces, and interactions with other components.

This version of the deliverable presents the current state of the architecture of EFPF ecosystem and its components. These architectural specifications will be developed further, and the updated architectural specifications will be included in the next version of deliverable "D3.2 EFPF Architecture-II".

0.3 Target Audience

This document aims primarily at project participants and external entities that are interested in interlinking their tools/services through the Data Spine and making them part of the EFPF ecosystem. In addition, this deliverable provides the European Commission (including appointed Independent experts) with an overview of the underlying architecture of the EFPF platform.

0.4 Deliverable Context

This document is one of the cornerstones for achieving the project results. Its relationship to other documents is as follows:

- **D2.1: Project Vision and Roadmap for Realising Integrated EFPF Platform:** Provides an overview of the EFPF project and platform
- **D2.3: Requirements of Embedded Pilot Scenarios:** Provide an overview of the pilot requirements on the federated EFPF platform

0.5 Document Structure

This deliverable is broken down into the following sections:

- **Section 1:** **EFPF Architecture Methodology:** Presents the methodology used to develop the architectural design of the EFPF ecosystem and its components;
- **Section 2:** **EFPF Architecture Context View:** Provides an overview of the high-level architecture of the EFPF platform, the Data Spine and the federated ecosystem;
- **Section 3:** **EFPF Architecture Functional View:** Describes the architecture specification of two major components of the EFPF ecosystem: the Data Spine and the EFPF platform along with their subcomponents;
- **Section 4:** **Development & Deployment View:** Describes the deployment of EFPF tools, services and components and the monitoring of the configured components on the platform;
- **Section 5:** **Base Platforms:** Provides an overview of the base platforms

- **Annexes:**

  - **Annex A**: Document History
  - **Annex B**: References
  - **Annex C**: Platform Profiles
  - **Annex D**: Survey of Platforms for Realising Data Spine

0.6 Document Status

This document is listed in the Description of Action (DoA) as “public”. It presents the architecture of the EFPF Data Spine and the EFPF Platform. The architecture can be used especially by external entities to interlink their tools/services through the Data Spine and make them integral part of the EFPF ecosystem.

0.7 Document Dependencies

This document is one of the two deliverables that describe the architecture of the EFPF Data Spine and the EFPF Platform. This first deliverable at Month 9 of the EFPF project describes the baseline architecture of the EFPF ecosystem and its components. The second and final deliverable at Month 18 provides the final architecture.

0.8 Glossary and Abbreviations

A definition of common terms related to EFPF, as well as a list of abbreviations, is available in the supplementary and separate document “EFPF Glossary and Abbreviations”.

Further information can be found at https://www.efpf.org/glossary

0.9 External Annexes and Supporting Documents

Annexes and Supporting Documents:

- None

0.10 Reading Notes

- None

1 EFPF Architecture Methodology

This section presents the key concepts related to the methodology used to develop the architectural design of the EFPF ecosystem and its components i.e. the EFPF Data Spine and the EFPF platform.

Standards and best practices are followed as described in the following subchapters. In addition, there have been meetings of the project partners to discuss, produce and refine the architecture design.

1.1 Software Architecture Design Fundamentals

The process used to define the architecture of the EFPF Data Spine and the platform is based on IEEE 1471 "Recommended Practice for Architectural Description for Software-Intensive Systems" [Hil00] and ISO/IEC/IEEE 42010:2011 “Systems and software engineering - Architecture description” [IEEE42010, 2011], by which it was superseded. The latter establishes a methodology for the architectural description (AD) of software-intensive systems. It implies a process, which includes the following steps:

- Identify and record the stakeholders for the architecture and the system of interest
- Identify the architecture-related concerns of those stakeholders
- Select and document a set of architecture viewpoints which can address the stakeholder concerns
- Create architecture views (one view for each viewpoint) which contain the architectural models
- Analyse consistency of the views
- Record rationales for architectural choices taken

Viewpoints are collections of patterns, templates and conventions for constructing one type of view. One example is the functional viewpoint (and therefore a functional view) that contains all functions that the system should perform, the responsibilities and interfaces of the functional elements and the relationship between them. These functions can be described using UML diagrams. Moreover, it also describes which stakeholders need to be involved and how to apply their needs in the architecture as stated in the "Architectural Perspectives" chapter by Rozanski and Woods [RW05].

1.2 Definitions

The following definitions are based on the ISO/IEC/IEEE 42010:2011 [IEEE 42010, 2011] standard and the definitions provided by Rozanski and Woods [RW05]:

- **Architecture**: Comprises of the “concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution”
- **Architectural Description**: A collection of products to document an architecture
- **Stakeholder**: An individual, group or organisation that has at least one concern relating to the system-of-interest

- **Concern**: An interest in a system, which is relevant to one or more stakeholders. It might be a requirement (functional or non-functional) or an objective a stakeholder has regarding the system.
- **View**: A set of models and descriptions representing a system or part of a system from the perspective of a related set of concerns
- **Viewpoint**: Collection of patterns, templates and conventions for constructing one type of view
- **Model**: A simplified representation of an aspect of the architecture, could be in form of a UML diagram

The relationships between these concepts and the system-of-interests are shown in Figure 1.

According to the specification of the ISO/IEC/IEEE 42010:2011 standard, the main concepts, architecture view and architecture viewpoint, are defined as follows:

- **Architecture viewpoint**: "Work product establishing the conventions for the construction, interpretation and use of architecture views to frame specific system concerns"
- **Architecture view**: "A representation of a whole system from the perspective of a related set of concerns"

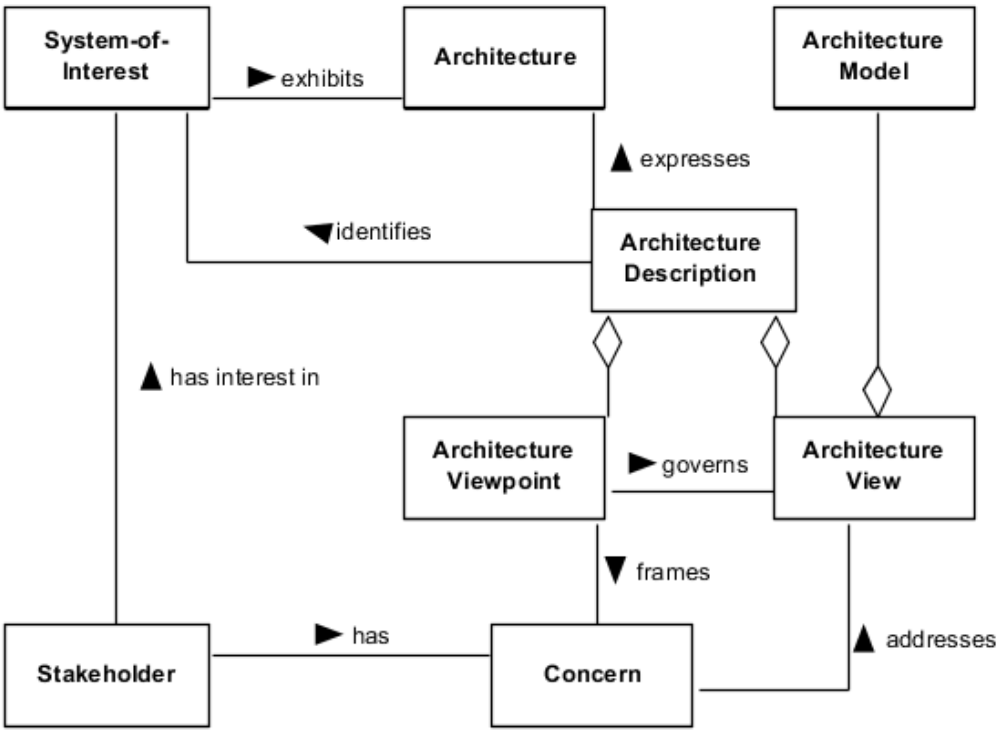

Figure 1: Architecture Description Concepts (Adapted from ISO/IEC/IEEE 42010:2011 "Systems and software engineering - Architecture description" [IEEE 42010, 2011])

A viewpoint defines the aims, intended audience, and content of a class of views and defines the concerns that views of this class will address e.g. functional viewpoint or deployment viewpoint. A view conforms to a viewpoint and communicates the resolution of a number of

concerns (and a resolution of a concern may be communicated in a number of views). According to [RW05], using vision and point of view to describe the system architecture can bring many benefits such as:

- **Separation of concerns**: Separating different models of a system into distinct (but related) descriptions helps the design, analysis and communication processes by allowing designers to focus on each aspect separately
- **Communication with stakeholder groups**: Different stakeholder groups can be guided quickly to different parts of the architectural description based on their particular concerns, and each view can be represented using language and notation appropriated to the knowledge, expertise, and concerns of the intended readership
- **Managements of complexity**: Treat each significant aspect of the system separately, the architecture can focus on each in turn and so help conquer the complexity resulting from their combination
- **Improved developer focus**: Separating those aspects of the system that are particularly important to the development team into different views, helps ensure that the right system is built

1.3 Software Architecture Design Process

In a software architecture design process, there are several principles that should be followed to ensure a high quality of the architecture design. The different stakeholders should be engaged and their concerns taken into account. There might be conflicting or incompatible concerns from different stakeholders, which must be dealt with. Architecture Definition Activities

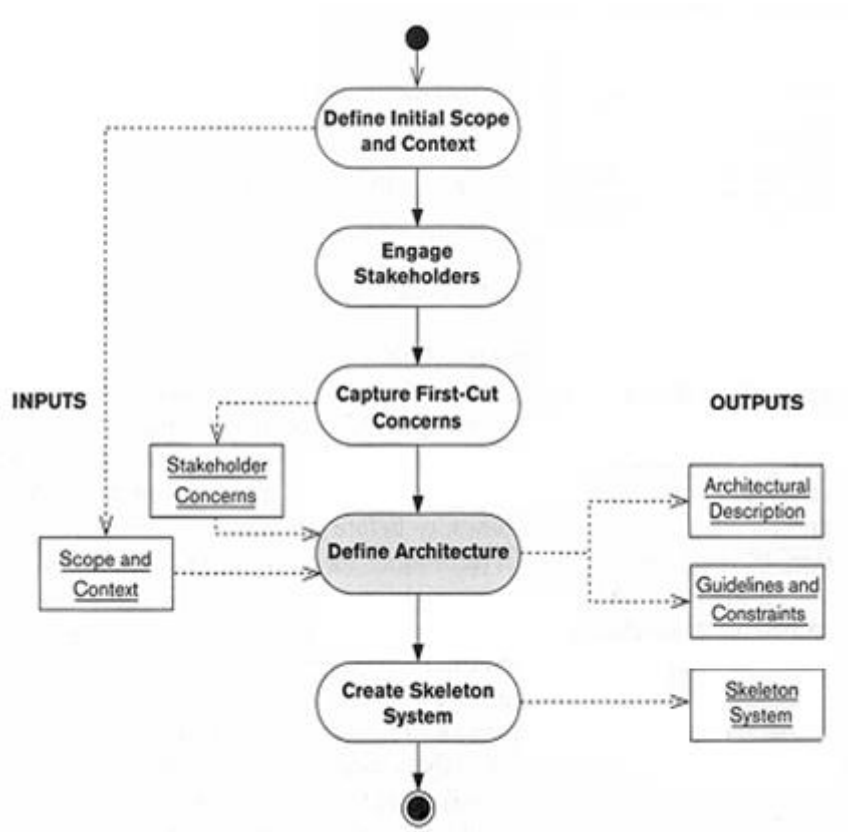

Figure 2: Activities Supporting Architecture Definition [RW05]

In addition, an effective way to communicate decisions and solutions should be implemented and the whole architecture design process should be flexible and pragmatic to be able to deal with the changing requirements and the iterative approach in this project. Also, the process should be technology-neutral.

Rozanski and Woods [RW05] have based the architectural design process on the following definition:

"Architecture Definition is a process by which stakeholder needs and concerns are captured, an architecture to meet these needs is designed, and the architecture is clearly and unambiguously described via an architectural description." [RW05] (p.56)

In EFPF, the foundation for the architectural definition process is the IEEE 1471 standard and the process that is aligned to this standard and proposed by Rozanski and Woods [RW05] has been followed (as shown in Figure 2). The EFPF project started with the definition of the initial scope and context of the EFPF ecosystem. This was followed by the involvement of stakeholders in the process of the scenario development and use cases description in WP2, and the subsequent requirements process. The stakeholders are included to express their needs and desires with regards to the EFPF ecosystem and the federation model adopted in the project to realise an ecosystem. The stakeholder input was also needed to capture quality properties that increase the success of the platform. The requirements from workshops, vision scenarios, and use cases, together with requirements from other sources (such as experiences of project partners) are processed as the input for the current architecture design phase, which has been reported in this deliverable. Based on this architectural description, the first prototypes are created, which can be seen as a

skeleton system with base functionality developed above that. These development efforts reveal some experiences and lessons learnt, which in turn constitute a valuable source for the derivation of additional requirements and the revision of the existing ones.

1.3.1 Viewpoint Catalogue

EFPF reflects on the following viewpoints, from which the views of the architectural document are derived:

- **Context viewpoint**: The context viewpoint describes interactions, relationships and dependencies between the system-of-interest and its environment. The environment includes those external entities with which the system interacts, e.g. other systems, users, or developers
- **Functional viewpoint**: This viewpoint describes the functional elements needed to meet the key requirements of the architecture. It will present proposals in a descriptive way and UML diagrams will assist in the understanding of the proposal. It will describe responsibilities, interfaces, and interactions between the functional elements
- **Information viewpoint**: The information viewpoint describes the data models and the data flow as well as its distribution. This viewpoint defines which data will be stored and where. The description of how data will be manipulated is part of this viewpoint too
- **Deployment viewpoint**: This viewpoint describes how and where the system will be deployed and what dependencies exist, considering e.g. hardware requirements and physical restraints. If there are technology compatibility issues, these can be addressed in this viewpoint as well
- **Development viewpoint**: This viewpoint addresses concerns from the developers' point of view. It describes how the software development process is supported, e.g. what conventions should be followed and how the artefact management will look like

To address quality properties and crosscutting concerns, architectural perspectives will be used. A typical example is security: It should be considered how the data is secured and which functional elements need to be protected. Another perspective is about availability of e.g. the hardware, the functional elements or the data.

2 EFPF Architecture Context View

Figure 3 gives an overview of the high-level architecture of the EFPF platform, the Data Spine and the EFPF federated ecosystem of base and external platforms. It provides a formal split of key components of the EFPF federation and depicts the interaction between them. In contrast to the high-level architecture diagrams shown in the DoA, the architecture in Figure 3 further details the composition and role of Data Spine in the EFPF ecosystem and its relationship with other components.

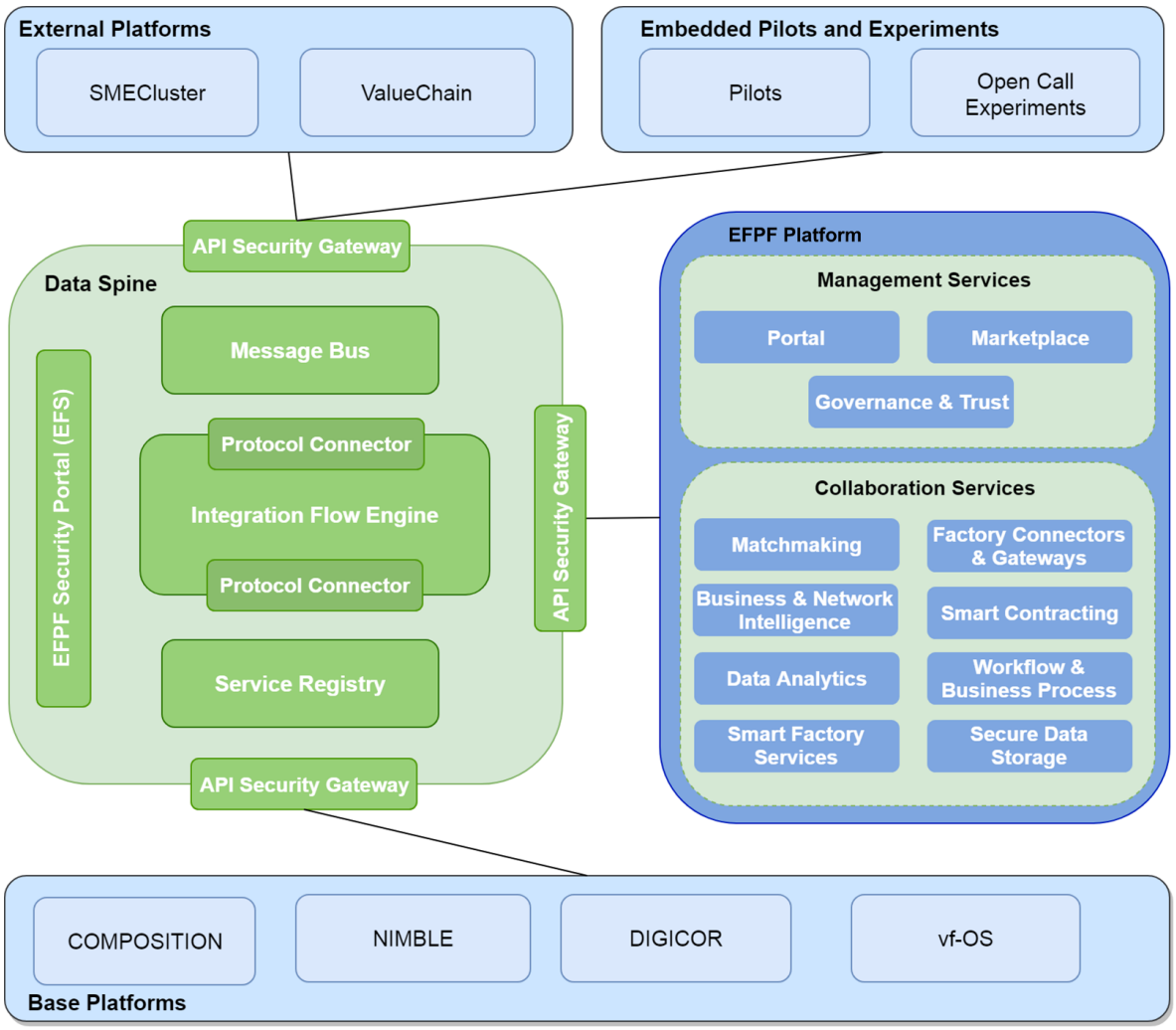

Figure 3 : High-level Architecture of the EFPF Ecosystem

The EFPF ecosystem follows the micro-service architecture approach in which different functional modules implement individual functionalities that can be composed based on specific user needs. In order to implement this approach, all components in the EFPF ecosystem are prescribed to implement and publish open interfaces, preferably REST interfaces (in case of synchronous communication), allowing the exchange of data and avoiding the lag-time introduced by interconnection buses.

The EFPF ecosystem is based on a federation model, which consists of distributed platforms, tools and components provided by several partners. The main elements in the EFPF federation are:

- **Data Spine:** This is the central entity or gluing mechanism in the EFPF federation. The Data Spine provides the interoperability infrastructure that initially interlinks and establishes interoperability between the four base platforms: COMPOSITION, DIGICOR, NIMBLE and vf-OS (see Section 5 for more details). It adheres to common industry standards and follows the micro-services pattern to enable the creation of a modular platform. Therefore, it can be easily extended beyond interconnecting the base platforms to 'plug' new external platforms in and interlink them with the existing platforms. Figure 3 also highlights the platform agnostic nature of the Data Spine i.e. it is evident from the high-level architecture that as far as interactions with the Data Spine are concerned, there is no distinction between the EFPF platform and the base platforms or any other platforms (external and third party). Thus, the Data Spine would be independent from the rest of the EFPF platform. This hypothetically means that even if the EFPF platform were 'switched-off' in the future, the Data Spine would not have to be 'switched-off' with it and therefore would continue to support an interconnected ecosystem
- **EFPF Platform:** This is a digital platform that provides unified access to dispersed (IoT, digital manufacturing, data analytics, blockchain, distributed workflow, business intelligence, matchmaking, etc.) tools and services through a Web-based portal. The tools and services brought together in the EFPF platform are the market ready or reference implementations of the Smart Factory and Industry 4.0 tools from project partners (see base platforms in Section 5). The collection of enhanced versions of such tools and services from the base or external platforms deployed together as micro-services would constitute the EFPF platform. These micro-services are made accessible through the EFPF Portal using the Single Sign-On (SSO) functionality offered by the EFPF ecosystem
- **Base Platforms:** The four base platforms (COMPOSITION, DIGICOR, NIMBLE and vf-OS) in EFPF are funded by the European Commission's Horizon 2020 program within the Collaborative Manufacturing and Logistic Cluster (FoF-11-2016). These base platforms are interlinked through the Data Spine that offers seamless interoperability of distributed tools and services by integrating, aligning and enhancing the open APIs of the existing platforms
- **External Platforms:** In addition to the four base platforms, the EFPF ecosystem enables interlinking of other platforms and open-source tools that address the specific needs of connected smart factories. The external platforms that joined the EFPF ecosystem at the beginning of the project are: ValueChain's iQluster platform[5] and SMECluster's Industreweb platform[6]
- **Pilots and Experiments:** These are the components and systems that will interact with the EFPF ecosystem (including the EFPF Platform and the Data Spine) during the course of the project

[5] https://valuechain.com/supply-chain-intelligence/iqluster
[6] https://www.industreweb.co.uk/

3 EFPF Architecture Functional View

The EFPF ecosystem consists of two major components: The Data Spine and the EFPF Platform, as illustrated in Figure 3. These components and their subcomponents, responsibilities, interfaces, and interactions with other (sub)components are described in detail in the following sections.

3.1 Data Spine

Data Spine is a collection of components that work together to form an integration, interoperability and communications layer for the EFPF ecosystem.

The following subsection introduces the Data Spine and its components.

3.1.1 Introduction

Data Spine is the interoperability backbone of the EFPF ecosystem that interlinks and establishes interoperability between the services of different platforms. The Data Spine is aimed at bridging the interoperability gaps between services at three different levels:

- **Protocol interoperability**: The Data Spine supports two communication patterns:
    a. Synchronous request-response pattern and
    b. Asynchronous publish-subscribe pattern

  While the Data Spine supports standard protocols that are widely used in the industry, it employs an easily extensible mechanism for adding support for new protocols
- **Data Model interoperability**: The Data Spine provides a platform and mechanisms to transform between the message formats, data structures and data models of different services thereby bridging the syntactic and semantic gaps for data transfer
- **Security interoperability**: The EFPF Security Portal (EFS) component of the Data Spine facilitates the federated security and SSO capability for the EFPF ecosystem

The process followed for the design of the conceptual components of the Data Spine included gathering of interoperability requirements from the base platforms and the external platforms to be integrated into the EFPF ecosystem. The technical profiles of these platforms were documented, which included the specific components from these platforms, their maturity level, exposed interfaces, protocols, data models, data formats, access control mechanisms, authentication providers supported, dependencies, programming environment, technical documentation, etc. The documented platform profiles are included in Annex C. Based on these technical profiles of the base and external platforms; the conceptual components of the Data Spine were defined.

Figure 4 depicts the architecture of the Data Spine showing a high-level conceptual view of the following core components that provide the expected functionality of the Data Spine:

- **The Integration Flow Engine** component of the Data Spine provides a platform to the system integrators, allowing them to create integration flows for interconnecting the different APIs and services
- **The Service Registry** component allows the service providers to register their services in the Data Spine. The Service Registry provides a facility for the service consumers or system integrators to discover these services and retrieve their metadata

information required to create the integration flows. The service providers can also write the integration flow specific information to the Service Registry

- **The Message Bus** component can be used for mediating the transfer of messages or data between the platforms or service communicating through the Data Spine
- **The EFS** component of the Data Spine is responsible for providing a SSO facility across the EFPF ecosystem (see API Security Gateway). In addition, the EFS component enables data integrity, security analytics, trust and reputation mechanisms, definition of policies and governance enforcement
- **The API Security Gateway** component of the Data Spine (and a sub-functionality of EFS) acts as the policy enforcement point (PEP) for the Data Spine and the platforms communicating through it. It intercepts all the traffic to the Data Spine and invokes the security service for authentication and authorisation decisions

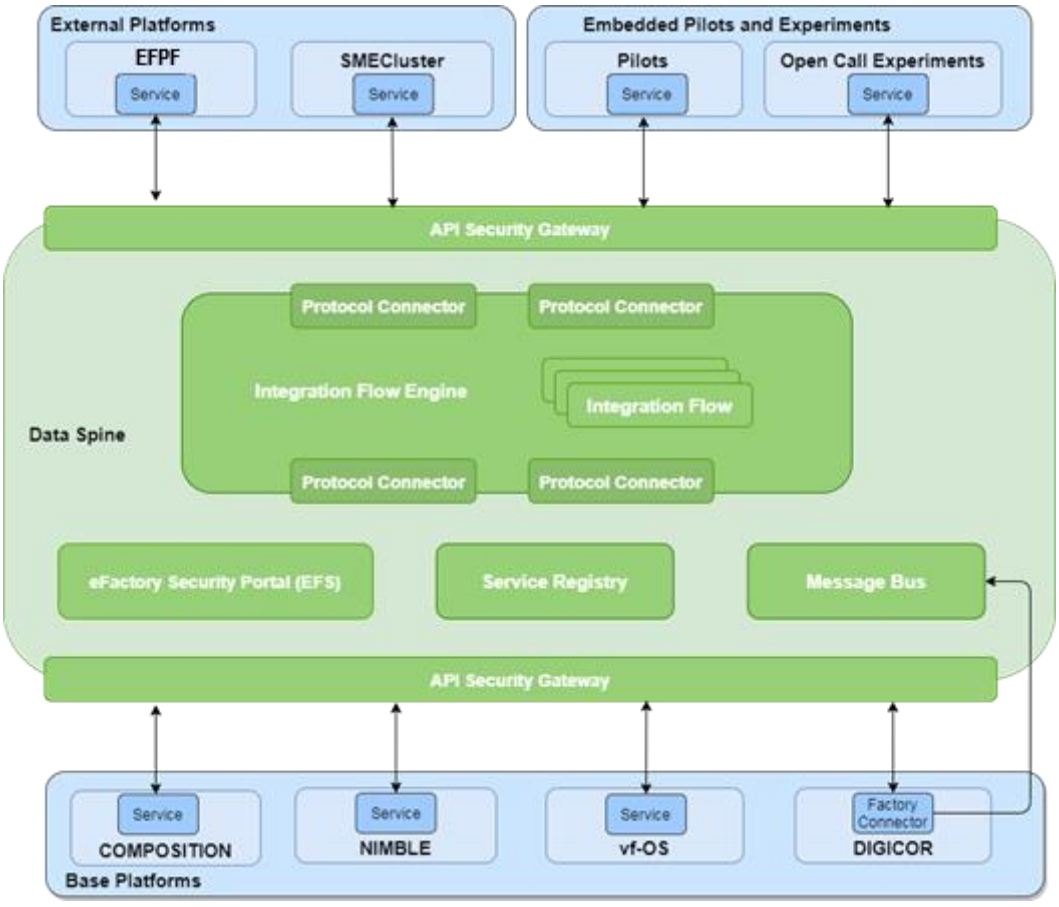

Figure 4: High-level Architecture of the Data Spine

3.1.2 Integration Flow Engine

The Integration Flow Engine component is a dataflow system that provides a platform to bridge the interoperability gaps at protocol level and data model level, between the heterogeneous services communicating through the Data Spine. The Integration Flow Engine follows the concepts from flow-based programming [Mor10] and visual programming [Shu86] paradigms to interconnect and interoperate between a particular pair of services

using a so-called 'Integration Flow'. The Integration Flows support connectivity, data routing, transformation and system mediation logic.

The Integration Flow Engine offers a drag-and-drop style Graphical User Interface (GUI) to the system integrators to create the integration flows. The integration flows are designed and implemented as directed graphs that have 'processors' at their vertices and the edges represent the direction of the dataflow. The processors are of different types depending upon the functionality they provide: The processors of type 'Protocol Connector' address the issue of interlinking the services that use heterogeneous communication protocols, the processors of type 'Data Transformer' provide means for transforming between data models and message formats, etc. The Processors are the extension points of the Integration Flow Engine. The Integration Flow Engine has in-built Protocol Connectors for standard communication protocols that are widely used in the industry. Support for a new protocol could be added through a new Protocol Connector to the Integration Flow Engine.

3.1.3 API Security Gateway

The API Security Gateway in EFPF acts as an intercepting proxy for the Data Spine. Figure 5 illustrates the flow of the EFPF token, sent from the EFPF Portal to the base platform:

1. API Security Gateway receives a `GET/efpf/item/uri=foo` request from the EFPF portal/EFPF client. This request contains the EFPF auth token provided by the EFPF IDentity Provider (IDP)
2. API Security Gateway requests EFPF Security (EFS) portal to validate the EFPF token
3. EFS validates the EFPF token
4. If the token is valid, EFS requests the base platform's token (e.g. from the NIMBLE platform) to authorise the request to the base platform. If the EFPF token is valid, steps below will be further executed. Otherwise an HTTP 401 unauthorised response will be sent
5. EFS retrieves the base platform's token
6. EFS sends the success (HTTP 200) response to the API Security Gateway with the relevant base platform's token
7. API Security Gateway decides which base platform's endpoint to use to route the original request (note that the endpoint discovery can be done via the Service Registry). The request is forwarded to the Integration Flow Engine (currently implemented through Apache NiFi) to perform any required extract, transform, and load (ETL) for the data model in the message
8. The Integration Flow Engine (NiFi) sends the requests to the base platform and retrieves the response

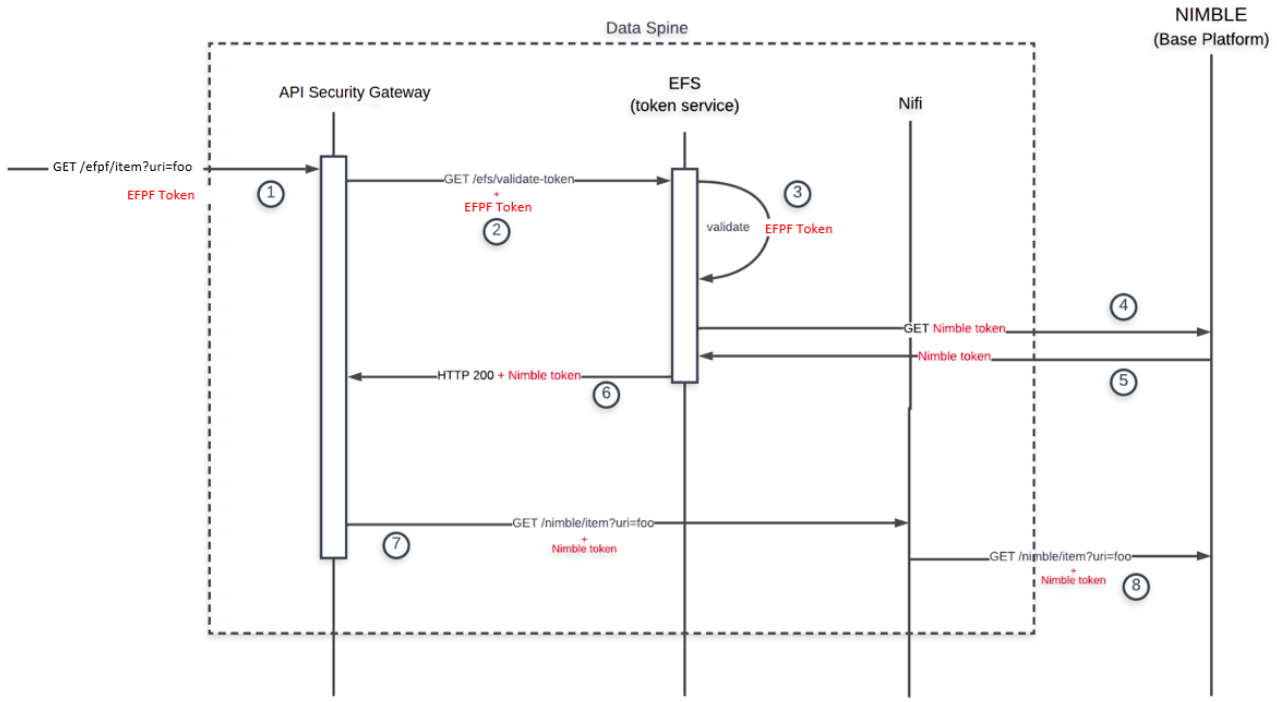

Figure 5: API Security Gateway Workflow Diagram

3.1.4 Service Registry

In an interconnected EFPF ecosystem, services of different platforms need to be orchestrated to achieve common objectives. In order for the services of one platform to discover the services of other platforms, the service providers should be able to advertise their services along with the associated metadata and make those discoverable for the potential service consumers and for the system integrators. The Service Registry component of the Data Spine fulfils this purpose. The Service Registry in EFPF is realised through the LinkSmart® Service Catalog (SC) technology. It provides a RESTful API for the lifecycle management of services and service discovery.

The rational for service registry comes from the fact that service consumers intend to search for services based on their functional specification whereas the system integrators are interested in retrieving the technical metadata of the services, e.g. protocols, endpoints, data formats, data models, etc. in order to write the integration flows. The EFPF Service Registry is capable of managing such heterogeneous metadata. In addition, its schema (see Figure 6) is capable of managing metadata for synchronous (request-response) as well as asynchronous (publish-subscribe) type of services.

```
{
        "id": "string",
        "type": "string",
        "title": "string",
        "description": "string",
        "meta": {},
        "apis": [{
                "title": "string",
                "protocol": "MQTT",
                "id": "string",
                "description": "string",
                "endpoint": "ssl://demo.linksmart.eu:8883",
                "spec": {
                        "type": "<document type>",
                        "url": "url to external document",
                        "schema": {}
                },
                "meta": {}
        }],
        "doc": "string",
        "ttl": 0,
        "created": "2019-08-09T15:46:36.793Z",
        "updated": "2019-08-09T15:46:36.793Z",
        "expires": "2019-08-09T15:46:36.793Z"
}
```

Figure 6: Service Description Schema of the Service Registry

Figure 7 shows the API of LinkSmart® Service Catalog.

REST Endpoint	HTTP Method	Description
/	GET	Retrieves API index.
/ {id}	POST	Creates new ‘Service’ object with a random UUID (Universally Unique IDentifier).
/ {id}	GET	Retrieves a ‘Service’ object
/ {id}	PUT	Updates the existing ‘Service’ or creates a new one (with the provided ID)
/ {id}	DELETE	Deletes the ‘Service’
/{jsonpath}/{operator}/{value}	GET	Service filtering API

Figure 7: LinkSmart® Service Catalog REST API

3.1.5 Message Bus

Some of the platforms in the EFPF ecosystem, interconnected through the Data Spine, offer their shop floor data as data streams through their factory connectors/gateways, while some of them offer control data that can be used to control the actuators installed in their factories. The Data Spine supports such asynchronous type of communication as well. In addition, the

Data Spine offers the Message Bus component to the publishers and/or the subscribers in these platforms. The Message Bus in EFPF Data Spine supports standard publish-subscribe-based messaging protocols such as MQTT, AMQP, etc. that are widely used in the industry. The Message Bus in EFPF could be extended to add support for new protocols via plugin mechanism.

3.1.6 EFPF Security Portal

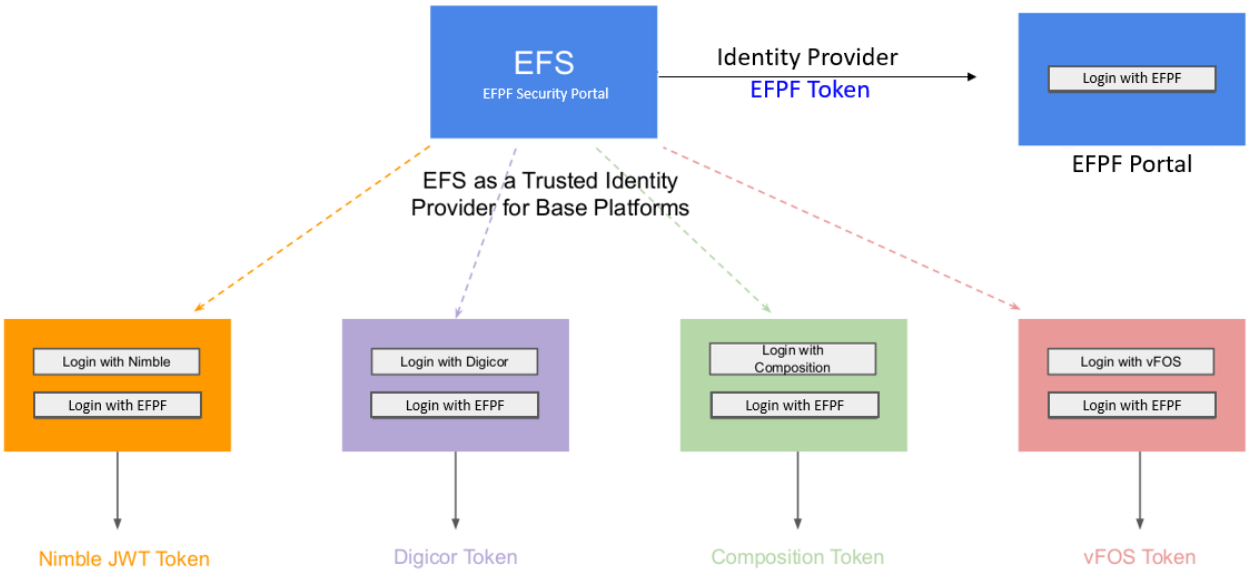

Figure 8: EFPF Security (EFS) Portal

To enable the federated identity features, the EFPF platform requires the EFS portal to be designed and implemented to efficiently govern the security management for various platforms, e.g. NIMBLE, COMPOSITION, vfOS and DIGICOR (see Figure 8). The EFS is a distributed single point of trust that enables a class of Super Administrator whose role is to provide secure authentication of any tenant platform in the ecosystem (e.g. multi identities to be managed across company's accounts). EFS hosts the components required for the identity and access management of the EFPF ecosystem and takes on the role of a trusted IDP for base platforms. In addition, EFS provides other security controls, e.g. data integrity, security analytics for risk identification, etc.

Figure 8 illustrates the Identity and Access Management setup of EFS. The user authentication protocol and workflows to enable platform federation are described in the following subsections.

3.1.6.1 Authentication Protocol

The EFPF Identity Protocol (IDP) uses OpenID Connect (OIDC) as the authentication protocol, which is an extension of OAuth 2.0. While OAuth 2.0 is only a framework for building authorisation protocols, OIDC is a full-fledged authentication and authorisation protocol. As all four base platforms in the project use Keycloak (www.keycloak.org), an open source software product identity and access management solution, as IDP to manage the authentication of the users. The EFS also uses Keycloak as IDP.

3.1.6.2 Authentication Workflow

The EFPF Portal acts as the entry gateway to the EFPF ecosystem. In order to provide access to the EFPF ecosystem, the users should register with the EFS portal that is based on the OIDC protocol. EFS makes heavy use of the JSON Web Token (JWT), which is an open standard (RFC 7519) that defines a compact and self-contained way to securely transmit information between parties as a JSON object. This information can be verified and trusted as it is digitally signed. JWTs can be signed using a secret (with the Hash-based Message Authentication Code (HMAC) algorithm) or a public/private key pair by using RSA encryption algorithm or Elliptic Curve Digital Signature Algorithm (ECDSA) digital signature algorithm.

The registered user will have a manual vetting process before being granted with user privileges to access the EFPF platform. The EFPF related roles and policy governance will be defined through the EFPF governance mechanisms. The successful login to EFS will generate a JWT for the user, which will be used by the EFPF Portal to perform API calls to the EFPF ecosystem.

3.1.6.3 User Federation with Base Platforms

To achieve user federation, the base platforms need to configure the EFPF IDP as the trusted IDP. A user's login interactions with the base platform and EFS can be performed using one of the following workflows:

- **Workflow 1** (see Figure 9) illustrates the bottom-up approach for user federated login procedure. In this approach, the user (represented by a blue circle) logs into one of the base platforms, e.g. NIMBLE, COMPOSITION, vf-OS, DIGICOR (PLATFORM 1 in the rest of this section) using his/her EFPF credentials. The user is redirected from the EFPF platform (PLATFORM) to the PLATFORM 1 with the authorisation code. The PLATFORM 1 requests from the user his/her token for the authorisation code, in order to match the user credentials and create the access for the user in PLATFORM 1.

Note Workflow 1 uses the OAuth2/OIDC protocol to enable federated access to the base platform.

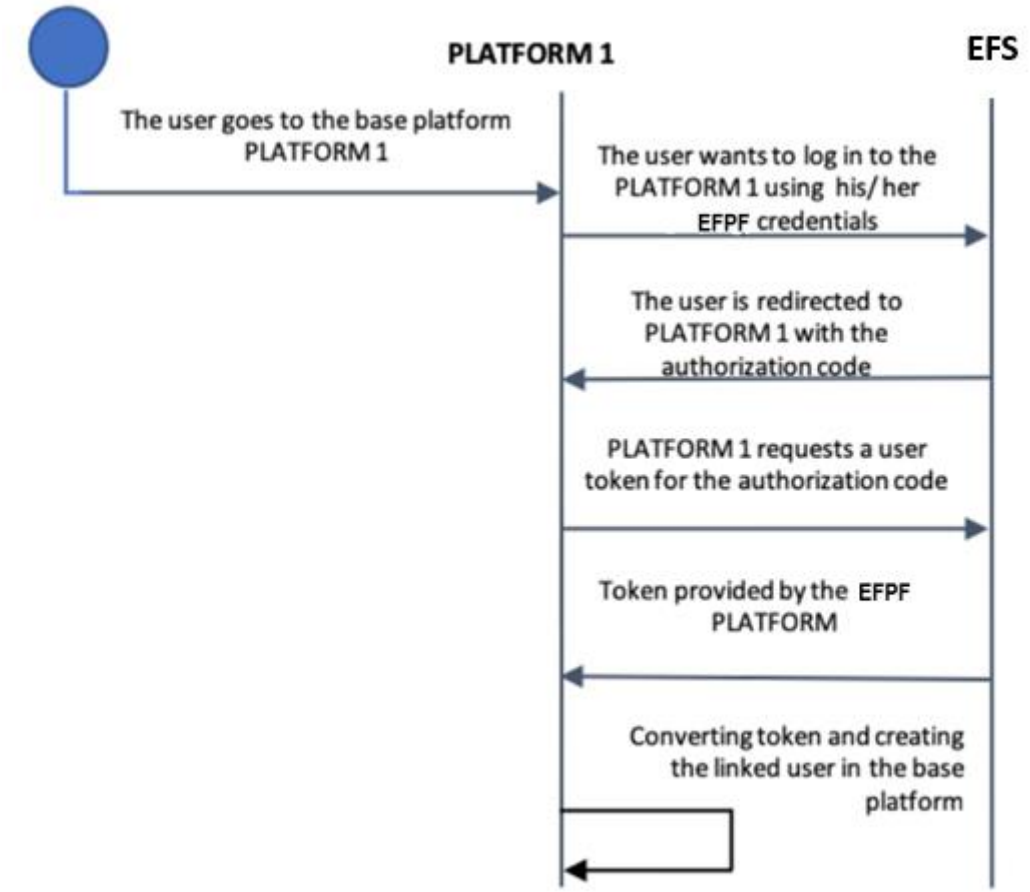

Figure 9: User Federated Login Procedure (The Bottom-up Approach)

- **Workflow 2** (see Figure 10) illustrates a top-down approach to user federated login. In this approach, the user (a blue circle) initially logins to the EFPF platform (via EFS) and then visits any base platform in the same browser session, e.g. PLATFORM 1, PLATFORM 2, etc. The user decides to log into the base platform using his/her existing EFPF credentials. The base platform e.g. PLATFORM 1 requests from the user his/her token for the authorisation code, in order to match the user credentials and create the access for the user in PLATFORM 1, and the user is allowed to access PLATFORM 1. Furthermore, the same user goes to PLATFORM 2, in order to log into this platform using his/her existing EFPF credentials. The PLATFORM 2 requests from the user his/her token for the authorisation, and creates the linked user in the PLATFORM 2. The user is now able to access PLATFORM 2.

Figure 10 illustrates the above described approach.

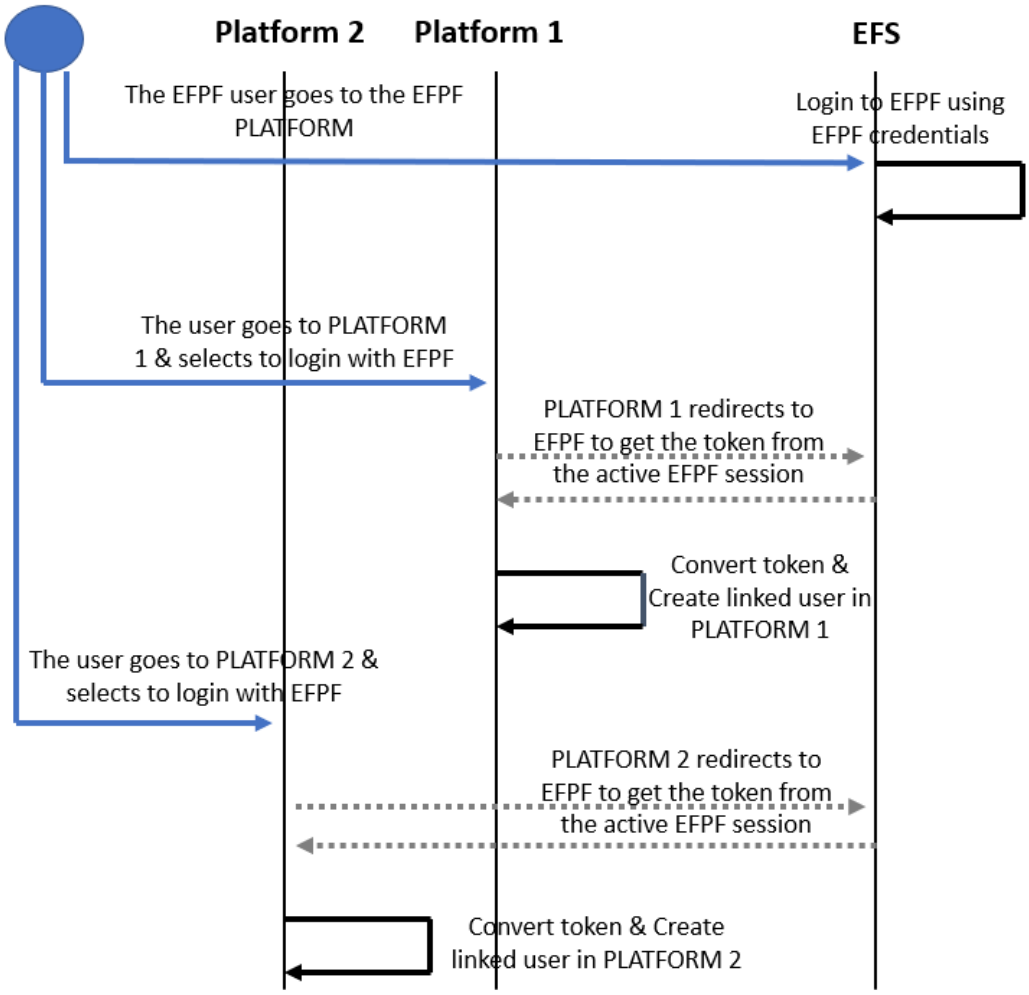

Figure 10: User Federated Login Procedure (The Top-down Approach)

3.1.7 The Realisation and Quality Assessment of the Data Spine

In order to realise the conceptual components of the Data Spine, available permissive open source technological platforms and frameworks were surveyed. The factors considered for the survey were Platform, License, Language, Plugin/Extension Mechanism, Supported Languages for Plugins, Hot Plugin Deployment, REST/API Management, Reverse Proxy Support, Identity and Access Management, Type, Message Bus and Relation to Data Spine conceptual component. The results of the survey are included in Annex D.

The survey of several candidate platforms based on the requirements above resulted into the identification of two possible solutions:

1. **WSO2 Carbon Infrastructure Stack:** The WSO2 Carbon Infrastructure Stack (WSO2 Platform) includes the components such as the Enterprise Integrator (WSO2 EI), the API Manager, the Governance Registry, the Identity Server and the Message Broker that could be used to realise the Integration Flow Engine, the Service Registry, the EFPF Security Portal service and the Message Bus of Data Spine respectively

2. **Apache NiFi with other components:** Apache NiFi is a Data-flow management tool based on the concepts of flow-based programming that could be used to realise the Integration Flow Engine of Data Spine and other components such as the LinkSmart Service Catalog or Consul, Keycloak, RabbitMQ, etc. could be used to realise the other components of Data Spine

Apache NiFi and WSO2 Platform with focus on WSO2 EI were shortlisted for an experimental evaluation. The objective of the experimental evaluation was the quality assessment of these solutions in order to realise the Data Spine. Along with the aspects/requirements mentioned above, the following factors are were considered for the experimental evaluation:

- **License**: The platform must have a permissive open source license (e.g. Apache License v2.0).
- **Usability**: The platform should offer an easy to use, intuitive and preferably Web-based drag-and-drop style GUI to the system integrators to create the integration flows with minimal effort.
- **Built-in functionality**: The platform should take care of the boilerplate code and should facilitate the system integrators to integrate their services by configuring only the service specific part of the integration flows with almost zero-coding effort.
- **Built-in Protocol Connector**: The platform should have built-in Protocol Connectors for standard communication protocols that are widely used in the industry.
- **Built-in Data Transformation Tools**: The platform should have built-in support for data transformation tools/languages, e.g. eXtensible Stylesheet Language Transformations (XSLT), that are widely used in the industry.
- **Extensibility**: The platform should offer flexibility to the developers for extending the platform by writing and plugging-in custom Processors.
- **Performance and scalability**: As all the communication in the EFPF ecosystem would pass through the Integration Flow Engine, the platform should be lightweight and scalable, e.g., it should be able to operate in a clustered fashion.
- **Identity and access management**: The platform should support a pluggable OpenID Connect provider such as Keycloak in order to connect to the EFS through the Data Spine and use the same user-base for authentication. In addition, it should have an in-built authorisation framework to control fine-grained access to its components such as the GUI, the REST API, etc.
- **Component integration effort**: The platform's integration with other Data Spine components should be as effortless as possible.
- **Maintainability**: An easy to deploy and maintain platform would be a bonus.
- **Documentation**: The platform should have a comprehensive documentation and an active user community.

The results of experimental evaluation of Apache NiFi and WSO2 Platform with focus on WSO2 EI are described below:

- **License**: Both the WSO2 Platform and Apache NiFi come with Apache License v2.0. WSO2 Platform also comes with a commercial license for receiving support from WSO2 development/support teams and access to WSO2 products' update service, etc.

- **Usability**: Both the WSO2 EI and Apache NiFi provide a drag-and-drop style GUI to the developers to create the integration flows with minimal effort. WSO2 EI's GUI is Eclipse-based whereas NiFi's GUI is Web-based. In the case of WSO2 EI, in order to create integration flows, a developer needs to install the Eclipse-based EI Tooling development environment and configure the WSO2 EI server. Once an integration flow has been created it can either be deployed directly from the EI Tooling if the remote WSO2 EI server is configured or it can be exported as a Carbon Archive file (WSO2's custom app packaging file) and uploaded through WSO2 EI's Management Console. Whereas, In the case of NiFi, in order to create integration flows, a developer can directly use NiFi's Web-based GUI through a Web browser.

  Also, learning curve for WSO2 EI and EI Tooling was observed to be much steeper than NiFi. The way in which service integrations are designed for synchronous and asynchronous services using the EI Tooling was observed to be completely different from each other that seemed to confuse the developers, which is not the case with NiFi.
  In addition, the collaboration of work concerning a particular workflow among different developers would be difficult in the case of WSO2 Platform as each developer needs to use the EI Tooling desktop application installed in the local environment, whereas in the case of NiFi, such a collaboration would be easy as NiFi provides a Web-based GUI for creating workflows and a Multi-tenant authorization capability that enables different groups of users to command, control, and observe different parts of the dataflow, with different levels of authorization.
  Therefore, NiFi was found to be better in terms of usability, developer productivity and ease of collaboration.
- **Built-in functionality**: Both the WSO2 EI and Apache NiFi provide a decent development environment for creating the integration flows with a drag-and-drop style GUI.
- **Built-in Protocol Connector**: Both the WSO2 EI and Apache NiFi provide connectors for standard communication protocols such as HTTP, MQTT, AMQP, etc. that are widely used in the industry.
- **Built-in Data Transformation Tools**: WSO2 EI provides several components, which in the architectural solution of WSO2 EI are called mediators, for mapping one data model to another using translation rules. The WSO2 EI mediators that have been tested are: Data Mapper, Payload Factory, XSLT and Script. All the aforementioned mediators achieved the intended functionality but each one with some minor drawback that depended on the use case in which they were used. The major drawback with WSO2 EI was lack of flexibility that WSO2 EI has, in fact it seemed complicated to extend the functionality provided by the system. The only mediator which was flexible enough: the "Script Mediator", which allows to programmatically access the message and modify it requires the restarting of the server with each deployment of the script. The only positive point of the WSO2 EI regarding the mediators was the simple, and mostly guided, UI which make the system more usable for a semi-technical person, but this is only applicable to very simple data transformations that do not include complex elements like arrays, for example.
  In comparison, NiFi provides less number of components, called processors, for the same purpose of data mapping. These data transformation processors in NiFi are: JoltTransformJSON, TransformXml and ExecuteScript. Apart from the steeper learning

curve, the processors look more flexible and manageable. For example, the processor ExecuteScript doesn't need the server to be restarted, by only updating the processor itself, making it easy to develop and debug. JoltTransformJSON is a processor which uses JOLT transformation rules to transform one JSON data model to another JSON data model (the mapping we intend to use) and in order to use this processor it is necessary to learn the rules of the transformation-language, resulting in a steeper curve for just transforming between JSON formatted data models. Apart from the previous drawbacks, NiFi with its less graphical UI, seems to be aimed at a more technical user, giving more power and responsibility to the user.
Moreover, all the aforementioned NiFi components are able to achieve the data transformation tasks in a flexible way, rather than the higher number of WSO2 EI mediators that are mostly non applicable to the identified purposes because of a lack of flexibility.

- **Extensibility**: NiFi is at its core built with extensibility in consideration. Points of extension include: Processors, Controller Services, Reporting Tasks, Prioritizers, and Customer User Interfaces. For example, it is possible to write a custom processor for NiFi in order to connect to an OPC-UA server (based on OPC-UA Java Stack) and read the data. WSO2 also has support for extension through custom mediators; however, WSO2 Platform's documentation recommends to avoid using the custom mediators as they incur a high maintenance overhead and they might also introduce version migration complications when upgrading WSO2 EI to a new version.
- **Performance and scalability**: Heavy resource utilization was noticed for WSO2 EI and WSO2 Message Broker whereas NiFi was observed to work seamlessly with the same amount of resource allocation. NiFi is also able to operate within a cluster.
- **Identity and access management**: NiFi supports a pluggable OpenID Connect based authentication provider such as Keycloak. Alternatively, NiFi also supports user authentication via client certificates, via username/password with pluggable Login Identity Provider options for Lightweight Directory Access Protocol (LDAP) and Kerberos or via Apache Knox. WSO2 Platform offers its own solution for identity and access management called WSO2 Identity Server.
- **Component integration effort**: It was experienced during experimental evaluation that the integration of WSO2 EI with WSO2 Governance Registry needs high integration effort, even though these components belong to the same software stack.
NiFi provides connectors for integration with external components. E.g., for integration with Kafka, NiFi has 20 built-in processors. Integration of NiFi with REST APIs of other components such as EFS was done with minimal effort.
- **Maintainability and Documentation**: During the experimentation, the WSO2 EI was evaluated with focus on Message Brokering. The WSO2 EI was installed on Ubuntu 16.04 TLS over Openstack. During the tests, a few times the software became slow, stopped working and crashed; since troubleshooting and maintaining is difficult, usually reinstallation was needed. As for Message Brokering for MQTT protocol, the tests were successful but connecting the WSO2 components to an external broker failed. For this problem, referring to the documentation wasn't helpful because at the first glance, the documentation looked very detailed and completed, but at the moment of need, it was very difficult to find solutions. The documents are not up to date; the structure is complicated and different versions of WSO2 Platforms have overlapping

documents. Since the WSO2 Platform has so many components and features the learning process needs to be easier and smoother.
Unlike the WSO2, NiFi's GUI is very simple, drag-and-drop style and easy to manage and as for documents, they are simple and useful.

- **Other factors**: Some inconsistencies were observed with WSO2's Management Console. E.g., when simple workflows were created using the Management Console's UI, they were not listed on the page that displays all workflows. Also, WSO2 EI internally makes use of SOAP (Simple Object Access Protocol) which incurs unnecessary overhead as most of the synchronous communication in EFPF ecosystem happens using REST over HTTP. No such issues were observed with NiFi.

Based on this experimental evaluation for quality assessment, Apache NiFi was selected to be the central Integration Flow Engine of the Data Spine.

3.2 EFPF Platform

The following subsections present a detailed overview of the components of the EFPF platform. The subsection 3.2.1 introduces the EFPF platform and its components, and the subsequent subsections present the architecture of components and describe the functionalities provided by each component.

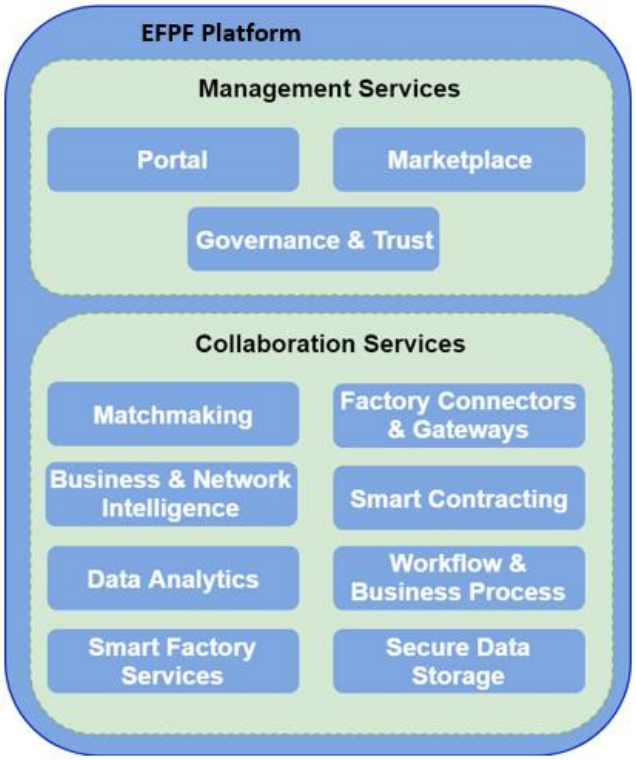

Figure 11: The EFPF Platform

3.2.1 Introduction to the EFPF Platform

The EFPF platform, shown in Figure 11, is a collection of smart tools and designed for the EFPF ecosystem. These tools and services aim to cover the complete lifecycle of production and logistic processes that will be validated by the three cross-domain pilot scenarios

brought forward by the project partners. Examples of the tools include e.g. data gateways, distributed production planning and scheduling, distributed process design, monitoring, decision support, process optimisation, risk management and blockchain based trust and message exchange. In future plans (e.g. through open experimentation calls), the platform may also integrate services that play a crucial role in collaborative processes. Examples include technology services such as cloud storage, high performance computing, and value-based services, e.g. training, smart contracts, legal advice etc.

The EFPF platform aims to integrate market ready or reference implementations of the smart factory and Industry 4.0 tools from project partners. Figure 11 illustrates tools and services that are broadly categorised into two types:

- **Management Services**: These are the EFPF-specific services that provide federated or aggregate management capability of the services of different platforms and provide a coherent interface to the user, e.g. the Marketplace.
- **Collaboration Services**: These are the utility services with a concrete capability for realising collaborative processes in IoT, connected factory and automation environments, e.g. the Data Analytics services.

3.2.2 Portal

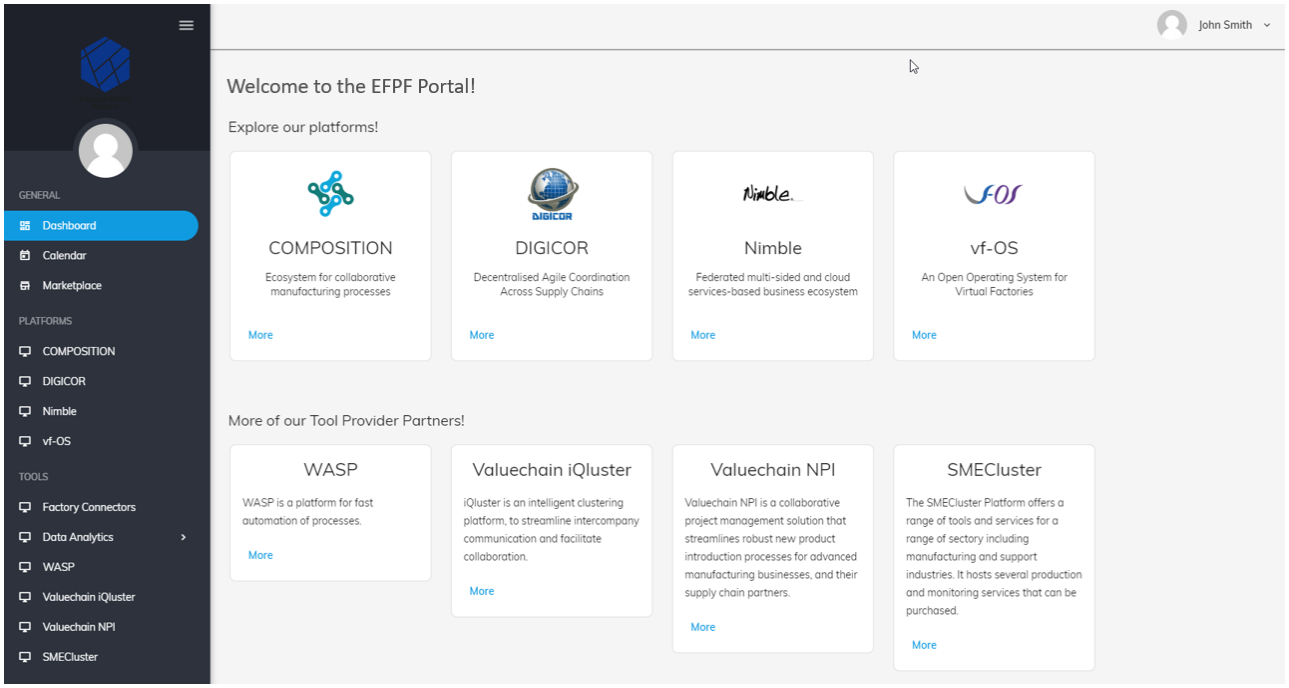

Figure 12: Snapshot of EFPF Portal Dashboard

The EFPF Portal component is the unification point of distributed tools and platforms in the EFPF ecosystem. It allows the user to access to connected tools, base platforms, marketplaces, experiments and pilots through a unified interface. The EFPF Portal is accessible at: https://efpf-portal.ascora.eu/,

The EFPF Portal and the communication to connected tools are secured by EFS, which is a part of the Data Spine (see Section 173.1). New users and visitors access a landing page that describes the EFPF platform and services. Additionally, the landing page allows existing

users to login into the EFPF platform and visitors to start the registration process to retrieve an account and become a member of the EFPF platform.

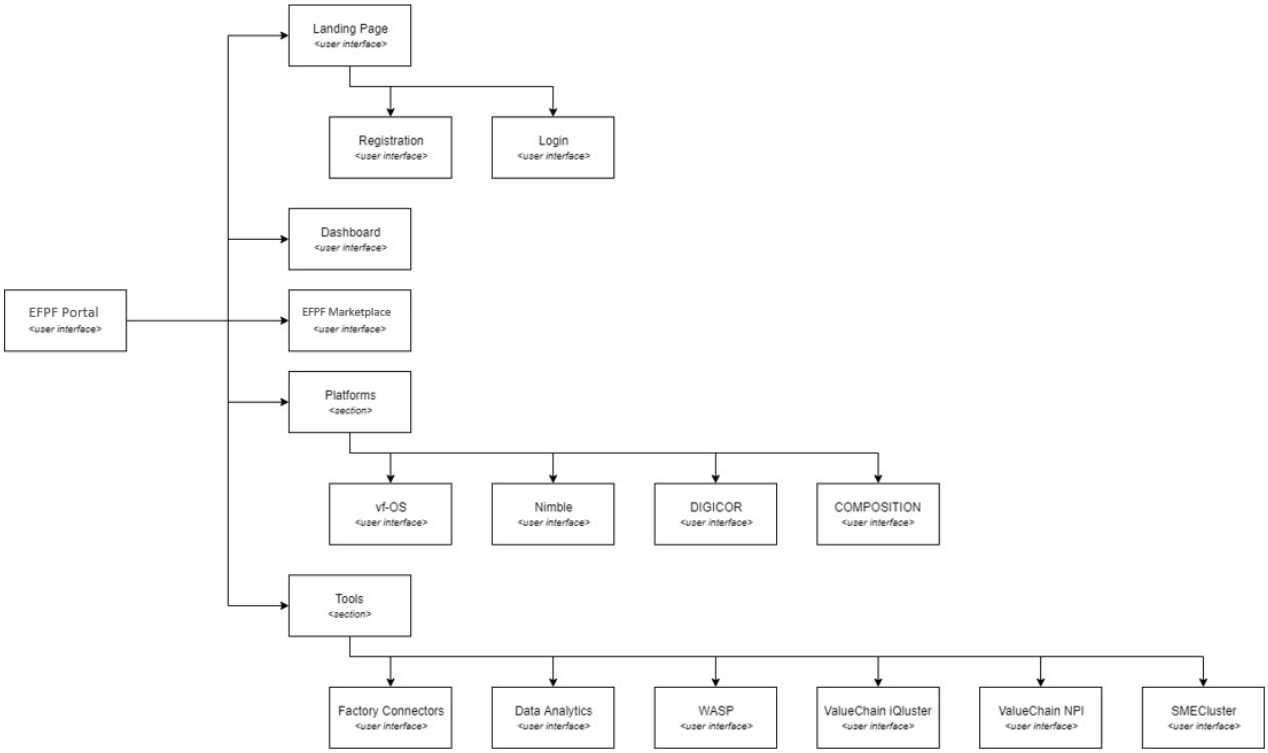

Figure 13: EFPF Portal Structure

The EFPF Portal structure shows currently available tools and four base platforms, besides generic sections like the Dashboard or the landing page. The Marketplace (see section 3.2.3) allows unified access to marketplaces from connected platforms. Additional entries for tools or platforms provided by the EFPF partners will be added in the future.

Technical Foundation: The EFPF Portal is a single-page application (SPA) based on the Angular web application framework[7] and can be deployed as a Docker[8] container. The system is currently available at https://efpf-portal.ascora.eu/[9].

[7] https://angular.io/
[8] https://www.docker.com/
[9] Subject to changes

3.2.3 Marketplace

3.2.3.1 Marketplace Overview

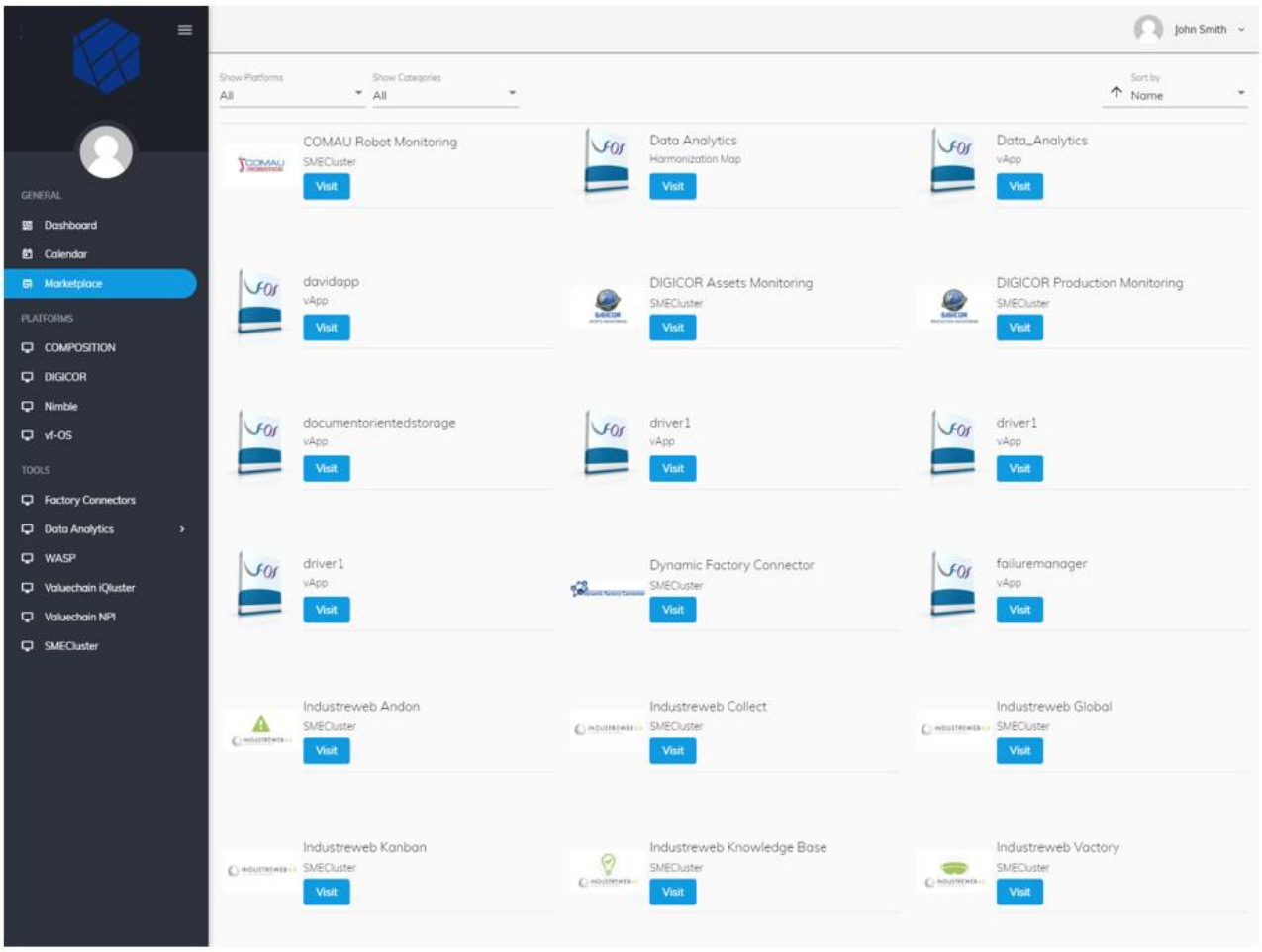

Figure 14: Snapshot of the EFPF Marketplace

The Internal Marketplace Framework provides access to items listed on marketplaces at different platforms provided by the EFPF partners. Additionally, its accountancy Service subcomponent provides features to track & trace and credit users of connected marketplaces, (more information in section 3.2.3.2).

Currently the marketplaces of the following platforms are linked with the EFPF Marketplace Framework (see Figure 14):

- vf-OS
- NIMBLE
- SMECluster
- COMPOSITION

The EFPF user can utilise integrated filter and sorting features to customise the view. To get more information and options, a click on each of the listed products will redirect the user to the connected marketplace where any transactions can be made.

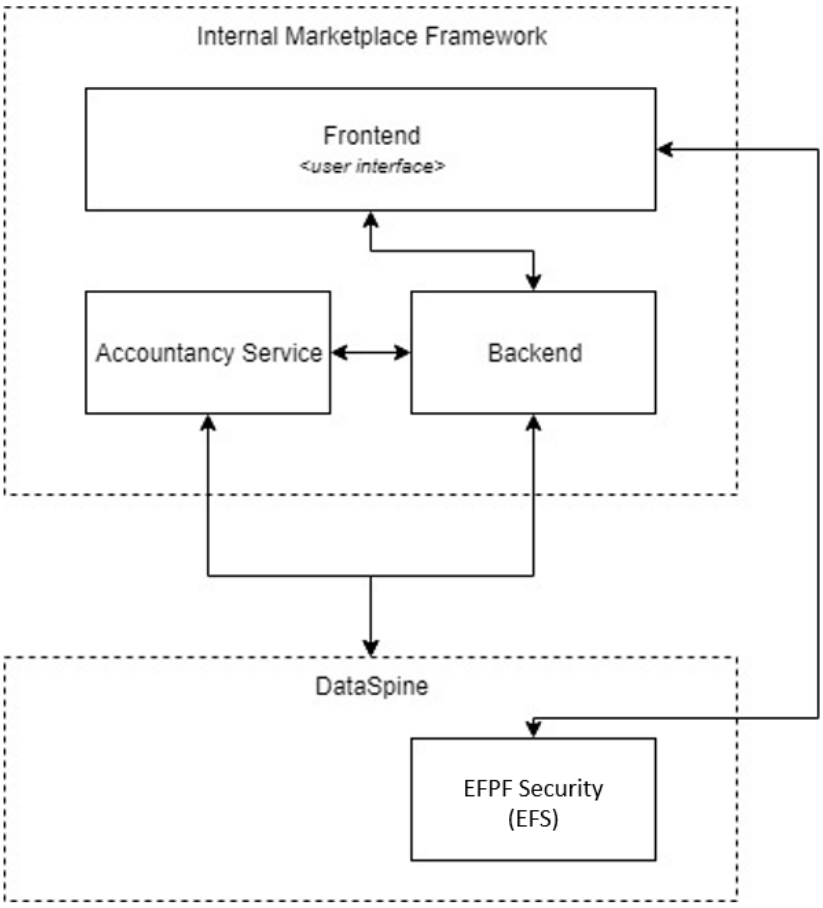

Figure 15: Marketplace Architecture

The architecture of the Marketplace component includes a frontend and a backend that assimilates data to be shown in the frontend. The subcomponent Accountancy Service communicates with base platforms via the Data Spine to track, trace and credit users coming from the EFPF Portal.

Technical Foundation: This component is based on Angular web application framework and is being delivered as a web component[10] to enable reuse in other platforms.

[10] https://www.webcomponents.org/introduction

3.2.3.2 Accountancy Service

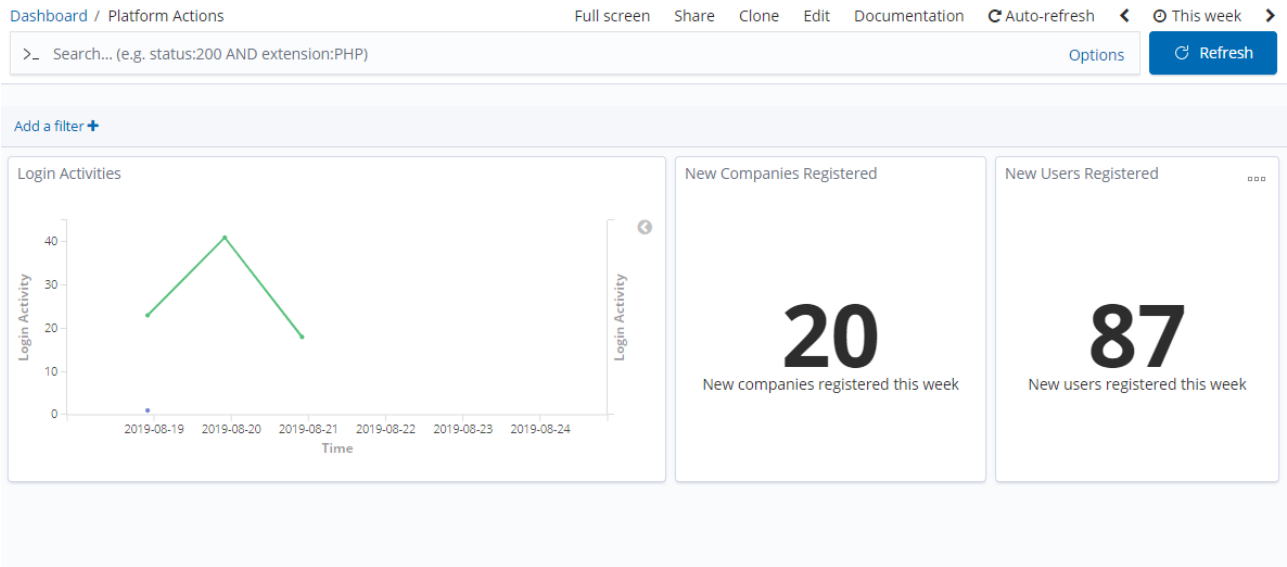

Figure 16: Accountancy Service Dashboard

The Accountancy Service is developed as a part of the EFPF Marketplace Framework and provides insight into users' interactions with the EFPF Platform, particularly any transactions that EFPF users make on different marketplaces, which are linked with the EFPF Marketplace Framework.

Tracking the user behaviour enables businesses to make productive decisions and develop effective business strategies. This is an important feature in the digital platform world, which is being used to support the long-term sustainability of the EFPF platform, beyond the span of the project. The Accountancy Service tracks and traces users' journey across the EFPF ecosystem and collects data about the transactions that EFPF users make on different marketplaces. The collected data will be used to charge a commission or referral fee from the marketplace where the EFPF user carries out a business transaction. Note, the Accountancy Service does not collect personal and/ or sensitive corporate data, instead the idea is to collected anonymised transactional data. In addition, the Accountancy Service contains a dashboard for the visualisation of the user behaviour.

A taxonomy is setup to identify the trackable user actions in which action items are listed in 'subject, verb, object' manner and these actions are grouped into two categories: 1) Platform Actions and 2) Business Actions. Platform Actions are users' basic interactions with the EFPF Platform such as login, register, inviting other users, while Business Actions include transactions realised between two companies such as product or service purchases, business message exchanges (e.g. Order, Invoice), etc.

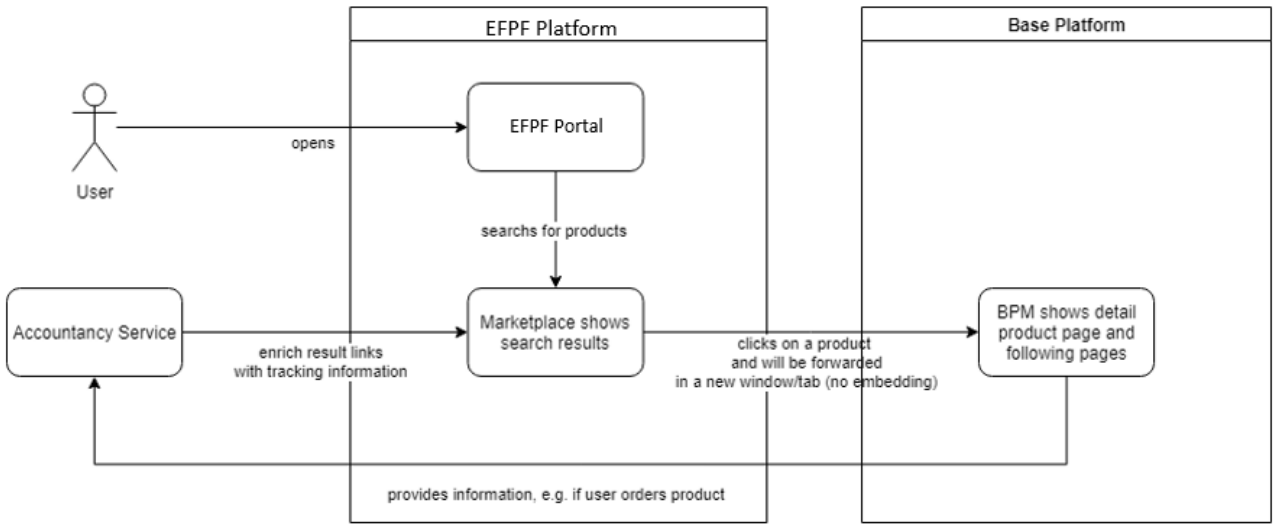

Figure 17: Cashback process

The EFPF Platform adopts a cashback mechanism (see Figure 17) that calculates the commission/referral amount paid back to the EFPF platform as a percentage of the overall transaction that an EFPF user carried out on a marketplace.

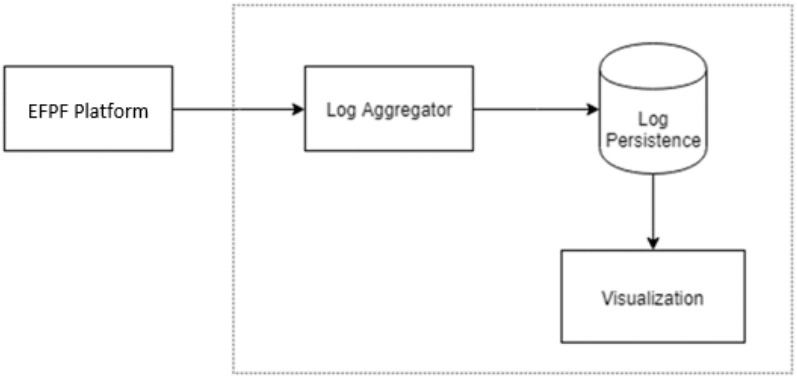

Figure 18: Accountancy Service Architecture

The Accountancy Service architecture consists of the following modules:

- **Log Aggregator**: It gathers user behaviour data from various components of the EFPF Platform, executes different transformations and filters the content, before sending the data to the Log Persistence component
- **Log Persistence**: It stores, indexes, retrieves and manages user logs to be later analysed. Since relational databases are not well-suited for managing log data, a NoSQL database is preferred due to their flexible and schema-free document structures, enabling analytics of the log data.
- **Visualisation**: It enables interactive dashboards, filters and advanced data analysis and exploration of user logs.

3.2.4 Matchmaking

The matchmaking and agile network creation mechanisms will be implemented for inter-platform and cross-domain scenarios in EFPF. The matchmaking process is divided into 3 steps:

- Federated search, e.g. search for a partner or a product/service
- Recommendation service/ matching algorithm
- Service that enables the user to start and perform a business transaction e.g. a bidding process

In the first step, the user of the EFPF platform is able to search for partners across the base platforms, based on different criteria e.g. capabilities, geographic locations and feedback/rankings or search for products and services. In the second step, the user is able to get relevant recommendations for products/services or partners based on different techniques of information pattern-matching. These techniques include information retrieval techniques and similarity matching techniques and are based on Machine Learning (ML) and data analysis. After finding the suitable products/services or partners, the users will be able to evaluate them for several aspects/indicators, e.g. cost, reliability, quality, etc. Finally, in the third step, the user decides how to proceed with business transaction execution.

3.2.4.1 Step 1: Federated Search

The goal of a federated search solution in EFPF is to enable (partner) search functionality over multiple sources (platforms) in the EFPF ecosystem, using one query. The architecture for federated search and recommendation is derived considering the existing base-platform's features, data sources and other architectural requirements for a recommendation engine. The selected architecture for federated search in EFPF follows the index-time merge approach. It requires content from base platforms to be acquired into a central index at the EFPF platform level, in order to perform platform level search for products/services and partners/companies across the base platforms in the ecosystem.

The index-time merge architectural approach is also used to implement traditional enterprise search systems, in which information can be retrieved across heterogeneous data sources in an enterprise. Figure 19 depicts the index-time merge architecture for federated search.

Advantages of the architecture (shown in Figure 19) are as follows:

- Most search engines perform ranking by relevance, which is what users usually expect. Through acquiring all data into a central index, sophisticated query enhancement and relevancy algorithms can be applied, providing the user with excellent search results.
- The indexed data and ML algorithms can be used to provide product/services/partners recommendations

Disadvantages of the index-time merge architecture can be summarised as follows:

- Acquiring the content from the various repositories and data sources of the base platforms requires considerable efforts, as it would need to be done via read-only processes on a schedule implemented in the data integration layer
- For different type of data sources, such as Resource Data Framework Schema (RDFS), additional data connectors need to be implemented to enable the data integration

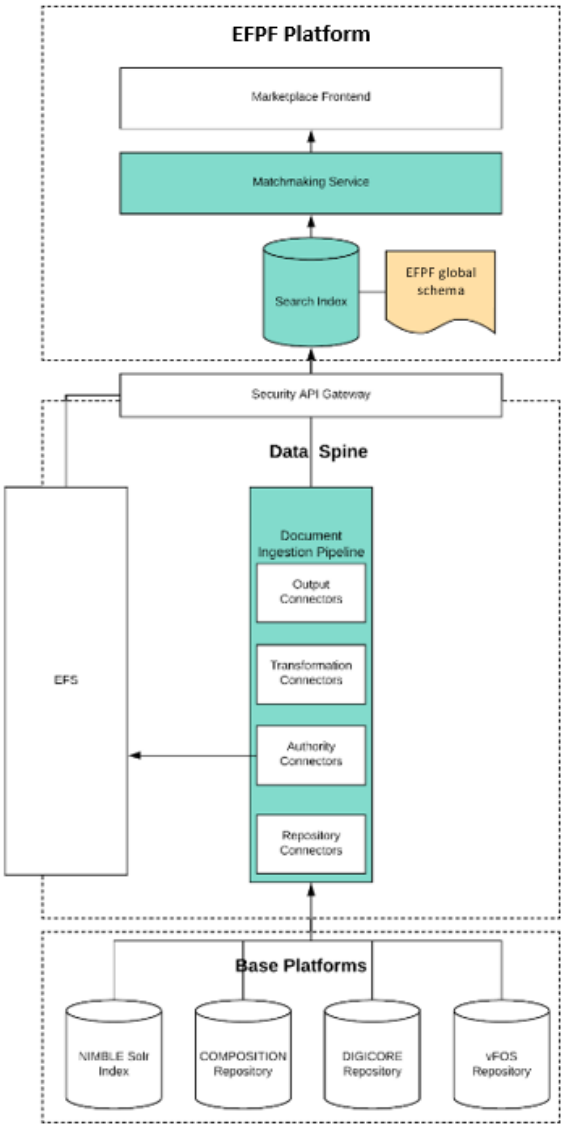

Figure 19: The Index-Time Merge Architecture

In the following, the major technical decisions for the implementation of the federated search architecture in EFPF are summarised:

- **Index-Time Merge** will be used as the federated search approach, which allows for flexibility in terms of search functionality and next level of matchmaking/ recommendation engine implementation
- **Apache Solr** will be used as the central search-index implementation
- A common EFPF search ontology/ Solr-schema needs to be defined to capture all required information about participants and value-units (domain knowledge)
- **Apache NiFi** will be used as the technology to implement the data integration flows from base platform's data sources
- Base platform technical partners will implement/configure input connectors for NiFi, to ingest data to the central index

- Base platform technical partners will configure the frequency/schedule of the data ingestion flow for their base platforms. Data ingestion frequency can be configured hourly, daily or weekly, depending on the data velocity of the base platform

Figure 20 illustrates the high-level view of the EFPF Manufacturing Ontology to be developed to enable an effective federated search in EFPF, that includes the following concepts:

- A class/category of a product/service/capability has 0 or more properties
- A property describes the product/service class in detail (eg: length, height, certificates etc)
- Each class has 1 or more item instances. These items represent the actual product/service or capability that will be manufactured/provided by a party/company
- A party has attributes such as its legal-name, keywords and activity sector giving more attributes for the matchmaking to execute

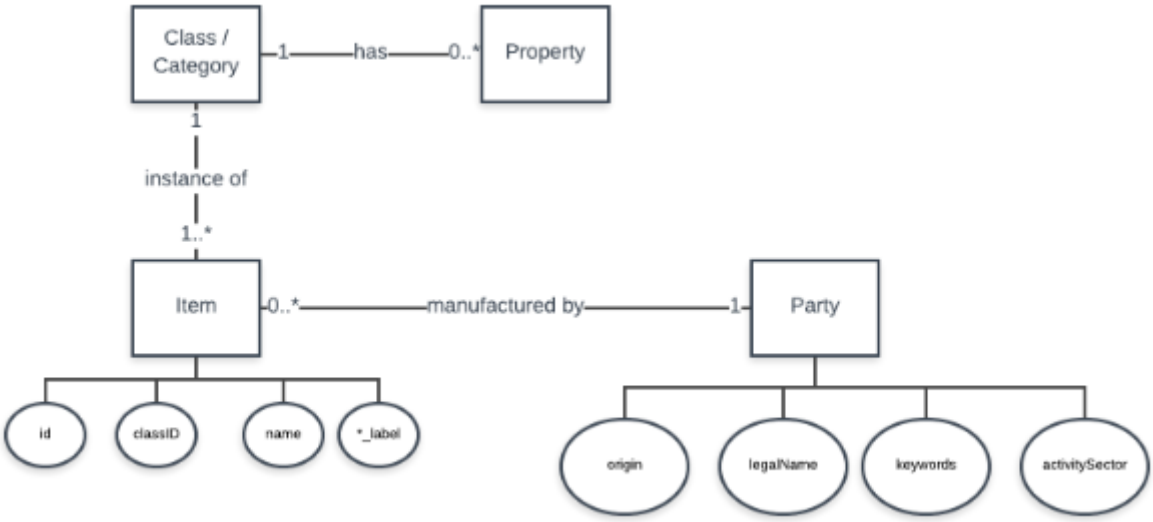

Figure 20: High Level View of the EFPF Manufacturing Ontology

3.2.4.2 Step 2: Recommendation System

The second step of the matchmaking process is about enabling effective recommendation services for products/services or partners to be used to initiate transactions. Following diagram (Figure 21) gives a high-level design view of the recommendation system in EFPF.

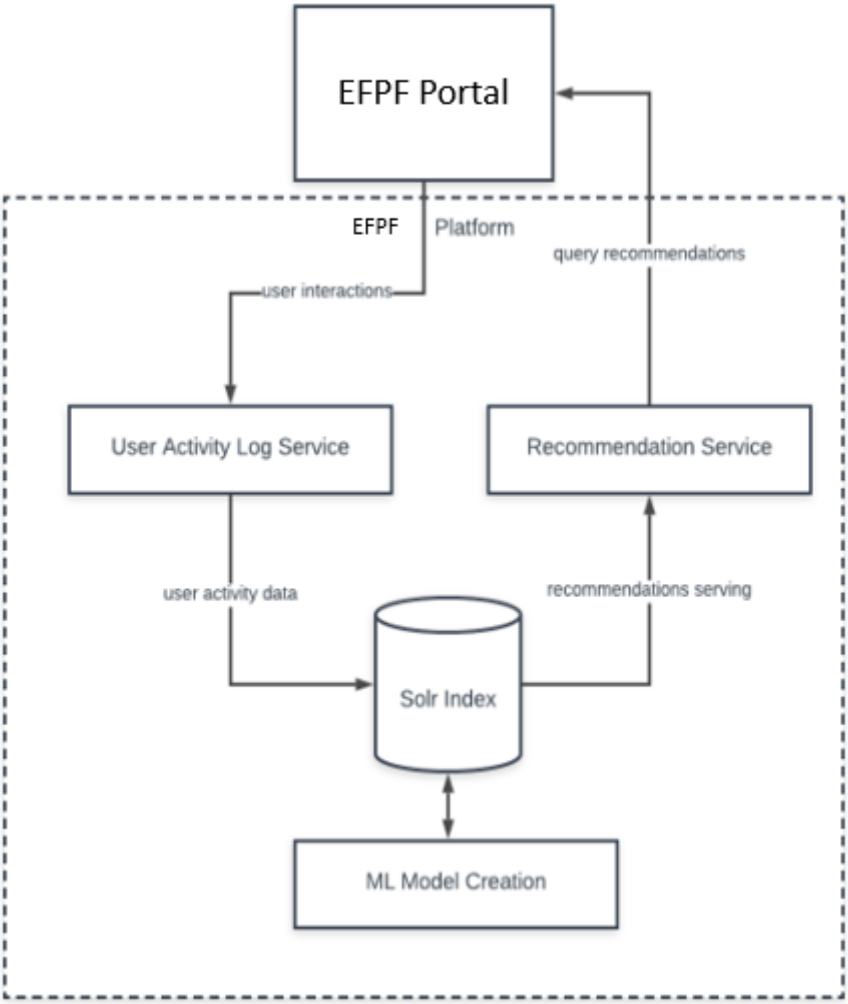

Figure 21: High Level View of the EFPF Recommendation System Enabling Matchmaking Features in EFPF

The EFPF recommendation system includes the following components:

- **EFPF Portal**: EFPF Portal (see Section 3.2.2) is the application where users search for products/services/partners
- **User Activity Log Service**: This component listens to user interaction events generated from the applications (EFPF Portal) and stores the user interaction data, e.g. item views and purchased items, in a way convenient for ML model creation
- **Solr Index**: In addition to maintaining the data about products/services and partners, the system will also store all the user interactions data, which will be used for ML model creation
- **ML Model Creation**: An ML library (Apache Mahout) will be used to create item-similarity and user-similarity-based recommendations for users, including history of user's activity as a data source
- **Recommendation Service**: The ML models will be served via a REST API on top of the Solr search API and will be based on similarity matching

The outcomes of the matchmaking will be presented to the user through an intuitive UI, which will be integrated in the EFPF Portal.

3.2.4.3 Step 3: Online Bidding Process

The online bidding process will be enabled through the EFPF Marketplace Framework. In order to support matchmaking of a request for a product/service with the best available offer at the EFPF platform level, the following components are included:

- The common schema/ontology that is implemented for federated search will be used to collect data from base platforms (basic information about companies, products and services etc.)
- An API that will provide post interfaces to enable the collection of data from base platforms and an RDF store that will store the data in a format suitable for offers/requests matching
- Apache NiFi instance to support the connection of API with basic platforms
- Rule-based mechanisms for inferencing knowledge from RDF store and Automated Criteria Weighting and Best Score algorithms for online offers evaluation and matching to requests. Apache Jena will be used to support inference over RDF
- An API capable to get other data related to offers that should be available on real-time by Marketplace Frontend (prices, payment terms, delivery time etc.)

3.2.5 Governance & Trust

Governance services in EFPF support various processes and platform management structure, e.g. supervision of data flows, alert management for tracking problems related to processing of data flows, compliance with the business objectives, applicable laws and regulations, etc. Governance services target policies and practices defining the platform management. For example, governance services define assignment of responsibility (user's roles) and manage configuration policies, in an effort to control various risks in EFPF. By contrast, compliance targets only the specifics of regulations and their requirements.

The governance framework in EFPF is grounded on literature review on several existing IT governance and management models including the following:

- ITIL framework that provides guidance on how IT processes should be planned, designed and implemented
- COBIT governance framework with a focus on business risk management
- ISO/IEC 27001/2 with a focus on implementation of organisation's Information Security Management System (ISMS), and
- Cloud computing (CC) governance model that identifies four key governance domains in the cloud: Cloud Migration (CM), Information Security (IS), Risk Management (RM) and Service Level Agreement (SLA) [BMH18]

The Governance Framework (GF) in EFPF is holistic by its nature, incorporating platform organisational standards, strategic planning, business rules and norms of behaviour in the ecosystem, software standards, regulatory requirements, etc. which all need to be continuously monitored and assessed. Figure 22 illustrates the preliminary, high level architecture of the EFPF GF. Apart from continuous platform monitoring and assessment, the framework includes Roles and Responsibilities, Governance Registry and Governance

Decisions and Feedback services. Some examples of roles and responsibilities are: Governance Manager (defines the roles and responsibilities; defines business process lifecycle; monitor services, etc.), Business Process Designer (defines business process choreography policies and choreography level agreements), Service Provider (register services, discover services, manage SLA, monitor services and business process execution, etc.), etc.

Depending on user's roles in the platform ecosystem, users set priorities and focus (metrics too), taking into account various elements from the Governance Registry in order to create the best governance decisions and feedback. In addition, the Governance Manager sets various policies in the system, including rating and ranking policies which are used for calculating trust and reputation of the EFPF stakeholders.

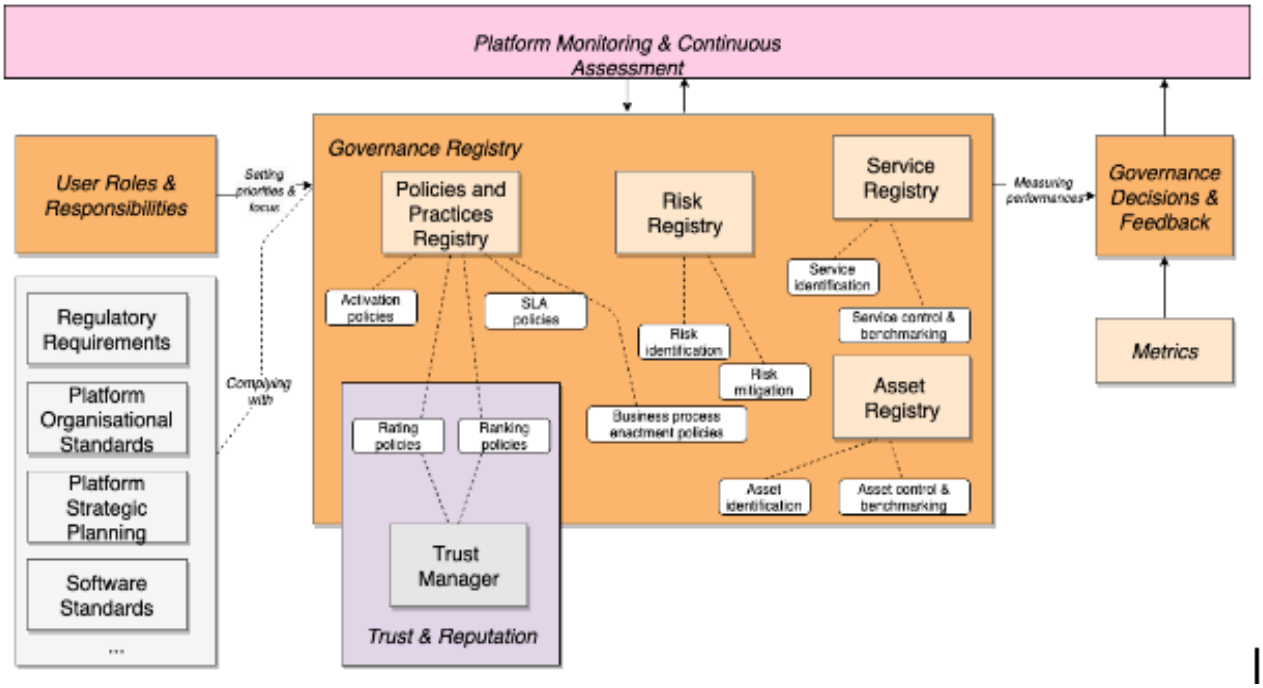

Figure 22: Holistic Governance Framework in EFPF

Furthermore, Figure 23 illustrates the access management in the EFPF ecosystem, which is done using the User Managed Access (UMA) protocol. UMA 2.0 is a federated authorisation framework defined on top of OAuth 2.0. It defines how resource owners can control access to a protected resource by clients, when access to the resource is based on resource owner policies.

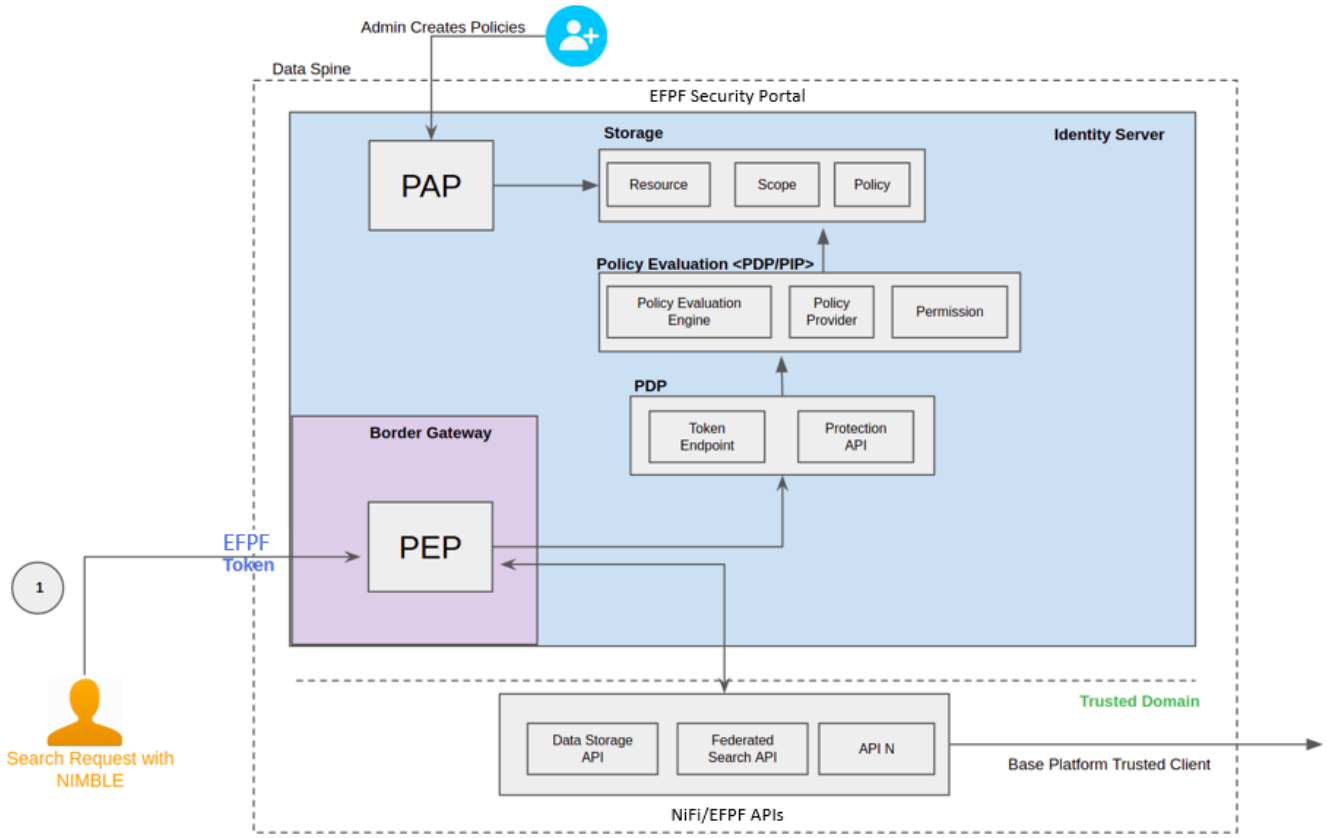

Figure 23: UMA-based Access Management Supporting EFPF Governance Mechanisms

The following bullets list the components that govern the user-managed access for resources/ HTTP endpoints.

- **Policy Administration Point (PAP)**: PAP provides a set of Administration Console to manage resource servers, resources, scopes, permissions, and policies. Part of this is also accomplished remotely through the use of the Protection API. The EFPF security admin will create the policies via the PAP
- **Policy Decision Point (PDP)**: PDP provides a distributable policy decision point to where authorisation requests are sent and policies are evaluated accordingly with the permissions being requested. The output will be to either allow or deny requests, based on the users permissions/roles
- **Policy Enforcement Point (PEP)**: PEP provides implementations for different environments to enforce authorisation decisions at the resource server side. The Identity Server provides some built-in Policy Enforcers
- **Policy Information Point (PIP)**: PIP helps the identity server to obtain attributes from identities and runtime environment during the evaluation of authorisation policies. For example, if the Identity Server needs attributes from the base platform to make a decision then the PIP can perform this action

3.2.6 Business & Network Intelligence

Modern manufacturing processes are becoming increasingly data driven. This has led to manufacturers demanding more digitalisation and data sharing from their supply chain. Data

sharing between digitally connected enterprises provide analysis opportunities that help in understanding key drivers behind performance, planning and other important metrics. Such analysis generates intelligence which is of high importance in Industry 4.0.

In EFPF, this intelligence is broken down into two categories:

- **Business Intelligence (BI)**: BI can be described as actionable information that enables better business decisions. BI provides historical, current and predictive views of business operations of a company which informs future strategic decisions. Streams of data from all business areas are integrated together and reported on in a single interface. This method surfaces inconsistencies, overlaps and other areas of opportunities for businesses to improve upon and grow. Moreover, businesses tend to share some of these favourable BI metrics (especially about quality control, rejects, material wastage, carbon footprint and audit performance etc.) with their prospects and customers as a badge of quality work to win more businesses

- **Network Intelligence (NI)**: NI can be understood as information that presents a higher-level understanding of the position, structure and evolving interactions of a company in a consortium, supply chain, industry association or a network of similar or complimentary businesses. This is derived by analysing trends and interactions between users (businesses in the case of EFPF) of the platform. This can be achieved by tracking the B2B interactions taking place through the Data Spine. Some of the outcomes of NI could be providing cluster segmentation by capabilities or risk. Another one could be suggestion of ideal partners, customers or suppliers to companies based on their interactions in the network

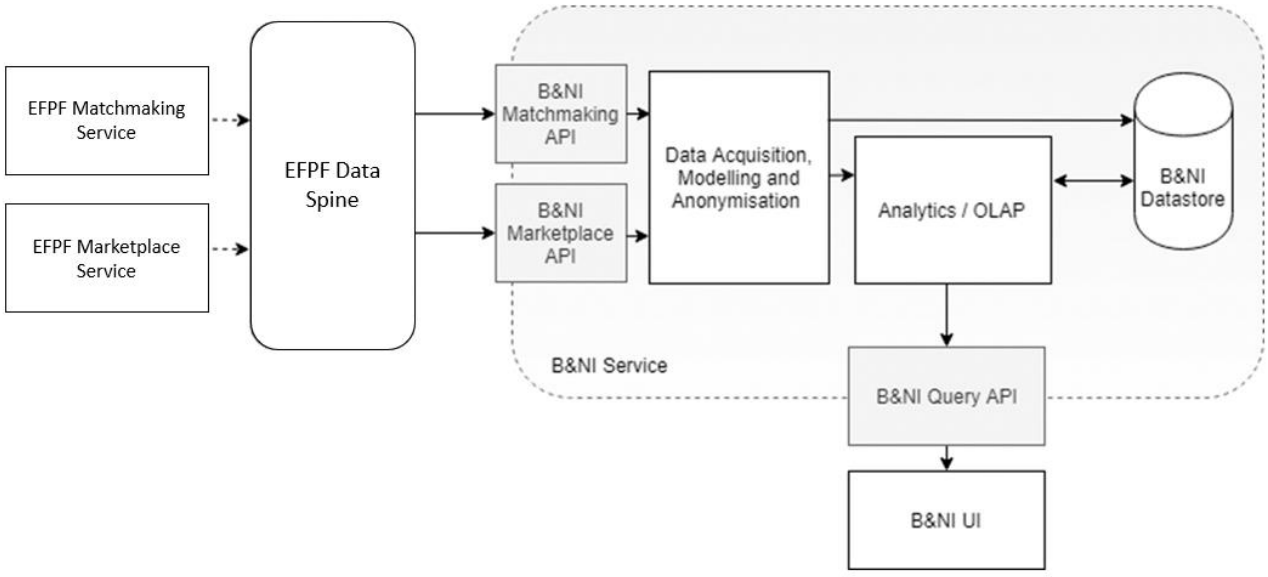

Figure 24: B&NI Service Architecture

It is essential for EFPF users to be able to generate, share and monitor intelligence throughout the platform to make actual business possible and beneficial to all. The Business and Network Intelligence (B&NI) service in EFPF is designed to extract transactional information from multiple tools/service in the EFPF platform. This includes information from

the B2B transactions (e.g. which companies are actively seeking collaborations; which capabilities are being sought, etc.) as well as B2C transactions (e.g. which products are being discussed, etc.). These types of information can be captured for example, from the Matchmaking (Section 3.2.4) and Marketplace (Section 3.2.3) services in the EFPF platform. In this respect, the architecture of the B&NI service (shown in Figure 25) describes the interaction with internal and external components in the EFPF platform. Based on this architecture, the B&NI service connects to the Data Spine to collect the transactions (queries and responses) carried out by the EFPF Matchmaking and Marketplace services. A link with other EFPF services can also be established in the similar fashion, depending on the relevance of their data.

The information captured by the B&NI Service (through dedicated APIs) is processed and stored in a datastore, allowing specific analytic queries to be performed extract business and network intelligence. An intuitive UI of the B&NI service allows users to tune their queries according to their needs and the results of the user queries are also presented through the UI.

In addition, the integration of 3rd party tools in the EFPF federation will allow the EFPF users to take advantage of advance B&NI analytic capabilities offered by these tools. For example, Valuechain's iQluster platform is a product of Horizon 2020 funded project – Data Integrated Supply Chain Optimisation. iQluster is a supply chain intelligence platform that enables businesses to record and promote their business intelligence. iQluster further enables, industry associations and / or supply chain networks to form cluster networks and generate as well as share some network intelligence.

iQluster has been integrated as a 3rd party add-on to EFPF platform. iQluster is currently available through the EFPF Portal. As a first step in proving the platform contribution towards the project, a digital network of members from Hanse Aerospace (EFPF partner association) has been created. Figure 25 shows the current dashboard of the network in the iQluster platform. It enables other platform users to explore network level intelligence on 'material capabilities', 'certifications' and geographical location of member companies.

This network intelligence is aimed to provide high-level guidance to EFPF users before engaging with other companies on the platform.

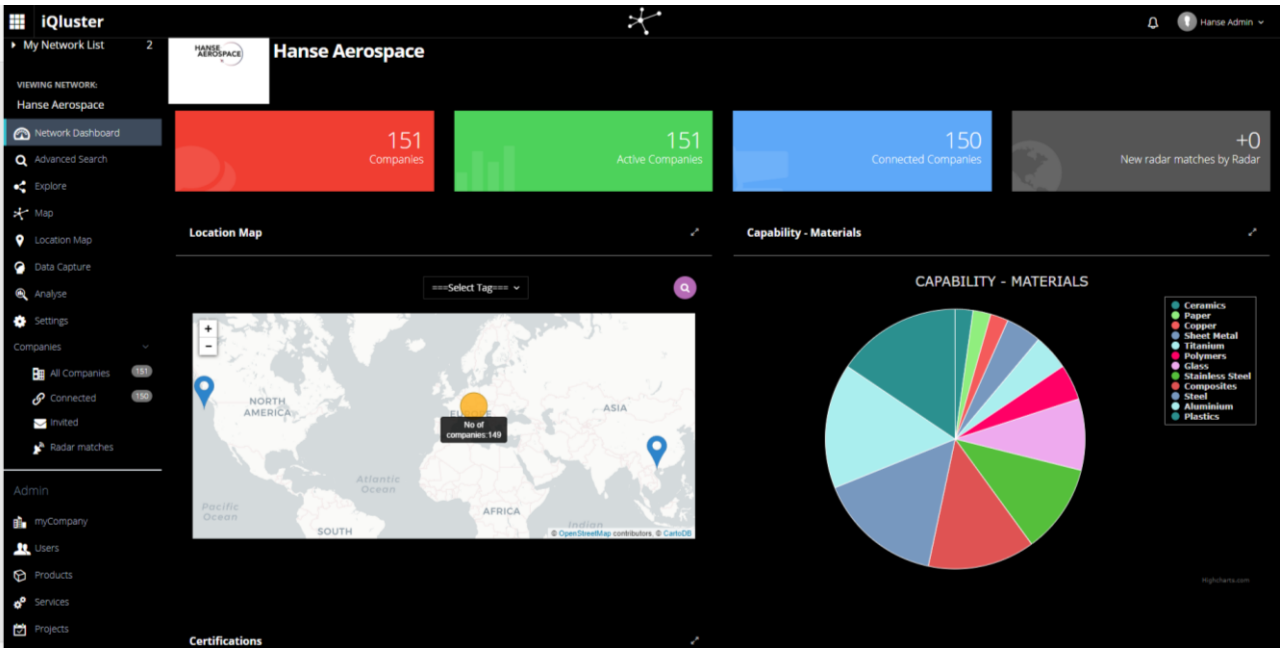

Figure 25: Hanse Aerospace Network Intelligence Dashboard

The intelligence generated through the iQluster platform will help EFPF users to explore the network, make a decision on collaborating and develop trust backed by data (or derived intelligence).The iQluster platform also allows EFPF users to intelligently navigate the platform and reach, connect and collaborate with other businesses of interest. Figure 26, shows a mind map visualisation of the Hanse Aerospace cluster, depicting different types of connections in the network.

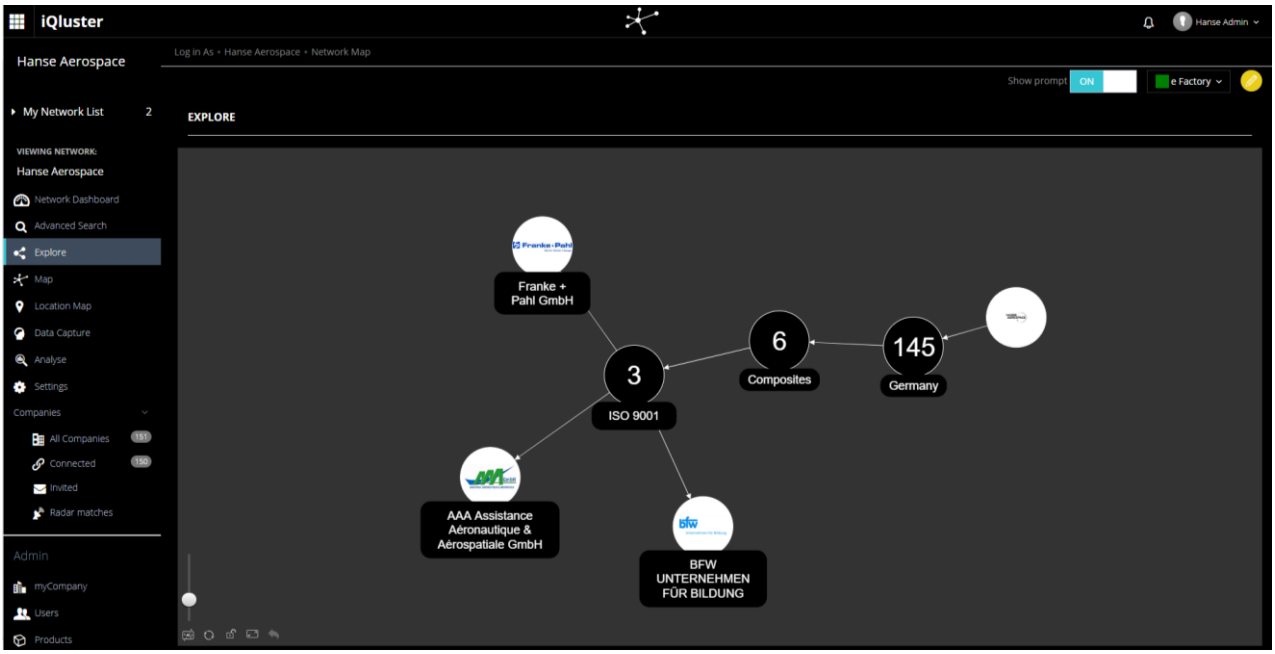

Figure 26: Network Intelligence Visualisation in the iQluster Platform

3.2.7 Smart Contracting

The Blockchain and Smart Contracting component provides a trusted system for smart contracting agile networks in the EFPF ecosystem. This component is based on the distributed ledger technology systems - of varying TRL – provided by the base platforms.

Distributed ledger technology is – according to our experience and by design - a suitable design mechanism [KRU04] for persistence when the system is situated in an environment where three main conditions exist: 1) a distributed, immutable log of transactions is needed, 2) there is a need for distributed trust and 3) there is an incentive to manipulate data. A set of generic user stories on which to base user stakeholder concerns following the ISO/IEC/IEEE 42010 standard, have been developed:

- Blockchain-based Tracking of Shipments
- Blockchain-based Certificate of Origin (CoO)
- Circular economy scenarios
- Permission Management
- Audit trail of supply-chain data
- Product condition and maintenance tracking

From the above listed generic user stories, a first set of generic functionalities targeting the needs of the Industry 4.0 domain have been derived. The first of the three functional categories handle the management of identities, access rights and consent to view data. The second category deals with the need to store business documents and production data in a shared distributed ledger. The third set is about handling user-specified transaction rules for the distributed ledger and smart contracts acting on data in the ledger and from external sources, e.g. temperature sensor data for tracking a cold chain delivery.

The above functional categories address the following subsets of capabilities:

- Identity and access management
  - Authenticate users and IoT devices
  - Authorise access to data, consent management
  - Confidential data

- Blockchain data store with verification of authenticity, origin and standards of data and services
  - Generic function for storing data items / documents
    - Supply chain data
    - Product data
    - Business transactions

  - Attach additional corroborating evidence to document
  - Allow several parties to sign a data item/document
  - Certificate of Origin (CoO)

- Smart contracting agile networks
  - Use data available in the blockchain, from sensors, CoO and business documents
  - Select from a pre-defined set of contracts
  - Trusted external data ("Oracles")

The base platform implementations that will be matured into the EFPF Blockchain and Smart Contracting system are the Blockchain as a Service (BaaS) and the COMPOSITION Blockchain.

In the COMPOSITION ecosystem, each marketplace has one blockchain, which is the store of marketplace stakeholder agent public keys, an audit trail of agent CXL (COMPOSITION eXchange Language) messages, and agent reputation data. A blockchain node may be deployed by any marketplace stakeholder. The COMPOSITION Blockchain API controls the private keys and addresses, and is accessible only to the stakeholder's agents. The blockchain nodes synchronise the state of the blockchain in a peer-to-peer network.

Regarding the Blockchain as a Service BaaS system it communicates through endpoints with the blockchain nodes which form the private permissioned network. The BaaS exposes an endpoint where it accepts requests and transactions from the EFPF platform which serves as the connector between the Data Spine and the blockchain storage and services (see Figure 27).

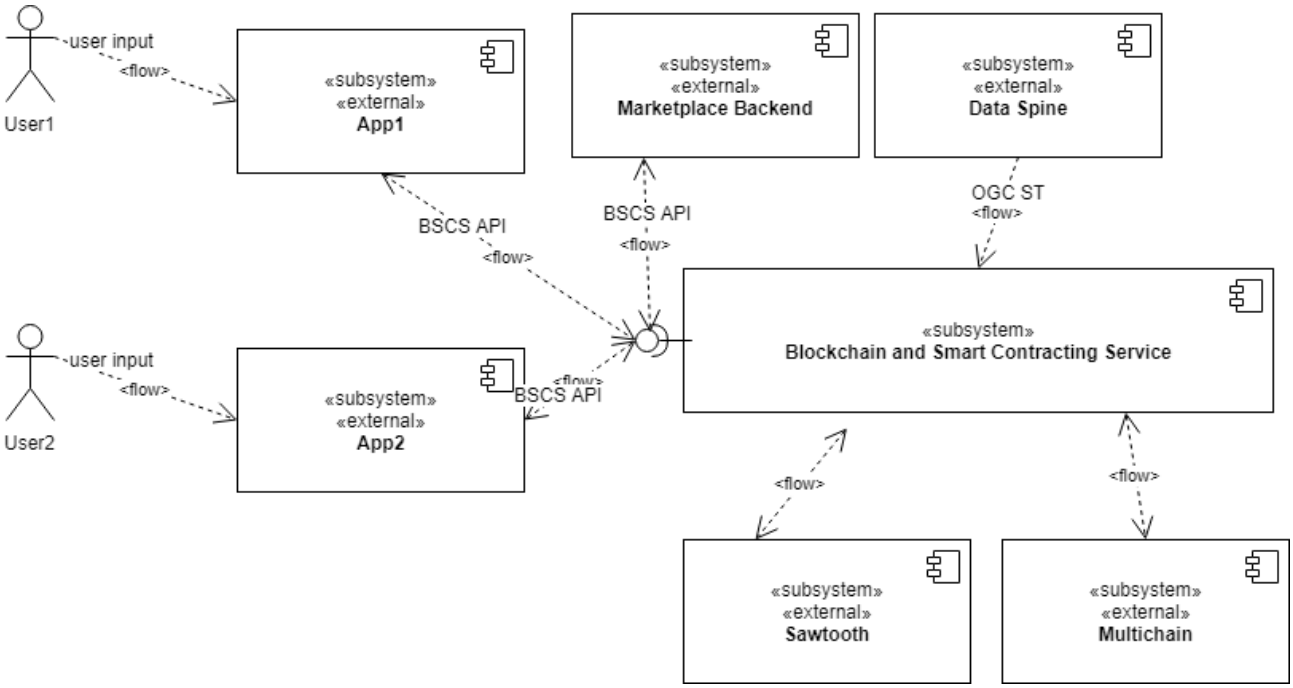

Figure 27: Context Diagram of the Blockchain and Smart Contracting System

3.2.7.1 Functional View

A conceptual functional view of the system is shown below. The Blockchain and Smart Contacting system has three main layers (see Figure 28):

- The Blockchain Implementation Layer, with the adapters for the use and management of different blockchain implementations,
- The Blockchain Services, with the functional packages for the general categories of user stories with supporting functionality such as connectors to the Data Spine; and
- The App Layer, comprised of specific applications built on top of the Blockchain Services. These applications can be designed at the level of integration with ERP system integrations, as standalone web applications, or mobile apps.

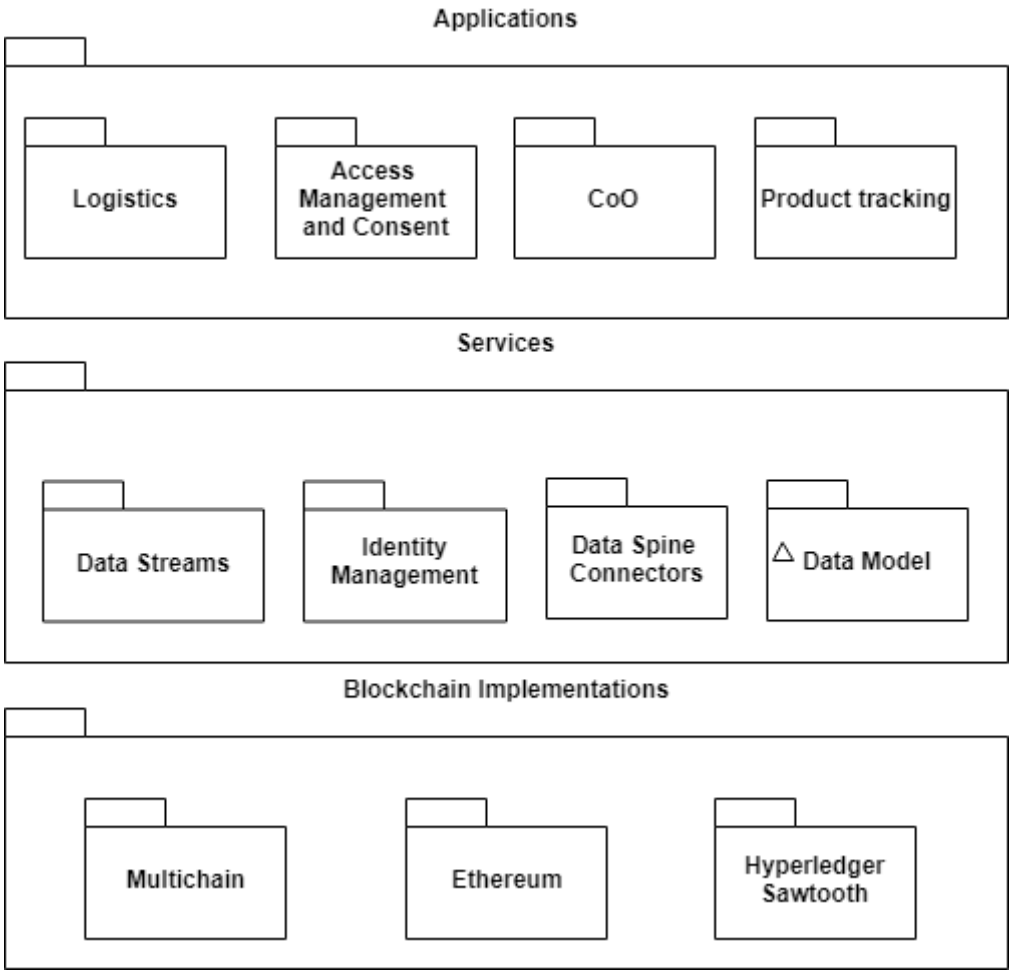

Figure 28: Functional Packages of the Blockchain System

3.2.7.2 Information View

A simple model to store and exchange data in a way that is agnostic to the blockchain implementation has been followed

The data stream concept is similar to the one found in Multichain[11] and OGC SensorThings[12]. It is an ordered set of data items in JSON format. Each item is "tagged" with a set of keys that can be used to filter the results. Each application using a blockchain as a key-value or time series data store uses the same model to represent the data it stores (see Figure 29). A data stream can represent a delivery process in which each item is an event, or a data stream can represent a product and each item of the product is a log entry of production steps, shipping, and post sales maintenance and repairs. When adding external data, additional information such as sensor values, images, or results of consensus algorithms can be supplied to corroborate the correctness of this data.

[11] https://www.multichain.com/developers/data-streams/
[12] https://www.opengeospatial.org/standards/sensorthings

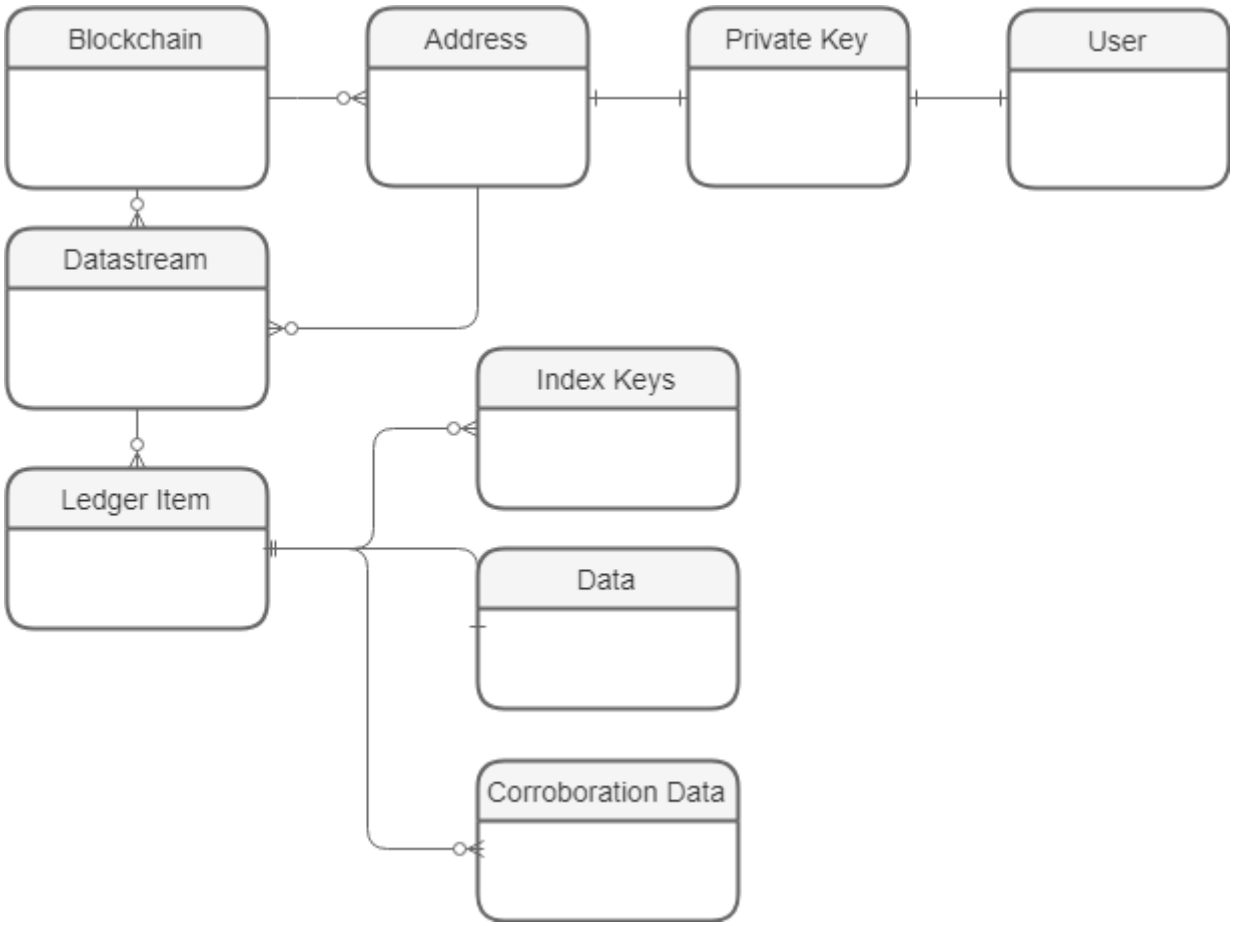

Figure 29: Model for the Data Store in EFPF

3.2.7.3 Security Perspective

The security perspective deals with issues such as the choice of and principles for the consensus process (private, consortium, public), support for permissioned and permission-less blockchains, and decision on where to generate and store private keys used to create addresses and sign transactions. Addresses serve as secure identifier to send and receive transactions (like an IBAN in banking transactions). These private keys are generally created and stored in the users blockchain "wallet", either at the node or at a third party linked to the other user identity (e.g. in the case of "web wallets").

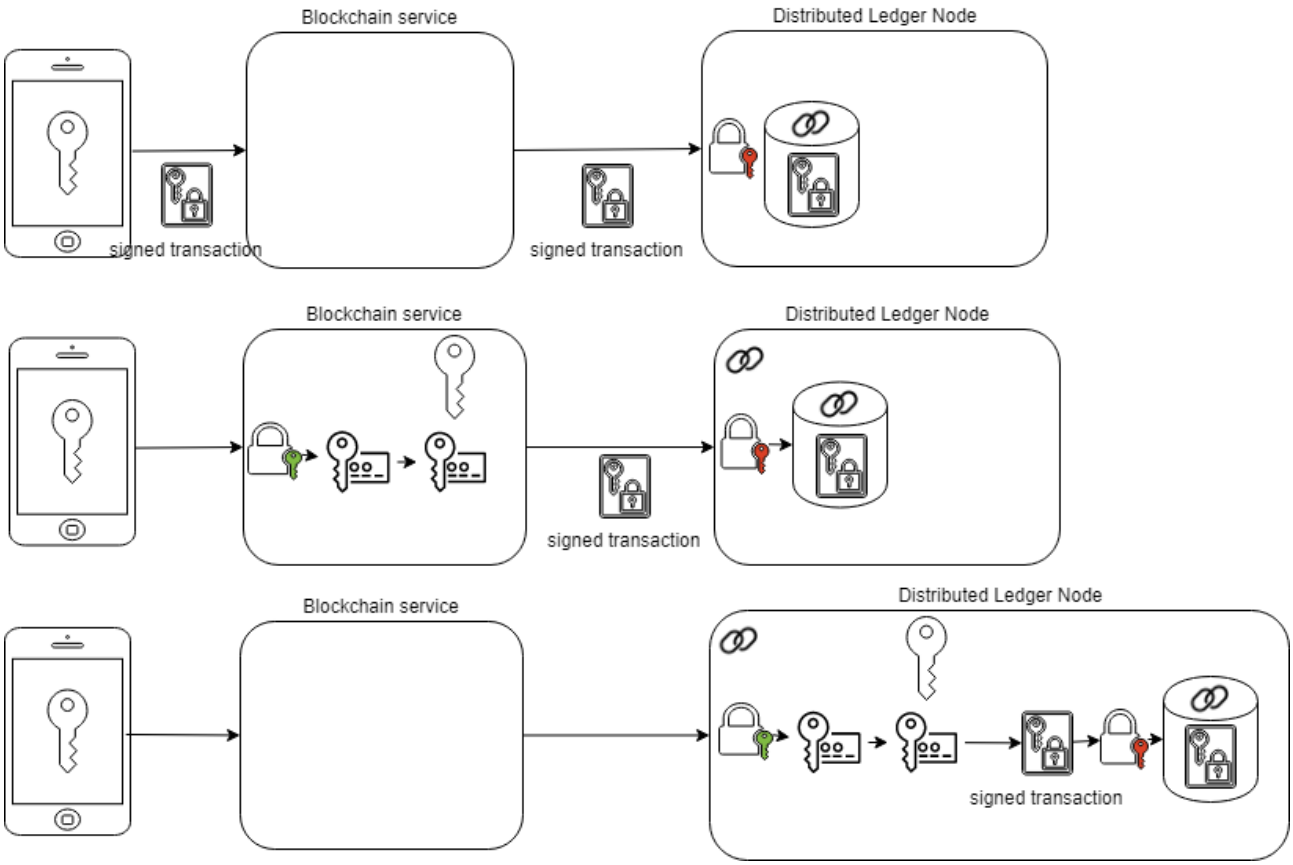

Figure 30: Private Key and Address Management in Blockchain Applications

Figure 30 depicts three solutions for private key management:

- The private key is stored at the user node and no other application user identity is needed to interact with the blockchain. In this case, messages can be signed and addresses constructed at this node;
- The private key is stored at the middleware server. In this case, the application user identity is matched to a blockchain private key (e.g. “web wallet”);
- The private key is stored at the user blockchain node, and the authorisation for the application user identity is performed there.

In the first and third option, the users in the system are in control of their keys and addresses and no centralised trust is required. The data in a blockchain can be trusted because transactions conform to the defined transaction rules, act on data in the blockchain and the consensus algorithm ensures that everyone agree on the correct state of the blockchain. When data is entered from external sources, however, this data cannot necessarily be trusted (e.g. sports results, financial data or sensor measurements). The security perspective will also document the exploration of designs in order to enable the distributed ledger and the smart contracts to trust data entered into the system, e.g. from the Data Spine. This is commonly known as the "oracle problem".

3.2.7.4 Blockchain Applications in EFPF

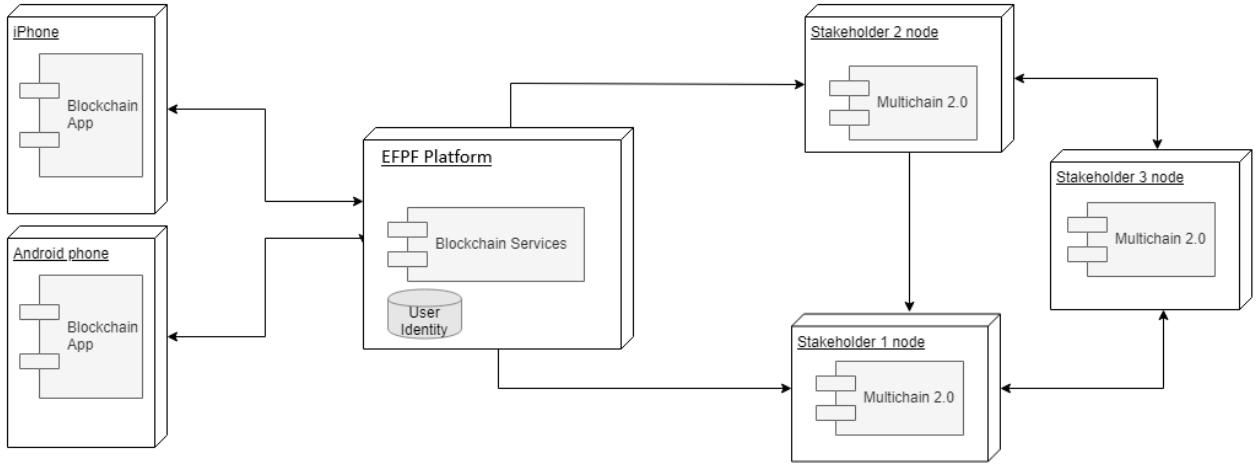

Figure 31: Supply Chain Mobile App

Currently two applications are built to serve as proof-of-concept and input to the architecture design.

- **Supply Chain Mobile App:** The key feature of the mobile application demonstrator (Figure 31) is to use physical interaction to address weak points or loopholes where supply chain can become vulnerable. Added sensor data and biometric identification are used to corroborate the transaction data (e.g. a receipt of received goods). As the data is a representation of a transaction taking place in the physical world and entered into the blockchain, it cannot be directly validated by the blockchain transaction rules or smart contracts. To circumvent this reliability problem of physical interfaces, other metadata is added such as Near Field Communication (NFC) tags, weight, images and sign in with face or fingerprint ID to provide other evidence that the event took place
- **Identity Management (IDM) based on Blockchain in EFPF:** Blockchain technology can be used to facilitate the user's authentication and authorisation, enabling decentralised, tamper resistant, inclusive, cost effective and user-controlled identity services [DP18]. Using Blockchain as a Service BaaS, the intention is to remove the complexity involved in setting up a blockchain network for each platform and to unify the process of user registration and authorisation through a cryptographically protected Distributed Ledger Technology (DLT) [SD16]. IoT devices are also subjects to blockchain IDM Public keys can be used to issue digitally signed transactions through the EFPF 's agnostic layer, after a user has given her/his consent for such actions or in cases where permissions are previously given and control over automatic actions is required to ensure the system's integrity and security. Thus, identities can be issued to smart IoT devices, which can then be authenticated and interact with the blockchain smart contract services

The data is also expected to be confidential and sensitive. Here, the system's blockchain private ledger is an ideal solution for this kind of storage. All the data on the permissioned blockchain can be encrypted and only users with privileges and/or permissions can decrypt and access this data. The EFPF blockchain technology is grounded on the Hyperledger Sawtooth (see Figure 32), which provide the following features:

- Parallel Transaction Execution
- Event System
- Dynamic Consensus Algorithms
- Private Networks with the Sawtooth Permissioning Features.

Special notice should be given to the feature regarding the Permission policies which can be applied on Sawtooth blockchain BC networks, which suits well the IDM needs.

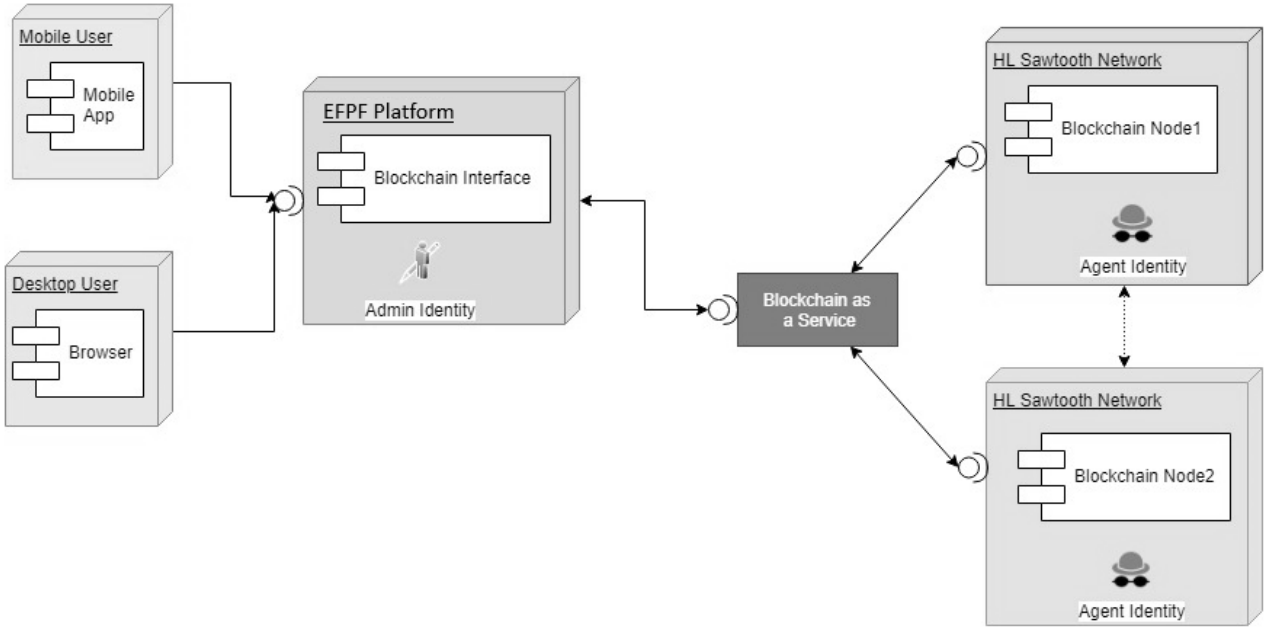

Figure 32: Blockchain as a Service

3.2.8 Data Analytics

For any manufacturing company these days, it is critical to know and take advantage of the data and information concerning machines, systems, processes and also customers. The capture and processing of information allows manufacturing companies to plan and execute manufacturing activities more efficiently, to effectively use available resources, streamline their activities and processes, target relevant customers, etc.

In EFPF, the data analytics solutions are provided through a Data Analytic Dashboard, which is accessible as an integral part of the EFPF Portal (as shown in Figure 33).

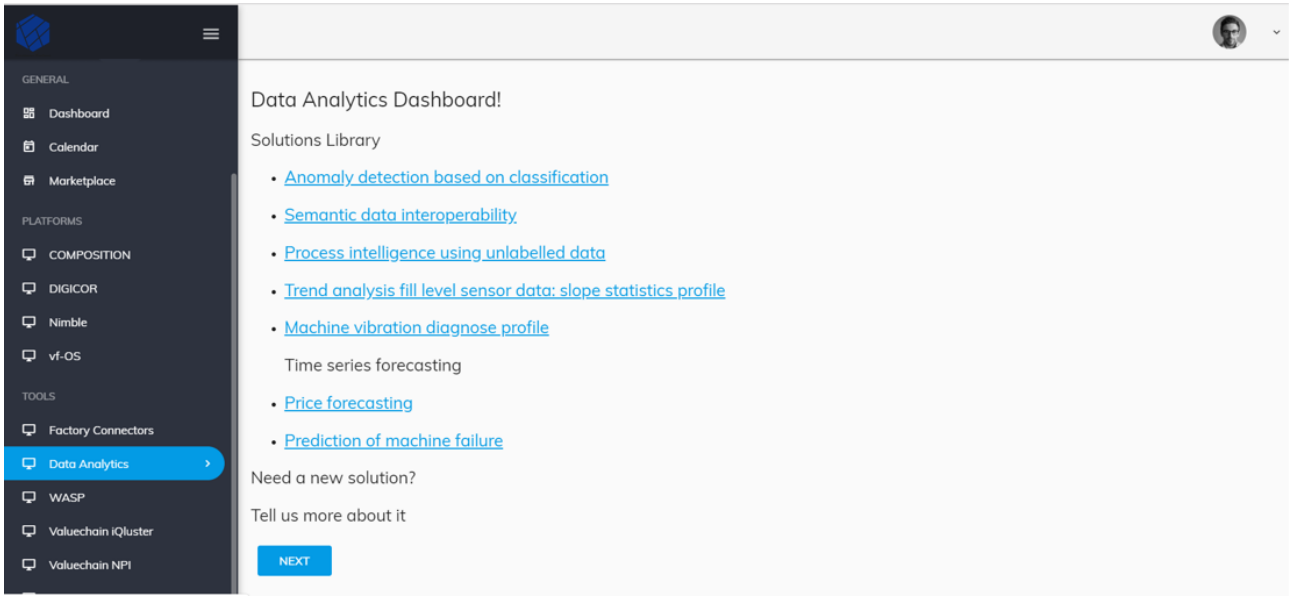

Figure 33: Snapshot of the Data Analytic Dashboard in the EFPF Portal

The Data Analytic Dashboard in EFPF is populated by a set of data analytic tools/applications provided by the EFPF partners. These include the applications that deal with factory data (sensor, machine, process, etc.) and applications that process and deal with stakeholder's data (customers, collaborators, etc.).

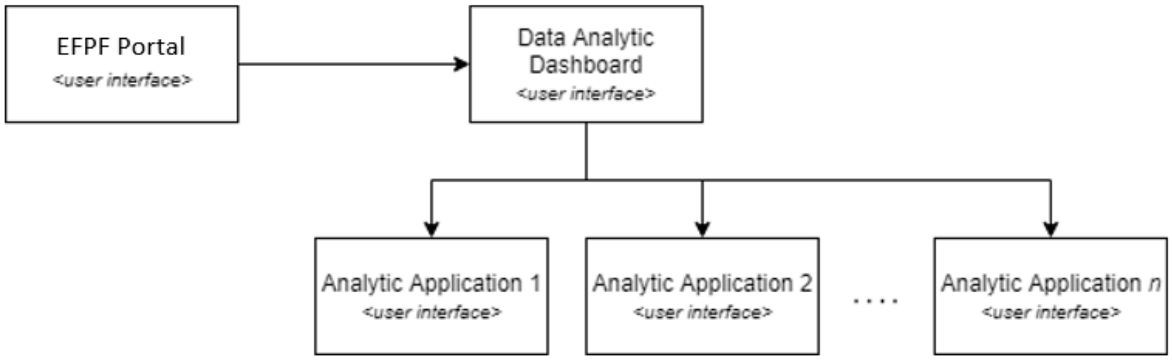

Figure 34: Structure of the Data Analytic Dashboard

The Data Analytic Dashboard is independent of or not concerned with the functionalities of the actual analytic applications. It merely provides a unified interface to the EFPF users, allowing them to make use of different analytic applications from the same place and (as far as possible) in a similar way. Thus, any number and nature of applications can be made accessible through the Data Analytic Dashboard.

The actual functionalities or internal mechanics of the analytic applications e.g. how the data is accessed, how data is processed and stored, which algorithm is implemented, where the application is hosted, how the results are presented to the user, etc. are left on the application developer (EFPF partners) to decide. In this respect, the dashboard can be seen as a container that provides unified access to multiple analytic applications provided by the EFPF partners.

The guiding principles for exposing analytic applications through the Data Analytic Dashboard are the following:

- Applications needs to be offered through a user interface
- Applications need to be offered in plug-and-play type approach where minimal customisations or configurations are expected from the users
- Applications need to be generic in the sense they serve the needs of wider user-base i.e. applications should not be designed to serve specific needs of specific users
- Applications need to be offered in a simplified form keeping in mind the target users i.e. manufacturing companies

The following data analytic applications are currently being made available through the dashboard. The development, deployment and maintenance of these applications remain responsibility of the respective partners in the EFPF project. These applications are primarily selected based on their relevance to the users' needs and business/activities.

- **Anomaly Detection:** This analytic application uses ML techniques to detect anomalies in the operating behaviour of machines
- **Trend Analysis:** This analytic application uses condition monitoring techniques to identify trends in shop-floor activities
- **Predictions of Machine Behaviour:** This analytic application uses unsupervised ML techniques to predict machine behaviours
- **Customer Behaviour Prediction:** This analytics application uses ML techniques to analyse and predict customer behaviour

3.2.9 Workflow & Business Process

The workflow and business process solution in EFPF platform is called the Workflow and Service Automation Platform (WASP). WASP allows users to model, automate and/or orchestrate processes of different types involving manual activities carried out by people, software-based processes, such as execution of (web) services, and manufacturing processes where manufacturing activities and machines are represented through software interface. WASP supports the automation of processes that are carried out within an organisation as well as processes that spread across multiple organisation.

WASP offers a complete workflow and business process automation solution where users can register to create an account and design workflows using an intuitive dashboard (see Figure 35). Users can be the owners of their workflows that can be shared with other users within the organisation. In extended functionality, users will be able to share their process with other collaborators outside their organisation.

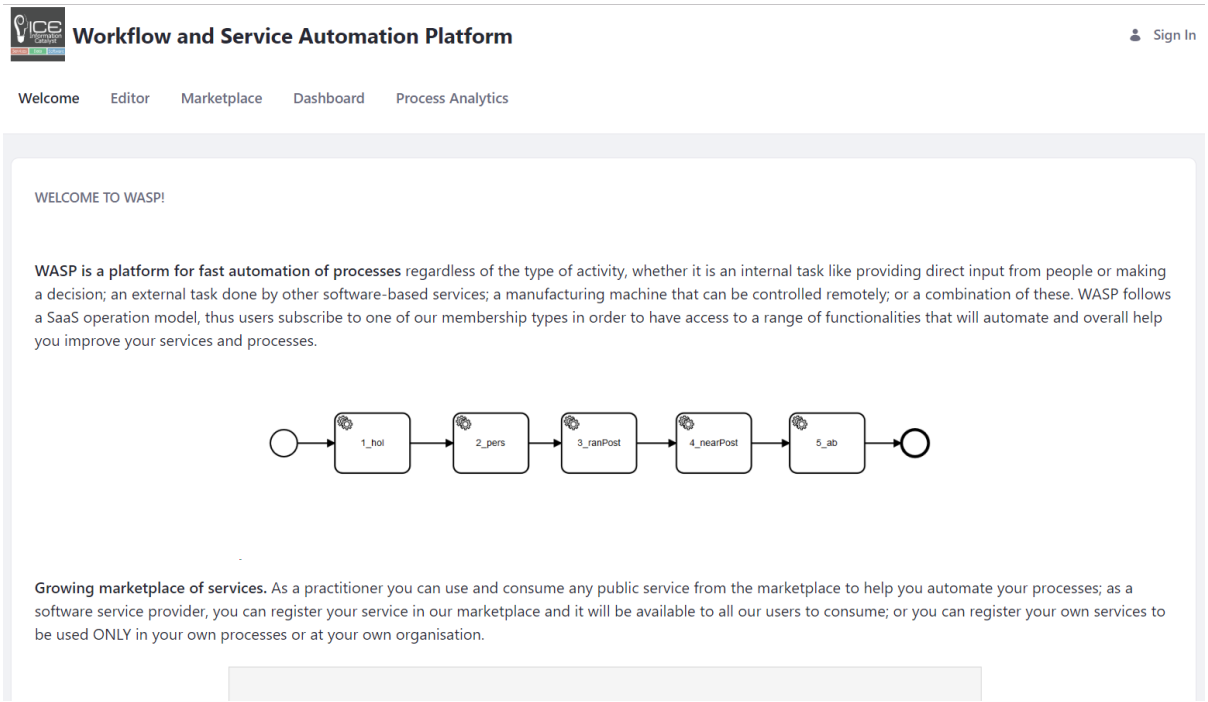

Figure 35: Snapshot of WASP Interface

Through its internal user rights and access management system, WASP helps users specify governance of workflows or parts of them (when working on the systems based on WASP). Monitoring of executed workflows is also included at different levels of granularity (e.g. activity, process, company, group).

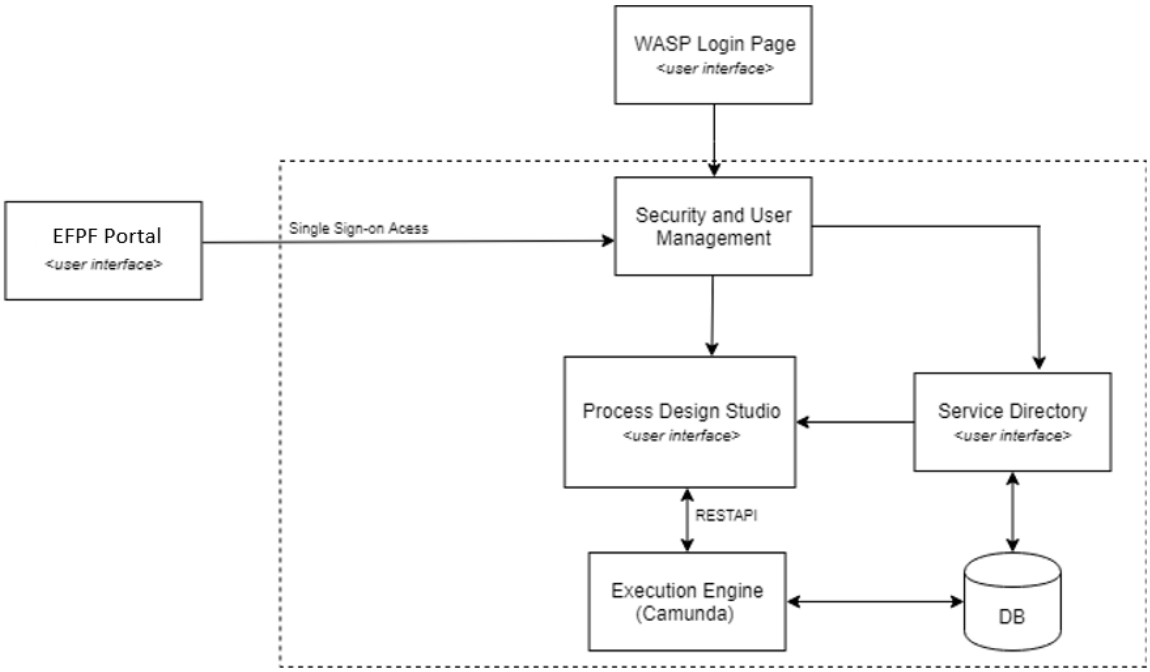

Figure 36: WASP Architecture

The WASP platform is composed of the following modules:

- **Process Design Studio:** The WASP studio provides an intuitive user interface for the design of workflows. Users can drag various Business Process Model and Notation (BPMN) operators (e.g. activities, conditions, etc.) from the floating toolbox, and connect them in a desired order. The toolbox elements can be dragged on the studio canvas and connected with each other using BPMN operators, connectors, joins and merges. Once designed, the workflows can be saved as standard BPMN based processes that can be executed in the integrated execution engine.
- **Execution Engine:** WASP uses open source Camunda engine to execute BPMN based processes. Camunda effectively serves as the core execution engine in WASP. The communication with Camunda occurs via its REST API.
- **Service Directory:** The integrated service directory in the WASP platform provides users the ability to create or expose web-services and link them with different activities in the workflows. Users can set the visibility level of their services to make them useable personally, at the organisation level or at the public level.
- **Security and User Management:** The security of the WASP platform is ensured by implementing industry standard protocols for user authentication and authorisation. The user management system is configured to support SSO within the EFPF ecosystem.

Technical Foundation: The WASP platform web-based BPMN process design, execution and monitoring environment takes advantage of cutting web technologies such as Liferay 7.1, JavaScript ES6, Angular 6.0 and HTML5. The foundations of the Process Design Studio make use of an open source BPMN modeller framework called BPMN.io, which is a rendering toolkit and web modeller for BPMN. It allows easy creation of BPMN2.0 diagrams using a web-based modelling component, which is extended to add the functionality for WASP. The service directory contains assets (Services) which are available within the Process Design Studio for drag and drop into the workflows.

3.2.10 Smart Factory Tools and Services

The EFPF ecosystem supports the delivery and co-ordination of Smart Factory Tools and Services from each of the base platforms and those provided by the EFPF partners. The integrated smart factory tools and services address diverse needs of connected factories and lot-size-one manufacturing scenarios and enables EFPF users to dynamically react to market opportunities and events in the production environments to maximise the business interests.

The initial development and integration efforts have focused on a subset of Tools and Services, e.g. Industreweb Display, Symphony Event Reactor, Risk Analysis Service (RAS) and Factory Connector Gateway Management Tool (FCGMT). These tools and services collectively fulfil the Manufacture / Service Provision phase of a generic scenario that deals with the collaborative production of an eBike product.

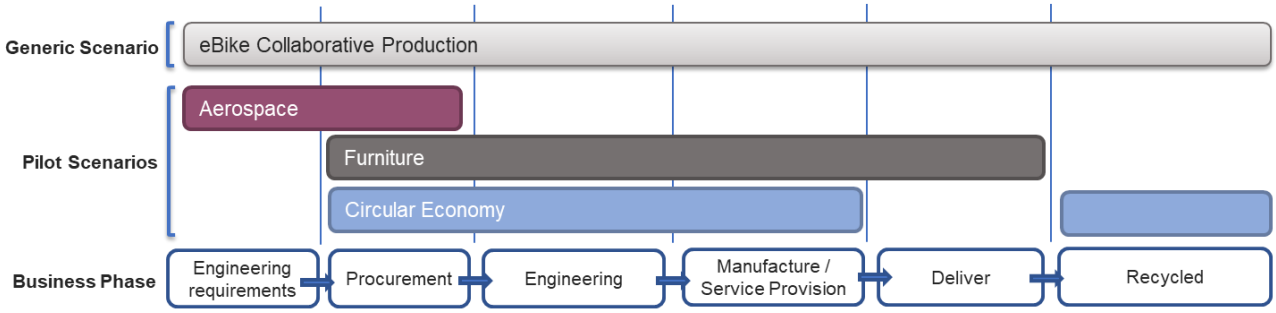

Figure 37: Generic Scenario for the Development of Smart Factory Tools

Figure 37 shows the scope of the generic scenario in relation to the embedded EFPF pilots. The combined usage of these tools and services will allow the end user to monitor production and be notified or risks associated with failing to meet production deadlines.

This section describes the above-mentioned tools and services used in the generic scenario, and how the API for each is applied within the architecture of EFPF Data Spine and platform.

3.2.10.1 Industreweb Display

Industreweb Display is a platform independent, browser-based visualisation tool that is designed to display production information and statistics from the Industreweb Collect server such as:

- Production Faults
- Production Performance
- Waste
- Historical Reports

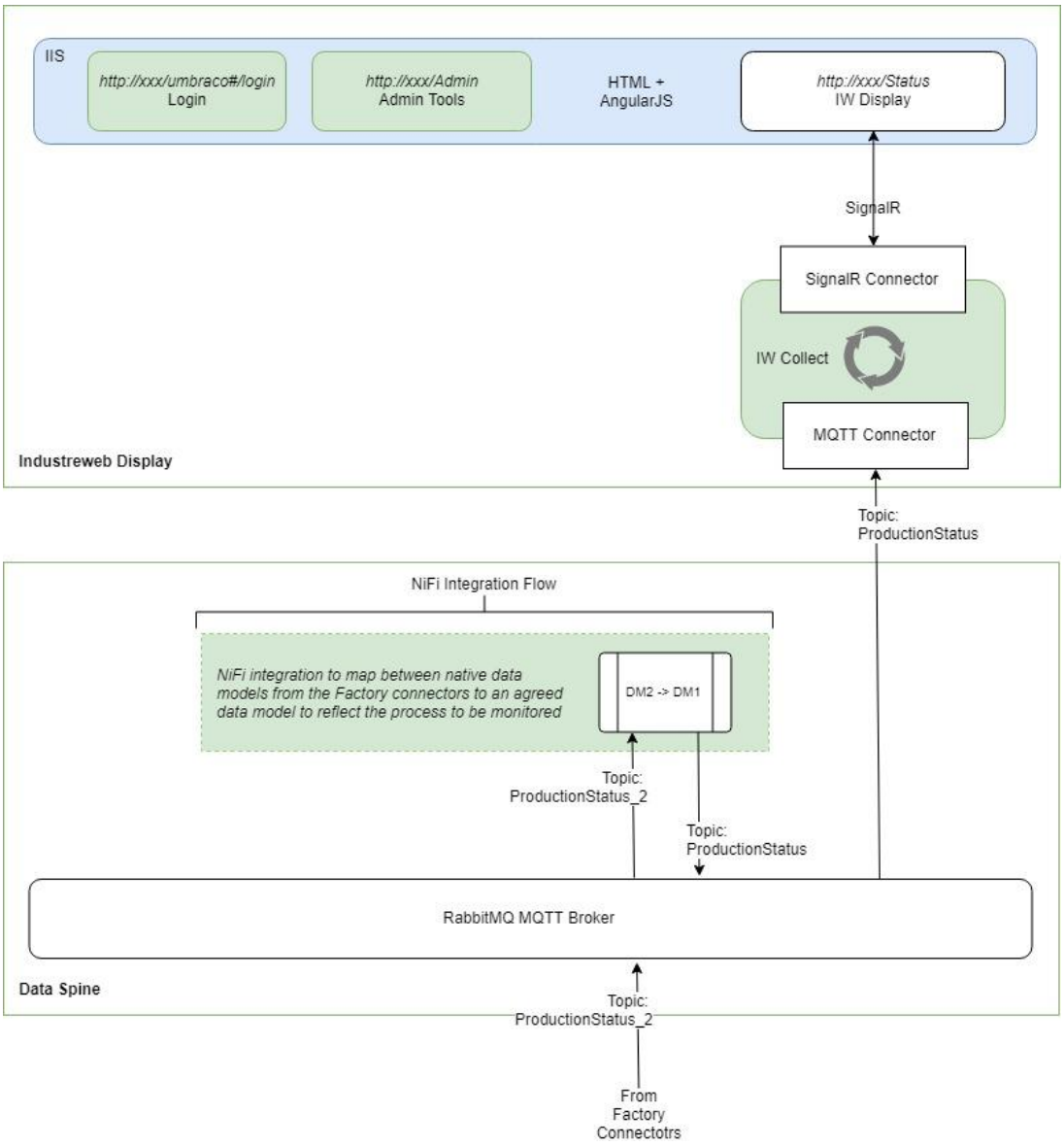

Figure 38: Industreweb Display Architecture

Industreweb Display consumes a Microsoft SignalR[13] stream in order to display live values from the production process. SignalR is a software library for Microsoft ASP.NET that allows server code to send asynchronous notifications to client-side web applications. Pages are constructed from HTML and AngularJS, where the AngularJS subscribes to values in the SignalR payload. The SignalR server is implemented within the Industreweb Collect runtime which is responsible for orchestrating data routing between the values coming from the Data Spine Broker to the SignalR stream. The Industreweb Collect subscribes to the MQTT broker and maps the received data to the SignalR connector (for more details, see Section 3.2.12.1). The architecture for Industreweb Display is shown in Figure 38.

[13] https://docs.microsoft.com/en-us/aspnet/core/signalr/streaming?view=aspnetcore-3.0

Figure 39 shows the wireframe design for the Industreweb Display screen used in the generic scenario to visualise the status of the production process in relation to the production schedule. Text in black refers to the "Collaborative Production Plan" designed by the companies within the collaboration. Text in red refers to real time data updates. Text in green refers to calculation done by service based on the average cycle time and the product quantity target. By comparing the estimated completion and the deadline the service can provide a colour code signal to reflect the status.

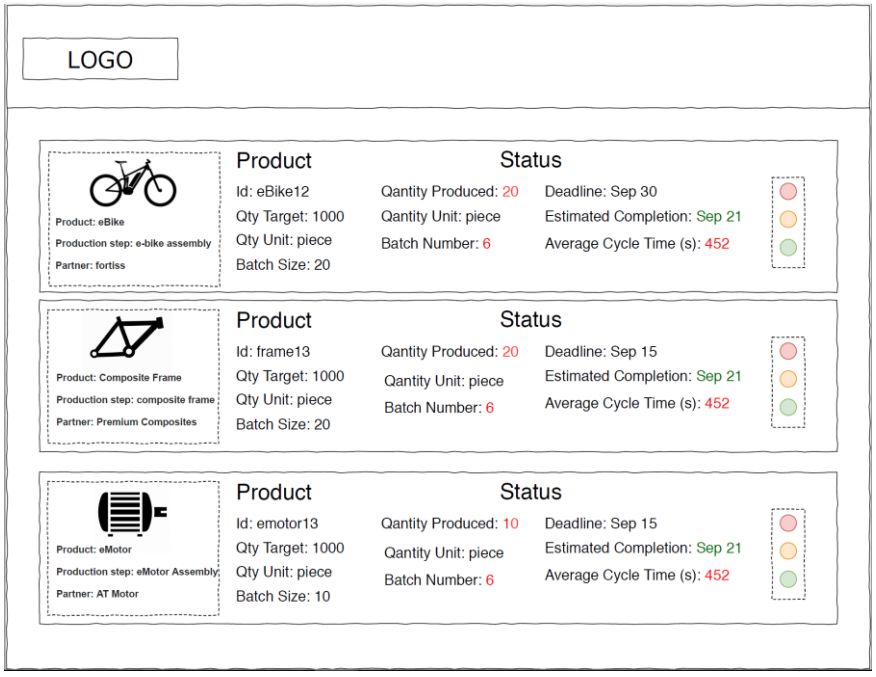

Figure 39: Wireframe of the IW Display Screen to Visualise the Status of Production Progress

3.2.10.2 Symphony Event Reactor

The Symphony Event Reactor gives the ability to trigger actions and alarms through its Event Manager/ Alarm Manager in response to different kinds of event types. Moreover, it offers a logging and lifecycle system for alarms. The data model of events is based on a JSON Schema which is human-and-machine readable and dynamically updatable. The event reactor is written in Python and has a GUI written in Blockly (google), a free and open source client-side JavaScript library or creating block-based visual programming languages (VPLs) and editors.

The Symphony Event Reactor leverages on a highly customisable logging system that allows to handle events locally and synchronise them remotely, together with user activity and alarm history.

The Symphony Event Reactor is composed of two separate software modules:

- **Event Manager (EM)**: The EM executes custom rules that combine information coming from different sources (local sensors and device monitors, user actions, video-cameras, intrusion detection systems, etc.) and data brokers (e.g. AMQP, MQTT) to determine actions to be taken (see Figure 40). Actions include actuations on field

devices, activation of scenarios, generation of events, notifications and alarms, and so on

- **Alarm Manager (AM)**: Alerts can be raised in order to present the situation to specific users or user-groups. The system provides a configurable priority based alert routing system that allows to target a single, a group or mixed sets of users with SMS, emails, pop-ups, etc. (see Figure 41) The AM has an internal state machine to track each alarm's status (Open, Close, Acknowledged, Resolved, Delivered). Also, it logs and keeps alarms history in a log database which is accessible through a REST interface. Different notification channels for the AM are being developed

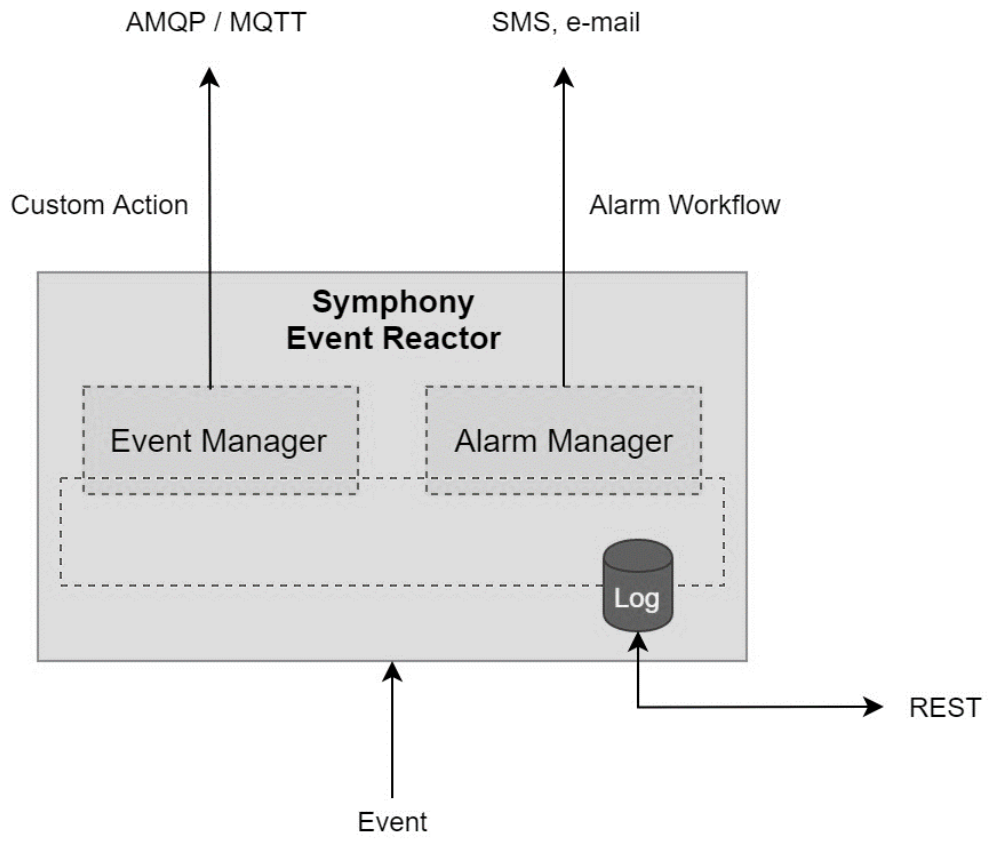

Figure 40: Symphony Event Reactor

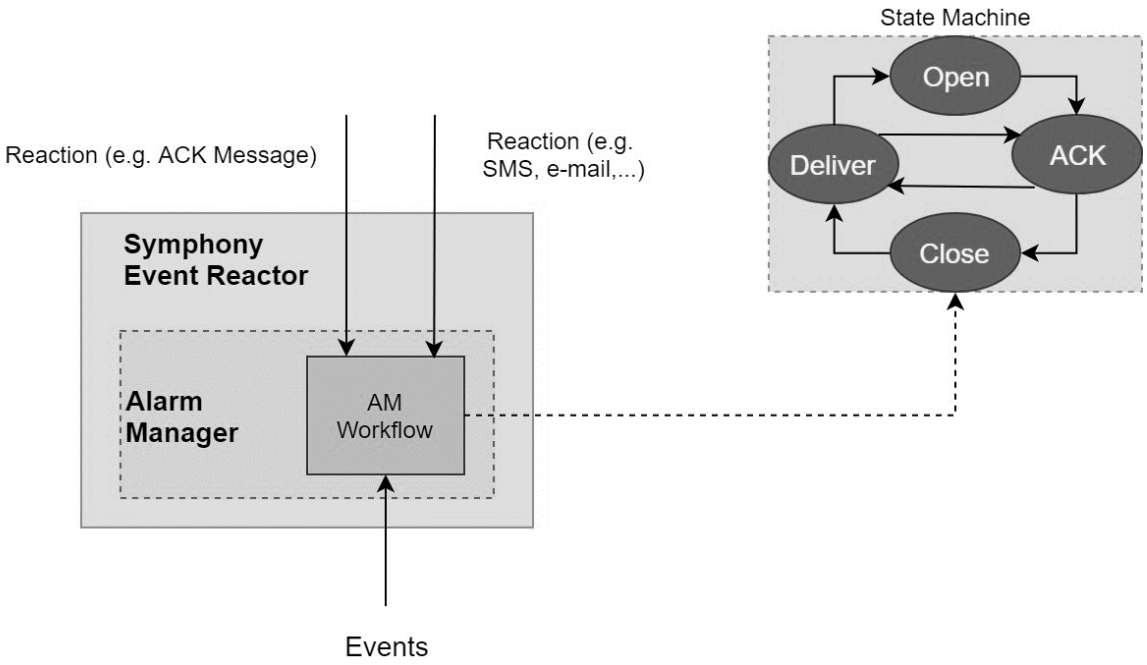

Figure 41: Symphony Event Reactor's Alarm Manager

3.2.10.3 Risk Analysis Service

The Risk Analysis Service (RAS) computes risk scores from data provided by REST API. The current focus is on production risks using data streamed from factories. Data Spine integration flows will be used to send production data to the RAS and to forward the resulting risk scores to other tools within the EFPF ecosystem.

Design: The RAS makes use of risk recipes. These are data analysis modules which take input data and produce risk scores related to the data. Recipes can be executed via REST API, which is further detailed in the following section. Workflows can be configured in the Data Spine to automatically execute recipes as soon as new data become available. In some cases, data specific to the configured workflow may be required in order to compute a risk score. For example, a recipe might require information about the deadline for a process, and this data may not be produced by the relevant data stream. In this case, the deadline would be configured in the RAS ahead of time.

Recipe-specific configurations can be made by POSTing to the following REST API endpoint:

```
/recipe/<id>/configuration
```

The expected schema of the input parameters will be specified by each recipe. At a minimum the configuration must include field which can be used as a key to uniquely identify the data stream to which these configurations apply.

Data flow from and to the Data Spine: The diagram below illustrates the flow of data between the Data Spine and the RAS, as well as within the RAS.

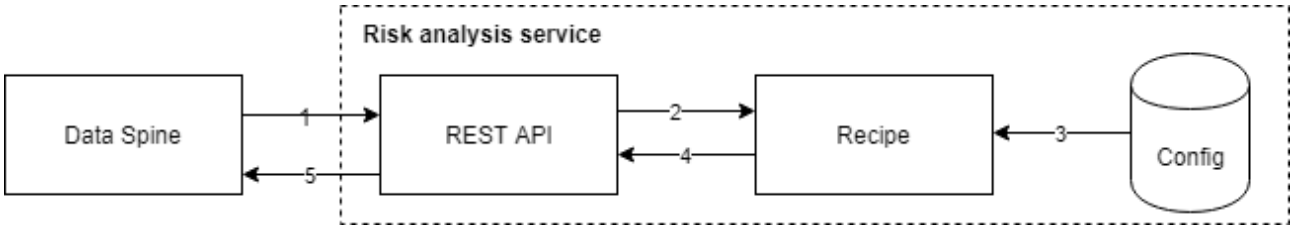

Figure 42: Data Flow of the Risk Analysis Service

The data follows the following path:

1. The Data Spine calls the RAS REST API with GET request to the following endpoint, where the input data is provided in the body of the request as a JSON object:

   `/recipe/<id>/output`

   The expected schema of the JSON object is unique to each recipe
2. The JSON object will be passed to the compute function of the recipe specified using `<id>`
3. The recipe will collect any relevant configuration information. Both the configuration and JSON data will be used to compute the risk score associated with the given data
4. An object containing the risk score and other relevant data will be returned to the REST API
5. The output object will be serialised and returned to the Data Spine

3.2.10.4 Factory Connector Gateway Management Tool

The Factory Connector Gateway Management Tool (FCGMT) is a web component to be integrated in the EFPF Portal for the purpose of managing the different IoT connectors and gateways registered in the EFPF platform.

The most relevant functionalities provided by this tool are:

- Get the list of connectors and gateways registered in the EFPF platform
- Visualise the data schemas or particular data provided by the connectors and gateways
- Subscribe to connectors and gateways to get its data as well as monitor the subscribed data
- Register of new connectors and gateways through registry mechanisms

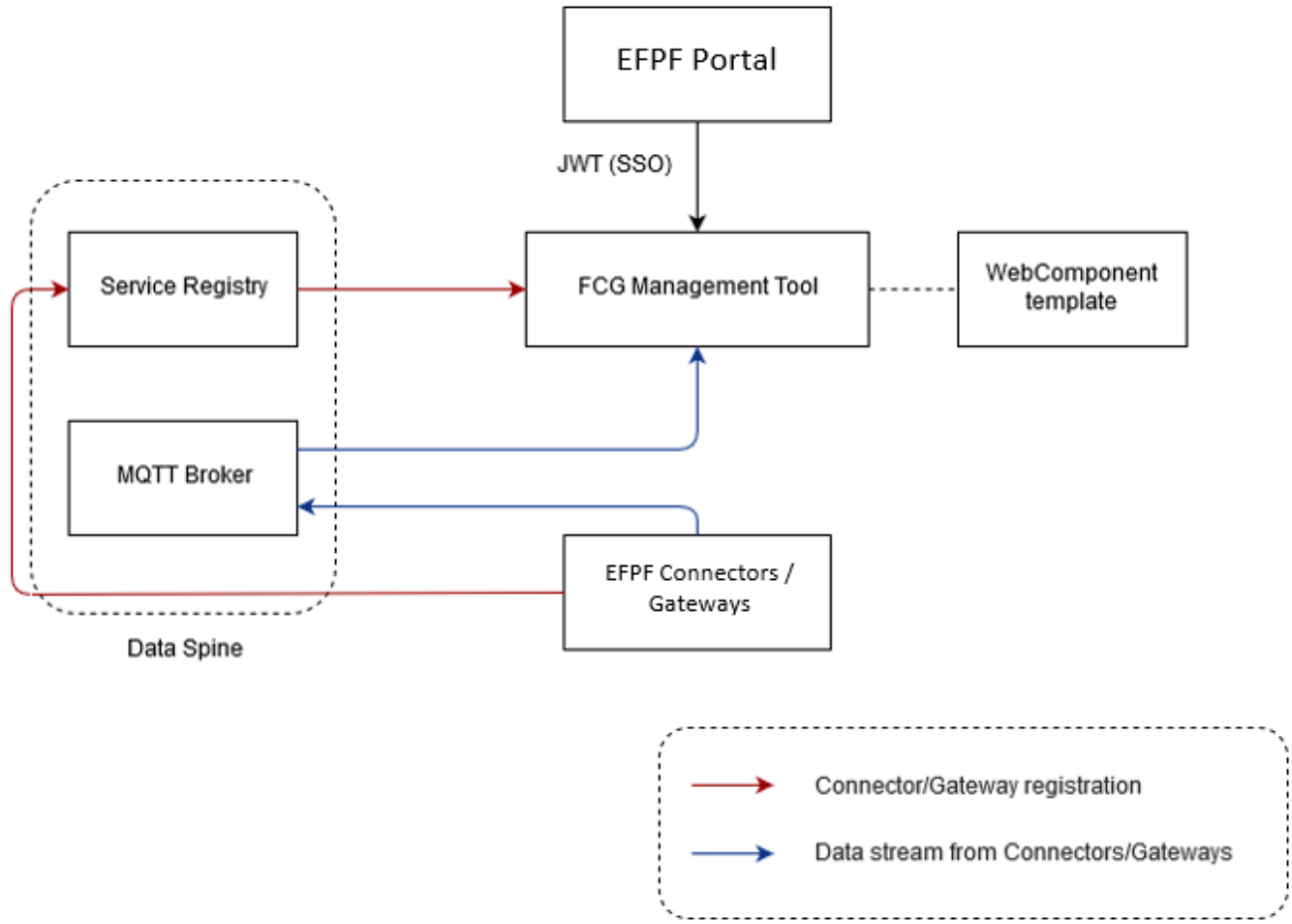

Figure 43: Architecture of the FCGMT

Figure 43 illustrates the architecture of the FCGMT. This is defined as a web component based on a shared template which is later integrated and accessed from the EFPF Portal.

On one hand, the Service Registry is the component from the Data Spine that provides the functionality required to register new factory connectors and gateways, so the FCGMT "consumes" this component to provide a manual registration of any new connector and gateway. The service registry may require extension or customisation to represent these connectors and gateways since there are additional properties such as the physical location of the connector that needs to be stored.

On the other hand, the MQTT broker component manages the network and exchanges the messages from/to the connectors and gateways. This component is also part of the Data Spine and provides mechanisms to get information from these resources, such as its corresponding data models, as well as subscribe to its data.

In general terms, the FCGMT needs to consume services from the Data Spine as follows:

- Request to GET the list of connectors/gateways available on the platform: Service Registry endpoint is required
- Request to GET the details of a particular connectors/gateway: Service Registry endpoint is required

- Request to GET the data model associated to a particular connector/gateway: Service Registry endpoint is required
- Request to POST a new connector/gateway to the platform: Service Registry endpoint is required
- Request to PUT changes made in a particular connector/gateway to the platform: Service Registry endpoint is required
- Request to DELETE a particular connector/gateway from the platform (Optional)

3.2.11 Secure Data Storage

The Secure Data Storage enables the persistent storage of data gathered through data analytics and factory connectors tools. Access to this component is secured by the EFS (Section 3.1.6), which is part of the Data Spine.

The Secure Data Storage is composed of the following components:

- **Authorisation Guard (UMA):** This component restricts the access to the stored data. By implementing User-Managed Access (UMA) it enables fine-grained access control
- **(Pseudo) Anonymiser:** This component allows developers to store relevant data as pseudonymised data sets (where personal data is involved)
- **Privacy Analysis:** The Privacy Analysis allows risk evaluation of the chosen form of pseudonymisation for sensitive personal data

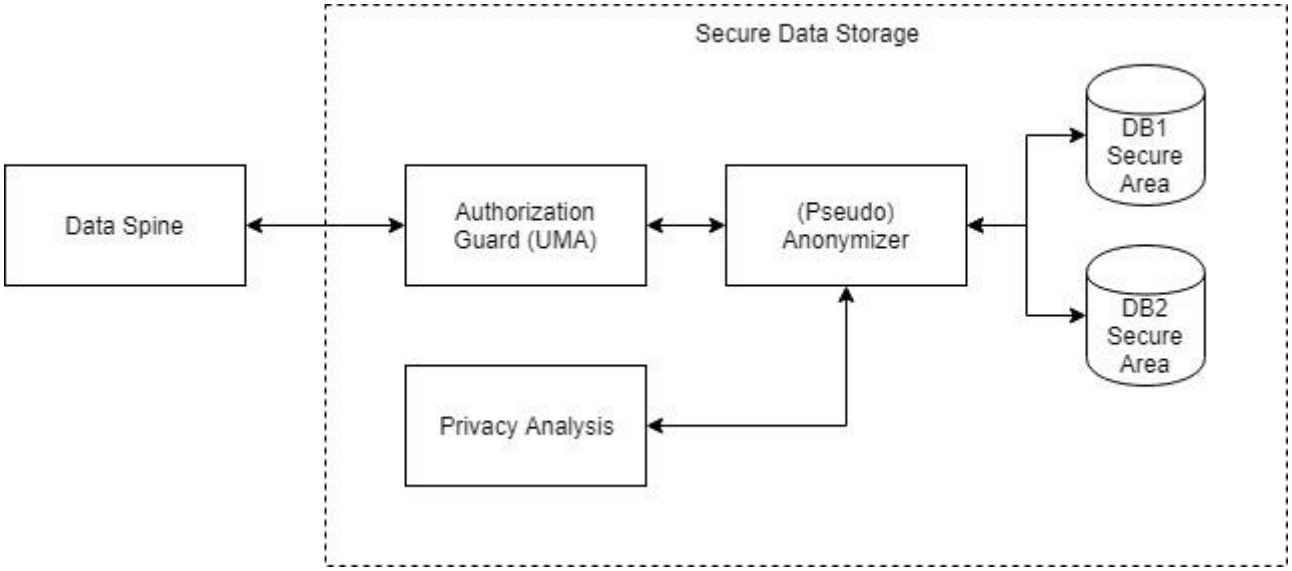

Figure 44: Secure Data Storage Architecture

The data to be stored, read, updated, or deleted is expected to use the NoSQL approach and JSON (JavaScript Object Notation). As a technical foundation, MongoDB is used, and respective queries can be used to filter data. Support for alternate databases is possible, with the restriction that the mechanism for specifying access controls managing privacy need to be able to address the database contents according to a URL mapping schema.

The complete component is setup as a docker image and usable anywhere, either in the cloud or at a local site. All the mentioned subcomponents are present in every instance of

the component. For this reason, data is not available globally but has to be identified via an access URL to identify the secure data storage instance that should be accessed.

Technical Foundation: Secure Data Storage manages the actual application of the specified data access controls. It utilises the UMA FCGMT, in conjunction with the OAuth and OpenID Connect standards, allowing data owners to specify access controls for stored data with rules that function within the greater context of the EFPF ecosystem.

3.2.12 Factory Connectors & Gateways

In order to access data from the numerous data sources available within the manufacturing facilities of members of the four base platforms, it is necessary to utilise a Connector or IOT Gateway that can interface with the devices, sensors and systems. This can be made available through a user defined data model for Smart Factory Tools to be able to operate. The purpose of data provided by these connectors varies from production status, alerting, Kanban / stock level monitoring, to energy consumption, machine/ process efficiency, and more.

In EFPF there are three implementations of Factory Connectors & Gateways, supporting between them the most widely used industrial standards and systems (e.g. OPC UA, Siemens, Rockwell, Omron, Schneider etc.). The generic scenario presented in Section 3.2.10 utilises all three implementations for Production Status to be monitored across a collaborative production process.

Figure 45 shows the high-level architecture of the three Connector / Gateways and the interaction with the EFPF Data Spine.

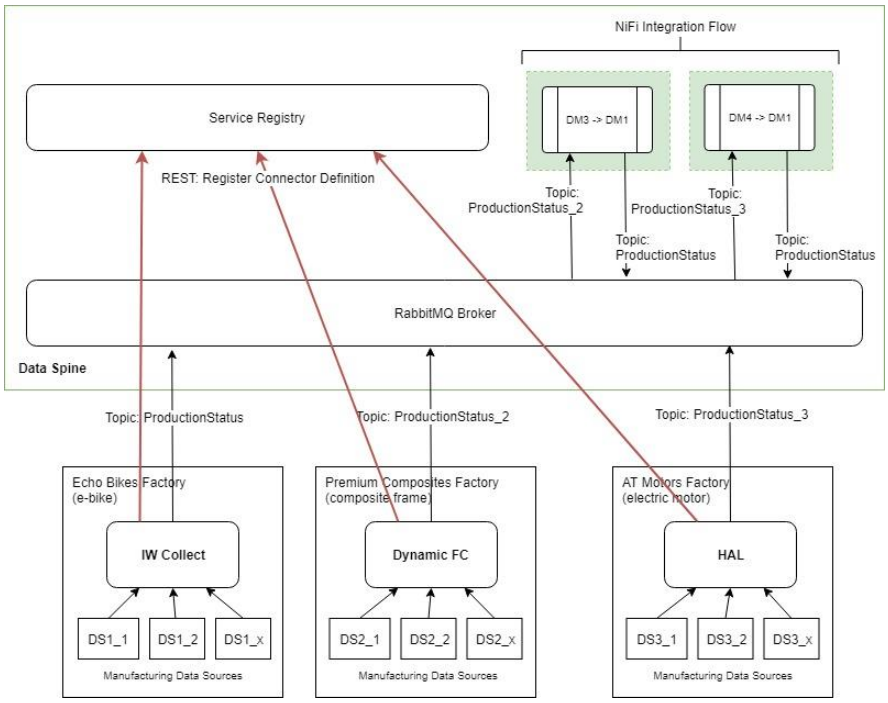

Figure 45: Factory Connectivity within EFPF

3.2.12.1 Industreweb Collect

Industreweb (IW) Collect is a high-speed data engine that interfaces with a range of systems and devices with the aim of extracting business critical data. The primary objective of IW Collect is to solve getting data from sources that may prove to be normally difficult or require a bespoke solution. All data sources are then transformed to a common data model to allow processing and event triggering.

Data sources can include industrial control systems (e.g. PLC, CNC) both modern models with Ethernet capability and legacy equipment, wireless networks and devices such as ZigBee, and industrial networks such as Profinet, Profibus, Modbus and AS-interface. It can also connect and interrogate databases such as MS SQL and MySql, and flat file formats such as XML and JSON.

The architecture of the IW Collect is shown in Figure 46 which is based around the concept of connectors to enable it to monitor a diverse range of data sources. Data acquired from connectors interfaced with the production systems and sensors. IW Collect runs on a Windows based embedded industrial PC platform that can be pre-installed on machines, or retro-fitted to existing machines. It requires Microsoft .Net 4.0 and optionally Microsoft SQL Server in order capture data. To commission the system, the industrial PC must firstly be interfaced with the production data sources, which typically involves physically connecting the required networks, and the installation of any intermediate 3rd party hardware such as network switches or wireless transmitter/ receivers. Once this has been carried out the interface settings are defined by editing the connector´s configuration file, which defines each data source connector and its properties necessary to function.

Following this stage, the rules to orchestrate the collection and manipulation of data are created which are based on logic events and subsequent actions. The system is then run in the background as a Windows Service, constantly monitoring the manufacturing process.

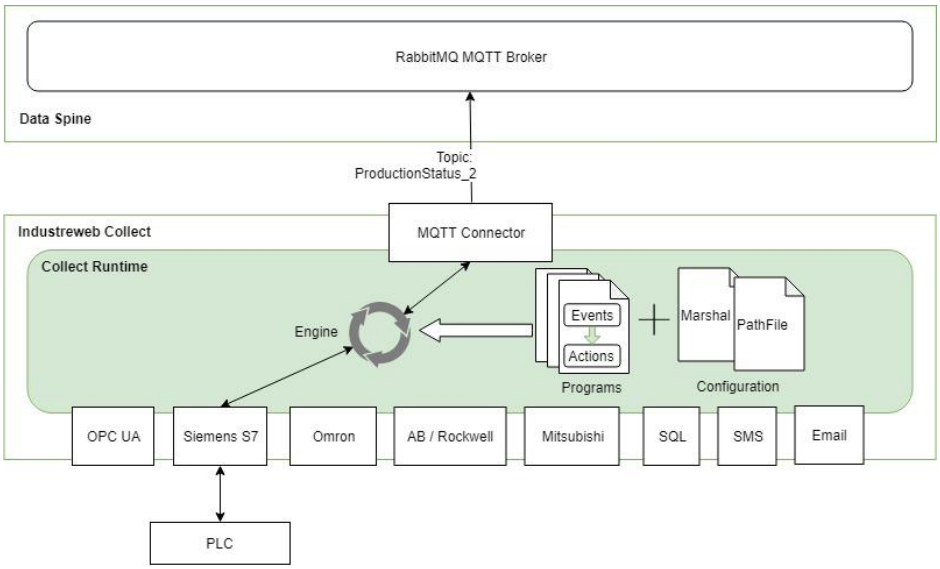

Figure 46: Industreweb Collect Architecture

Actions that IW Collect can trigger may include changing data in a connector, displaying an alert on a screen, sending an SMS or email, or writing a value to a database.

For interfacing with the MQTT broker in EFPF the data is collected from the range of production systems and sensors via the appropriate connector instance and mapped to the MQTT connector instance data model. Upon data changing within the process or at a set time interval the MQTT broker publishes the data using the “ProductionStatus” topic. This is then subscribed to by the Smart Factory Tools and Services.

3.2.12.2 Dynamic Factory Connectivity Service

The Dynamic Factory Connectivity (DyFC) primary objective is to enable the dynamic information sharing about the manufacturing process status among independent companies participating in a collaborative manufacturing network. The cross-factory information exchange implemented by the DyFC, considers connectivity and interoperability between various factory data sources and cloud-based services while addressing data control concerns and minimising integration efforts.

DyFC provides secure two-way communication between the factory data sources and the cloud-based platform and forms an on-premise endpoint. To monitor the collaborative manufacturing process and provide interoperability, DyFC uses a meta-model (inspired by ISA-95 standard) that enable the description of collaborative manufacturing processes as well as the information to be shared with the rest of the partners. Besides, DyFC implements an information exchange process, described in the following steps:

1. Upon receiving the collaborative manufacturing plan and the information required by the production monitoring service, DyFC supports the Factory Engineer (FE) in collecting the local factory data sources necessary for answering these information requests
2. DyFC also supports the FE in aggregating these local data sources to match the required semantic and syntactic descriptions, which is then mapped to the appropriate information requested by the service

3. After the setup process is established, the progress status of the manufacturing activities is automatically sent to the cloud-based service with the semantic and syntax described by the collaborative manufacturing monitoring service

The DyFC is composed of three main components, as illustrated in Figure 47.

- The Collaborative Process Twin acts as the digital representation of the collaborative manufacturing processes the company is involved in
- The Mapping and Aggregation Engine supports the FE in aggregating the local data sources and map it to the corresponding information requested by the service
- Local Connectivity is responsible for connecting to the various local data sources, using OPC UA protocol

In EFPF, the DyFC is used in the context of collaborative manufacturing network model involving SMEs since it is mostly suited for sharing information about the progress of manufacturing activities for monitoring and coordination purposes. DyFC targets manufacturing SMEs since it takes into account some of their adoption challenges such as data control.

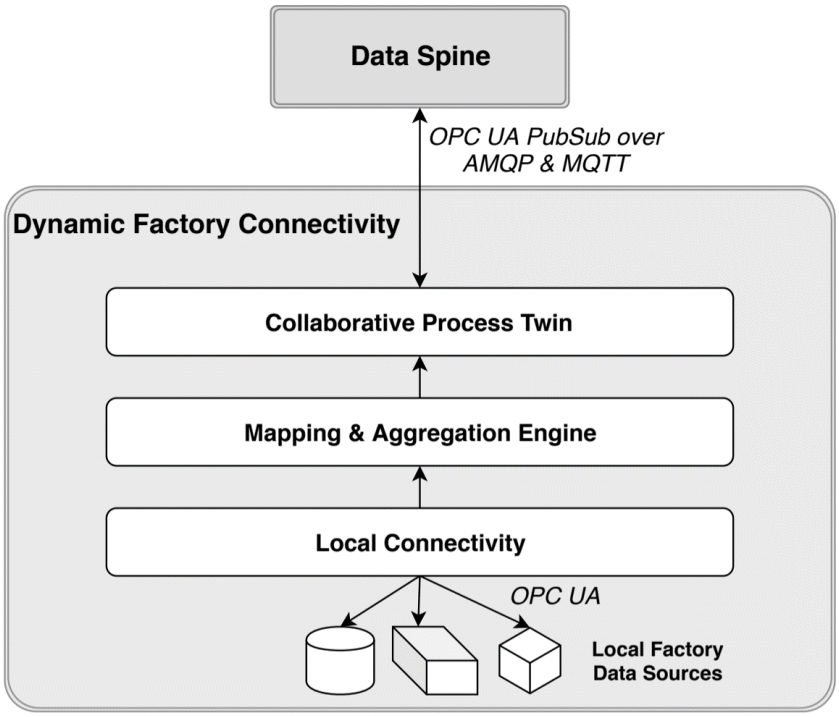

Figure 47: Dynamic Factory Connectivity Components

3.2.12.3 Symphony Hardware Abstraction Layer (HAL)

Hardware Abstraction Layer (HAL) is a software module that is part of NXW's Symphony platform[14]. It primarily abstracts the low-level details of various heterogeneous fieldbus technologies and provides a common interface to its users. It adapts the device protocols and provides the necessary logic to manage them accordingly to their respective constraints (e.g. timing constraints). It also implements optimisations, e.g. avoid spamming the KNX

[14] http://www.nextworks.it/en/products/brands/symphony

bus[15] with too many messages, pack contiguous Modbus reads into a single multi-register read.

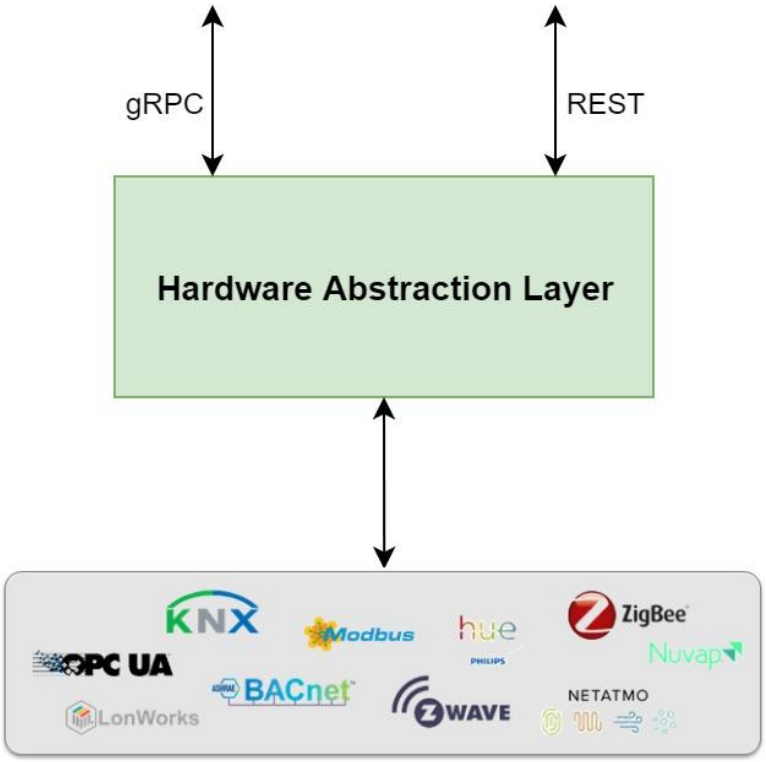

Figure 48: Symphony HAL

The HAL supports KNX, BACnet, Modbus-TCP and Modbus-RTU as well as, several other proprietary control protocols. It can be extended by developing modules that can be dynamically plugged into its core. It can be interconnected with specific field buses either directly (via RS232/485 serial ports or GPIOs) or through the use of IP based gateways, such as KNX IP router and/or interface, Modbus/TCP gateways.

The HAL component provides access to any available resources (sensors and actuators) as datapoints. The datapoints are primitive objects with basic data type (int, float, boolean) but devoid of any semantic annotation (physical object type, measurement unit, …) or are presented according to the OGC SensorThings data format standard. The HAL supports access via REST and gRPC and furthermore enables publish/ subscribe features via MQTT.

[15] https://www2.knx.org/no/knx/association/what-is-knx/index.php

4 Development & Deployment View

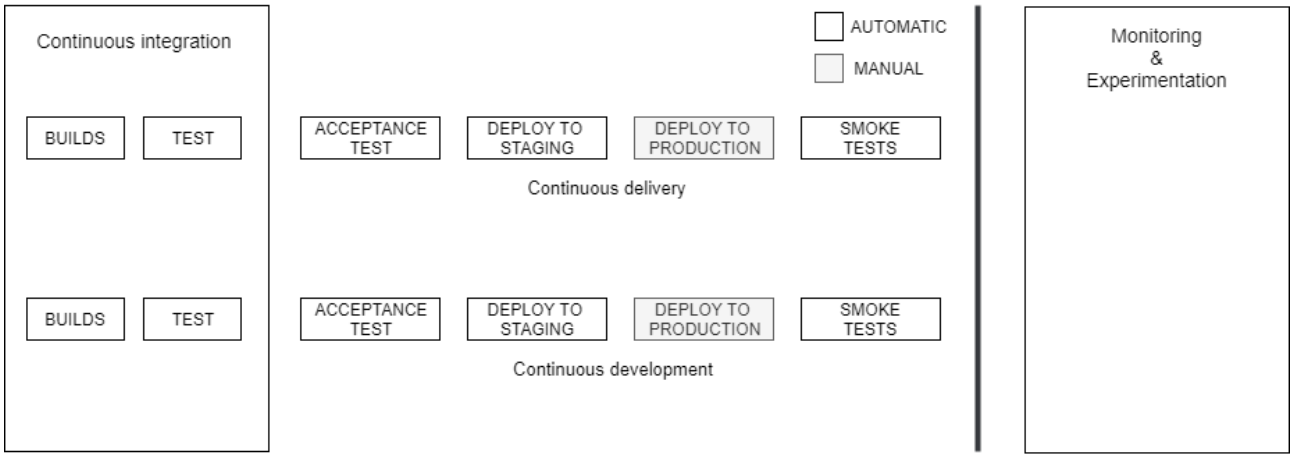

Figure 49: CI/CD Overview (adapted from www.atlassian.com)

The deployment of EFPF tools, services and components and the monitoring of the configured components on the platform, fall within the Deployment and Operational Viewpoints of the Architectural Description. The major architectural concerns here are on run-time architectural qualities that affect the operability and maintainability of the EFPF platform to ensure that the platform can support the large-scale experimentation activities and organic growth of the ecosystem.

The management of cloud deployments of component systems is supported by DevOps services [HF10]. A centralised repository is used for configuration and component management. To support development, deployment and operations the project has installed a GitLab instance. GitLab is a DevOps platform that offers functionality for project planning, source code management as well as continuous integration (CI), continuous delivery (CD) and monitoring.

The continuous integration and testing environment will be available for all EFPF base platform and service providers but mainly used for the core EFPF infrastructure. The repository will be successively populated, and adapted for integration of additional external base platforms and/or tools.

The monitoring framework will provide a global EFPF model for Quality of Service (QoS) with respect to the contractual obligations of an associated product quality model with micro services. This will include run-time quality attributes based on the ISO/IEC 25010 standard [ISO11] of product quality model, such as performance efficiency, reliability, security and maintainability.

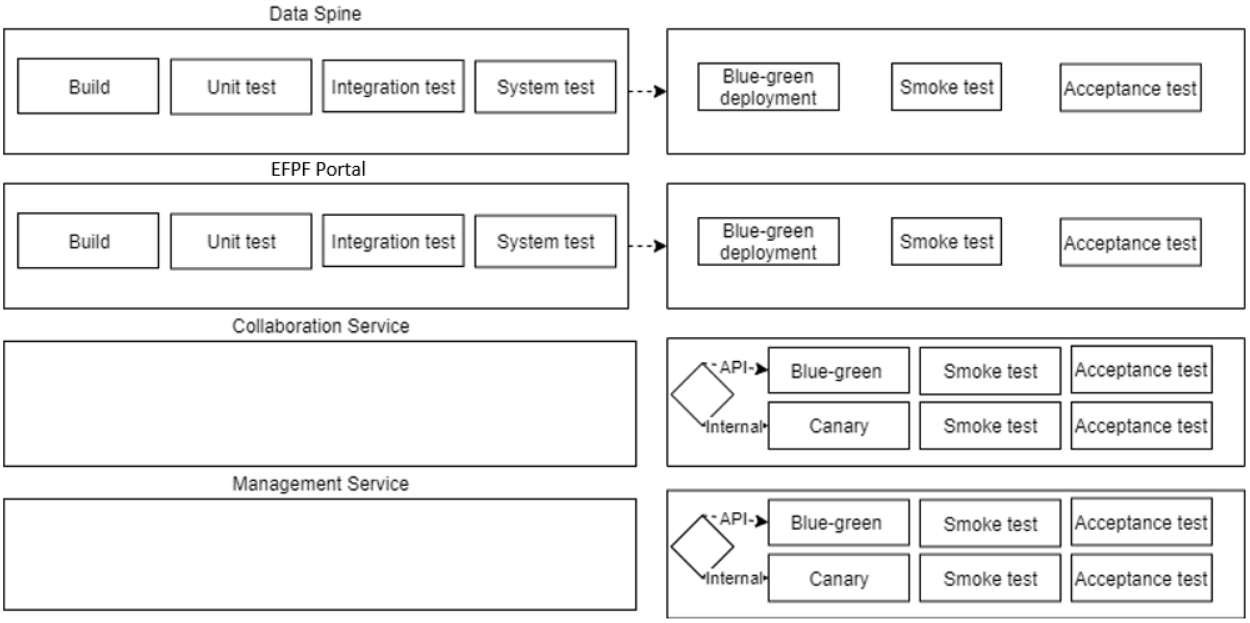

Figure 50: EFPF CI/CD Pipelines

The EFPF project employs a microservice architecture. Each microservice is developed by a dedicated team using its own parallel pipeline including test server, then be deployed to production. CI pipelines are internal to the microservice. New versions may be delivered to production using a suitable strategy and tested by EFPF in a black-box manner.

Marketplace services, base platforms and tools will be developed, deployed and managed by the owners of these resources. The Data Spine and EFPF Portal are deployed on partner infrastructure and development, tests and delivery are managed within GitLab.

4.1 Delivery

4.1.1 Dependencies

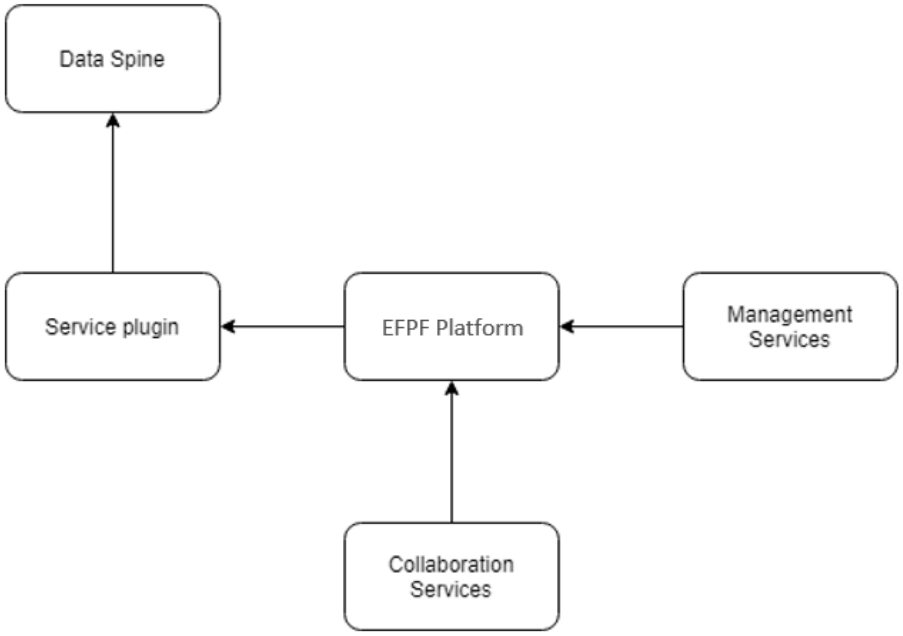

Figure 51: Integration Points Dependencies

4.1.2 Policies

The aspects of integration and delivery will be expressed in a set of policies; guidelines and rules to govern how microservices in the EFPF platform must deal with delivery, versioning and tests. They may also prescribe test coverage levels and a set of software metrics to ensure the quality of deployed code.

4.1.3 Frequency

Each microservice will manage its own delivery schedule, from continuous deployment to manual deployment, but the frequency of contract-changing/dependency-breaking updates needs to be controlled as described above. A system-wide deprecation schedule with long enough window until “sunset” may suffice. Conversely, parties responsible for a microservice will have to commit to updates frequent enough to accommodate for sunset of other services it depends on or urgent security fixes.

The policy for updates can include alternative strategies, depending on the type of update. For component code updates/fixes, Rolling updates are used to fully replace existing services, which will require high-availability roll-back and recovery services. An alternative strategy is to initially deploy the updates to a pre-selected subset of dependent service clients (referred to as Canary release), which implies a multi-stage update procedure. Finally, for API updates, the previous versions will be maintained in parallel to the updated service interfaces (so called Blue-green deployment), following a deprecation period.

4.1.4 Versioning

Given the microservice architecture of EFPF, the items that need to be version managed on project/EFPF level are the connectors and contracts between microservices: interfaces, data schemas and formats, protocols and standards. Code versioning internal to components are mainly of interest for error reporting and debugging. Policies for versioning need to be detailed, dealing with backward compatibility, depreciation schedules and parallel versions. Versions of microservices that do not break the contract (API) but only extend it or introduce bug fixes, performance optimisations or other internal changes, can be handled differently from changes that may break dependent components.

4.2 Monitoring

Monitoring of user-experienced problems during operation in the experimentation and exploitation phases of the EFPF system will use the Gitlab infrastructure that is already in place. The Gitlab Service Desk allows external users to create issues without an account, and support personnel and the developers responsible for a microservice to handle the issue using the same system where development is managed. This can be further integrated with team communication software like Slack for rapid collaborative problem-solving.

Continuous monitoring and logging infrastructure allow deep analysis of the performances of deployed software that can both be carried out before the final deployment and during real-world operation. As this kind of infrastructure often is integrated with the deployment platform (AWS, Azure, Google Cloud, or others) the choice of monitoring infrastructure is dependent on the choice of platform. There is also a wealth of free tools available to automate the monitoring of the status of endpoints, e.g. Postman [POS19] or PHP Server Monitor [PHP19]. In the exploitation phase, each stakeholder operating a microservice is likely to already employ an operational monitoring tool and the monitoring may be done through this.

5 Base Platforms and Interface Contracts

5.1 COMPOSITION

COMPOSITION (Ecosystem for Collaborative Manufacturing Processes – Intra- and Inter-factory Integration and Automation) is a digital platform that provides:

- An Integrated Information Management System (IIMS) which optimises the internal production processes by exploiting existing data, newly deployed sensors data, knowledge and tools to increase productivity, optimise factory procedures and dynamically adapt to changing market requirements.
- An online virtual ecosystem to support the interchange of data and services between factories and their suppliers with the aim to optimise and invite new market actors into the supply chain.

The COMPOSITION IIMS provides solutions to value chain related to predictive maintenance, decision support and asset tracking. The COMPOSITION Ecosystem provides solutions to supply chain related to smart waste management and an agent-based marketplace that provides to its participants an automated transactions ecosystem by offering an automated bidding process functionality enhanced by a blockchain application.

The deployment of COMPOSITION components is based on Docker [DOC19] containers and their management is enabled by using Portainer [POR19]. The communication mechanisms were based on HTTP protocol and on the setup of a RabbitMQ [RAB19] message broker for MQTT and AMQP messaging. From security perspective, KeyCloak [KEY19] for identity management and EPICA [EPI19] for access control were used. OGC SensorThings [OGC19] was adopted for the description of tools and system-generated data, and of sensors data as well.

The main components from the COMPOSITION project that are provided to EFPF ecosystem are listed below:

- **Marketplace 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. Two main categories of agents can be defined a priori, depending on the kind of provided services: Marketplace agents and Stakeholder agents. *Marketplace Agents f*ollowing FIPA [FIP19] specifications, an Agent Management System (AMS) is a mandatory component of every agent platform, and only one AMS should exist in every platform. *Stakeholder agents* are deployed at the stakeholder's premises and their purpose is to fulfil the stakeholder's interests.
- **Matchmaker,** the core component of COMPOSITION's semantic framework, is used by Marketplace's agents in order to match customers with suppliers and requests with offers in the COMPOSITION Marketplace. The component provides an Agent Level Matchmaking Module, which handles a request by an agent (through HTTP calls) for a requested service/product and after the appliance of a set of semantic rules in the response to the agent with a list contains the agents who support a matching offer for this request. Besides this, the component provides an Offer Matchmaking Module as well. An agent sends a request for available offers' evaluation to the Matchmaker which

applies automated weighted algorithms in order to rank the preferences of the requester(price, delivery time, rating, payment and delivery terms etc.) and then applies best score algorithms in order to suggest the best available offer. This component enables the automated bidding process of the project.

- **Analytic tools** of COMPOSITION provide solutions related to predictive maintenance and machine failures detection inside a factory, and solutions for supply chain procedures optimisation. The Forecasting Tool offers different types of analytic methodologies (Trend Analysis, Statistics, Local Outlier Factor, Machine Vibration Diagnosis Profile, Density Based Spatial Clustering, Markov Chain Models, Genetic algorithms). Moreover, a Deep Learning Toolkit based on ANNs has been deployed in order to predict failures in industrial ovens and to provide price forecasting for recycling materials. Finally, a web-based Visual Analytics tool has been implemented in order to provide advanced visualisations of the aforementioned algorithms. The tool is interactive with the end users and offers many types of graphs for data representation.
- **Blockchain** in COMPOSITION project aims to provide an audit trail for manufacturing and supply chain data, enabling both product data traceability and secure access for stakeholders. The implementation mechanism chosen for the blockchain was Multichain [MUL19], an open-source blockchain implementing the Bitcoin [BIT19] API. It provides configurable permissions for assets and consensus, high transaction speeds and several useful abstractions for dealing with general time-stamped data without the need to explicitly use cryptocash or other assets.
- **Symphony Building Management System** (BMS) is a complex system that requires specific configurations to be put in place before running on a shop-floor. Internal mechanisms are implemented to ensure scalability functionality with respect on the number of sensors installed and the amount of data transmitted by these devices. Nevertheless, the system is targeted on building and factory environments, that can be big scopes, but somehow bounded on predictable scales. On the other hand, if a single instance (currently set up for COMPOSITION) would not be enough for the intended purposes, there is no limit to the number of BMS instances that could be deployed, also because in a real application each factory has its own BMS instance.

5.2 NIMBLE

NIMBLE stands for the collaborative Network for Industry, Manufacturing, Business and Logistics in Europe. It provides infrastructure for a cloud-based, Industry 4.0, IoT-enabled B2B platform on which European manufacturing firms can register, publish machine-readable catalogues for products and services, search for suitable supply chain partners, negotiate contracts and supply logistics. Participating companies can establish private and secure B2B and Machine-to-Machine (M2M) information exchange channels to optimise business workflows. The infrastructure is developed as an open source software under an Apache type, permissive license. The governance model is a federation of platforms for multi-sided trade, with mandatory interoperation functions and optional added-value business functions that can be provided by third parties. Prospective NIMBLE providers can take the open source infrastructure and bundle it with sectoral, regional or functional added value services and launch a new platform in the federation.

In EFPF, the EFPF IDentity Provider (IDP) that enables user identification and authorisation based on Keycloak, is setup and hosted by the partner SRFG (NIMBLE project coordinator).

SRFG is also running specifically created microservice for translation of tokens between the four base platforms participating in the EFPF ecosystem.

5.3 DIGICOR

The DIGICOR platform is an open ICT platform with tools and services to support the management and control of collaboration networks. The platform provides basic services for security, safety, governance and knowledge protection, with access to automation systems and smart objects in the IoT by vertical integration via interfaces. It also provides a marketplace for ad-hoc setup of collaborations.

Two DIGICOR middleware architectures have emerged from the DIGICOR project. For the Aerospace domain a unified portal with integrated services and for the Automotive domain consuming selected DIGICOR Tools and Components as discreet services through an external platform.

5.3.1 Aerospace Domain Portal

The DIGICOR middleware architecture for the Aerospace use case uses micro-services as an architecture style combined with event-driven architecture (EDA) as an architectural pattern while leveraging Docker containers for deployment. The main drivers influencing the architectural design decisions were:

- DIGICOR is a dynamic ecosystem, therefore it requires a very loosely coupled architecture, supporting broadcast communication in a distributed style.
- The DIGICOR team is spread into several autonomous teams located in different locations with various expertise, thus the use of micro-services architecture.
- Decoupling into many independent services allows for choosing technology for each team independently as it is only the interface that matters.
- Many modern approaches (OPC UA as an example) are used in DIGICOR. The implementation of the functionality can be done using different technologies. Using small separate bounded services allow an easy redesign and integration of new technologies if necessary.
- The DIGICOR platform requires flexibility in terms of deployment as well as high availability. Micro-services run separately and can be updated without affecting the overall platform.

Figure 52 shows the service interaction between Company, Collaboration and the Factory Connector.

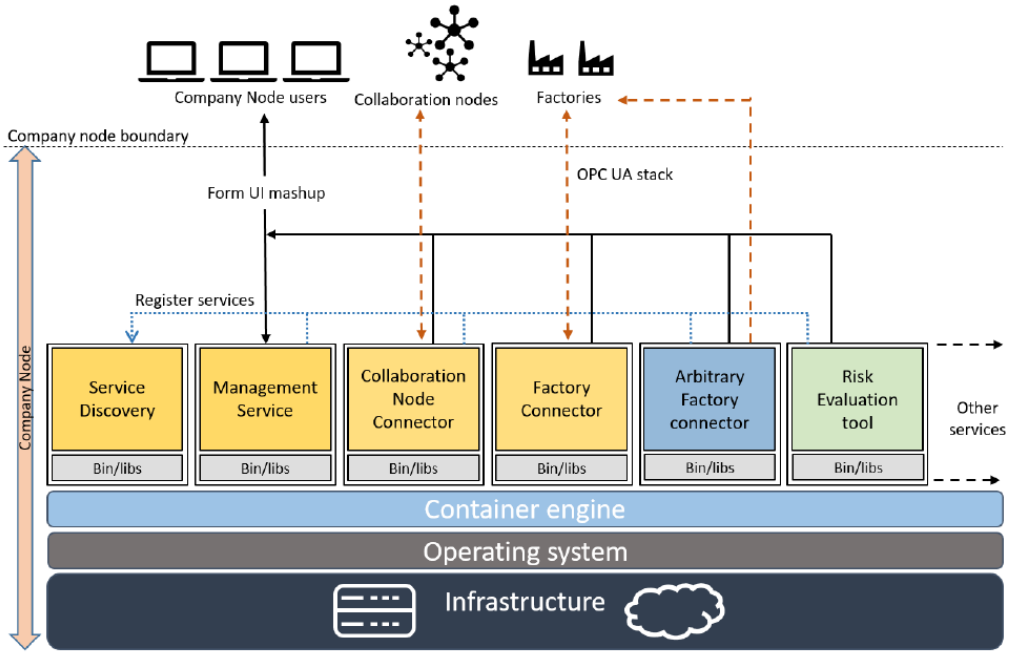

Figure 52: DIGICOR Component Interaction

5.3.2 Automotive Domain Service Delivery

In the automotive use case, DIGICOR forms a synergy with the SMECluster business platform. SMECluster provides Tools and Services via a marketplace available to its members. This marketplace includes tools from multiple platforms. It is also enabled for interoperability with DIGICOR tools and services, demonstrating how DIGICOR tools and services can gain additional market exposure. The business model of SMECluster is to offer opportunities to collaborate between members and to support this goal through readily available technology that will provide productivity and quality improvements.

The SMECluster platform architecture is based on a federation of service libraries orchestrated via calls from the integrated workflow engine and the client web UI, as illustrated in Figure 53: SMECluster Component Interaction. Whilst technology agnostic, the main stack runs on Microsoft .Net infrastructure under IIS, currently hosted on the SMECluster dedicated server but can equally be hosted on a cloud infrastructure. Data storage is provided by Microsoft SQL Server.

Figure 53 shows the interaction between services within the SMECluster platform, and those from the DIGICOR platform.

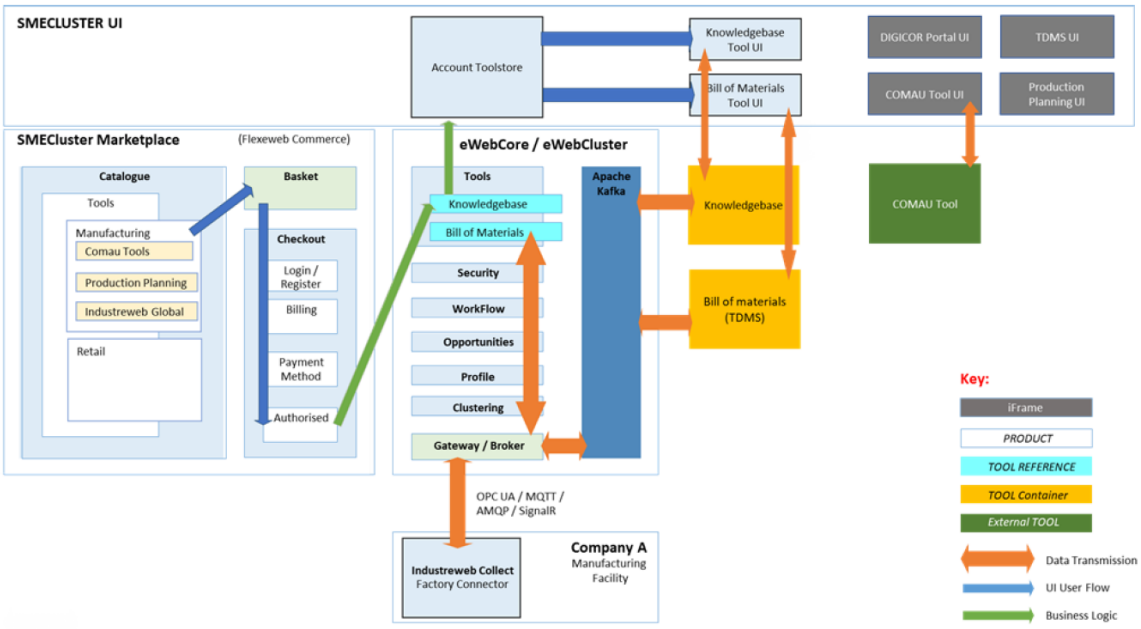

Figure 53: SMECluster Component Interaction

5.4 vf-OS

vf-OS (Virtual Factory Operating System) is a digital platform that provides an operating system targeted to the development of applications for the manufacturing companies. The platform provides mechanisms that interact with the real factory assets, Cloud-based services and the Internet to automate, enhance or optimise different types of processes and activities that take place in manufacturing environments.

One of the key offerings of the vf-OS platform is a framework named virtual factory Open Application development Kit (vf-OAK). The vf-OAK is composed of a set of integrated tools. The central component of the vf-OAK is its Software Development Kit (SDK), an environment for the development of applications and, generically, for the centralised access of the vf-OS assets and functionalities. This SDK is complemented with a GUI that forms its frontend Interface, the vf-OS Studio, which includes tools for code development, editors that include features like Business Process Management Notation (BPMN), syntax highlighting and drag-and-drop, debugging, and analysis of the developed code. The Studio also encapsulates other tools developed in the scope of the project, such as a Frontend Composer, a Process Designer, a System Dashboard, and provides access to the vf-Store marketplace for development and publishing of the products.

In this respect, the main components of the vf-OS platform or the vf-OS OAK covering different application development needs are listed below:

- **vf-OS Process Enabler:** Graphical environment for the developer to design vApps by connecting different vf-OS assets
- **vf-OS Frontend Environment:** Graphical framework that facilitates the rapid composition of stylesheets vApps' frontends

- **vf-OS Service Development Kit (SDK):** The kernel and heart of the vf-OS Framework, putting all needed vf-OS resources at the tip of the developers' fingers by means of libraries
- **vf-OS Studio:** vf-OS development IDE, which provides the necessary tools and means to develop, publish, and deploy vApps on the marketplace
- **vf-OS IO Toolkit:** The toolkit helps developers in integrating and developing connectors to devices (drivers) and other software (APIs)
- **vf-OS Data Mapping:** The data mapping component enables to design ETLs to support data integration activities between different document schemas
- **vf-OS Data Analytics:** The data analytics component allows developing and deploying models for solving analytics manufacturing problems and needs
- **vf-OS Enablers Framework:** This framework is used to facilitate the usage of FIWARE, Manufacturing and vf-OS Specific Enablers which encapsulate repetitive operations

After going through several validations (piloting and hackathons), the vf-OS platform is made available through the EFPF Portal (https://efpf-portal.ascora.eu/). The vf-OS platform is hosted by EFPF partner ICE, whereas the vf-OS Marketplace (separate web-based portal) is hosted by the EFPF partner ASC.

5.5 Interface Contracts Between EFPF and Base Platforms

This section describes the interface contracts between EFPF and the base platforms. Interface contracts generally define the association between two components/APIs/services and may consist of one or more clauses defining expected behaviours at method call boundaries.

The purpose of defining interface contracts is to introduce a certain level of transparency and expectancy in the EFPF platform, regarding the provisioning and behaviour of base platform services. This means, that the base platform and EFPF developers can refer to interface contracts (this section) to analyse any impact of their future updates.

5.5.1 Interface to Factory Data: Industreweb Display

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	SMECluster (DIGICOR)
Base Platform Service	Industreweb Display
Base Platform Service Provider	Control 2K Ltd
Expected Functionality (as-is) of Base Platform Service	Asynchronous web dashboard for production KPI's and Operator displays
Status of Base Platform Service	Implemented Validated in DIGICOR and commercial scenarios. The Industreweb Display has been customised to provide production monitoring dashboard that is integrated with EFPF Portal

Base Platform Service Ownership and Provisioning	Service provided, hosted, and maintained by Control 2K
Base Platform Security Protocols	Integrated security based on .Net Security Provider
Dependent EFPF Services	As a UI component, there is no EFPF service dependent on Industreweb Display. The Industreweb Display receives SignalR data from an integrated Industreweb Collect Engine, which subscribes to data on the EFPF broker (RabbitMQ v3.7.18 – supports AMQP1.0 and MQTT3.1)
Artefacts depended upon or exposed to dependents	Data format is: JSON by default but can be defined by user

5.5.2 Shop-floor Connectivity: IndustreWeb Factory Connector

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	DIGICOR
Base Platform Service	IndustreWeb Factory Connector
Base Platform Service Provider	Control 2K Ltd
Expected Functionality (as-is) of Base Platform Service	Connectivity with shop-floor assets (machines, sensors etc) and porting of shop-floor data to higher level information systems
Status of Base Platform Service	Implemented Validated in DIGICOR and commercial scenarios. The connector is already used in EFPF to extract shop-floor data and make it available to applications through the Data Spine
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner C2K
Base Platform Security Protocols	Custom security setup that can be tuned to user needs
Dependent EFPF Services	The following EFPF tools/services make use of this service: • Risk Management Tool • Event Manager
Artefacts depended upon or exposed to dependents	

5.5.3 Blockchain for Monitoring of Distributed Activities: COMPOSITION Blockchain

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	COMPOSITION
Base Platform Service	COMPOSITION Blockchain
Base Platform Service Provider	CNet
Expected Functionality (as-is) of Base Platform Service	Agent Blockchain API, generalized for use in other contexts. The blockchain infrastructure is enhanced in enhanced in EFPF support the delivery tracking application
Status of Base Platform Service	Implemented Validated in COMPOSITION. The Blockchain solution is being enhanced and generalised for use in EFPF pilots
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner CNet. COMPOSITION server hosted and maintained by partner FIT
Base Platform Security Protocols	Multichain API 2.0, basic auth.
Dependent EFPF Services	The following EFPF tools/services make use of this service: • DApp POC
Artefacts depended upon or exposed to dependents	Blockchain API (Swagger specification)

5.5.4 Agile Collaborations: Online Bidding Process

Contract Party – Service Consumer	EFPF Marketplace
Contract Party – Service Provider (Base Platform)	COMPOSITION
Base Platform Service	Online Bidding Process
Base Platform Service Provider	LINKS
Expected Functionality (as-is) of Base Platform Service	Automatic negotiation between service requesters and suppliers
Status of Base Platform Service	Implemented Validated in COMPOSITION. The bidding process is being generalised for use in EFPF pilots and open call experiments

Base Platform Service Ownership and Provisioning	Service provided maintained for the EFPF project by partner LINKS
Base Platform Security Protocols	HTTPS, security by COMPOSITION Border Gateway
Dependent EFPF Services	The following EFPF tools/services make use of this service: • EFPF Integrated Marketplace Framework • EFPF Bidding Process UI • CERTH Matchmaker
Artefacts depended upon or exposed to dependents	HTTP REST API, AMQP Broker, Proprietary data format

5.5.5 Data Analytics: Deep Learning Toolkit

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	COMPOSITION
Base Platform Service	Deep Learning Toolkit
Base Platform Service Provider	LINKS (COMPOSITION and EFPF Partner)
Expected Functionality (as-is) of Base Platform Service	Predictive Maintenance and Price Prediction of waste material.
Status of Base Platform Service	Implemented Validated in COMPOSITION. The toolkit is integrated with the EFPF portal
Base Platform Service Ownership and Provisioning	Service provided maintained for the EFPF project by partner LINKS
Base Platform Security Protocols	No security provided within the tool, security provided by COMPOSITION border gateway
Dependent EFPF Services	The following EFPF tools/services make use of this service: • Data Analytics Dashboard • Data Analytics (CERTH)
Artefacts depended upon or exposed to dependents	HTTP Rest/RPC, proprietary data format

5.5.6 Indexing Service: NIMBLE Indexing Service API

Contract Party – Service Consumer	EFPF Marketplace
Contract Party – Service Provider (Base Platform)	Nimble

Base Platform Service	Nimble Indexing Service API
Base Platform Service Provider	SRFG
Expected Functionality (as-is) of Base Platform Service	Product provisioning
Status of Base Platform Service	Implemented Validated in NIMBLE
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner SRFG
Base Platform Security Protocols	HTTPS, OAuth2, OpenID
Dependent EFPF Services	The following EFPF tools/services make use of this service: • EFPF Integrated Marketplace Framework
Artefacts depended upon or exposed to dependents	

5.5.7 Catalogue Service: NIMBLE Catalogue REST API

Contract Party – Service Consumer	EFPF Marketplace
Contract Party – Service Provider (Base Platform)	Nimble
Base Platform Service	Nimble Catalogue REST API
Base Platform Service Provider	SRFG
Expected Functionality (as-is) of Base Platform Service	Product image provisioning
Status of Base Platform Service	Implemented
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner SRFG
Base Platform Security Protocols	HTTPS, OAuth2, OpenID
Dependent EFPF Services	The following EFPF tools/services make use of this service: • EFPF Integrated Marketplace Framework
Artefacts depended upon or exposed to dependents	

5.5.8 Product Provisioning (vf-OS)

Contract Party – Service Consumer	EFPF Marketplace

Contract Party – Service Provider (Base Platform)	vf-OS
Base Platform Service	vf-OS Marketplace
Base Platform Service Provider	ASC
Expected Functionality (as-is) of Base Platform Service	Product provisioning
Status of Base Platform Service	Implemented Validated in vf-OS. The marketplace is already integrated with the EFPF Portal through single sign-on and integrated marketplace search service
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner ASC
Base Platform Security Protocols	HTTPS, API token
Dependent EFPF Services	The following EFPF tools/services make use of this service: • EFPF Integrated Marketplace Framework
Artefacts depended upon or exposed to dependents	

5.5.9 Product Provisioning (SMECluster)

Contract Party – Service Consumer	EFPF Marketplace
Contract Party – Service Provider (Base Platform)	DIGICOR/SMECluster
Base Platform Service	Search for Products
Base Platform Service Provider	Control 2K
Expected Functionality (as-is) of Base Platform Service	Product search and provisioning
Status of Base Platform Service	Implemented Validated in commercial scenarios. The product provisioning service is integrated with EFPF Portal through the integrated marketplace search service
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner Control 2K
Base Platform Security Protocols	HTTPS
Dependent EFPF Services	The following EFPF tools/services make use of this service: • EFPF Integrated Marketplace Framework

Artefacts depended upon or exposed to dependents	

5.5.10 Event Reactor: Symphony Event Reactor

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	Symphony (External Platform)
Base Platform Service	Symphony Event Reactor
Base Platform Service Provider	Nextworks
Expected Functionality (as-is) of Base Platform Service	Ability to trigger actions and alarms through Event Manager/Alarm Manager modules, depending on the type of the event. It has logging and lifecycle systems for alarms
Status of Base Platform Service	Implemented Validated in commercial applications. The Event Reactor is being integrated in the EFPF shop-floor connectivity solution
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained by Nextworks
Base Platform Security Protocols	Custom security setup that can be tuned to user needs
Dependent EFPF Services	Input to/from EFPF message broker, messages from Event Broker are mapped within the Data Spine (NiFi)
Artefacts depended upon or exposed to dependents	APIs: REST Communication protocols: MQTT, AMQP

5.5.11 Hardware Abstraction: Symphony Hardware Abstraction Layer

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	Symphony (External Platform)
Base Platform Service	Symphony Hardware Abstraction Layer (HAL)
Base Platform Service Provider	Nextworks
Expected Functionality (as-is) of Base Platform Service	It primarily abstracts the low-level details of various heterogeneous fieldbus technologies and provides a common interface to its users.

Status of Base Platform Service	Implemented Tested and validated in production environment
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained by Nextworks
Base Platform Security Protocols	Custom security setup that can be tuned on user needs
Dependent EFPF Services	Output published to the EFPF Broker
Artefacts depended upon or exposed to dependents	APIs: REST, gRPC Communication protocols: MQTT, gRPC, KNX, BACnet, Modbus, etc. Data formats: OGC SensorThings, SAREF, SAREF4B, proprietary

5.5.12 Risk Analysis: SSM Risk Management Tool

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	SSM (External Platform)
Base Platform Service	SSM Risk Management Tool
Base Platform Service Provider	University of Southampton IT Innovation Centre
Expected Functionality (as-is) of Base Platform Service	Risk Analysis tool
Status of Base Platform Service	Implemented Validated in previous EU projects
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner University of Southampton IT Innovation Centre
Base Platform Security Protocols	Custom security setup that can be tuned to user needs
Dependent EFPF Services	N/A
Artefacts depended upon or exposed to dependents	API, security protocol, communication protocol…

5.5.13 Partner and Capability Search: Federated Search

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	NIMBLE
Base Platform Service	Search service

Base Platform Service Provider	Salzburg Research (SRFG)
Expected Functionality (as-is) of Base Platform Service	Federated Search: Keyword based/facet-based search on the products/services and companies
Status of Base Platform Service	Implemented Validated in NIMBLE. Federated Search data indexing Nifi workflows, Apache Solr index are already created for EFPF platform and hosted by SRFG
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner Salzburg Research
Base Platform Security Protocols	API protected by authorization headers
Dependent EFPF Services	Product/Service/Company search in EFPF portal, Product Recommendation Service Matchmaking Service
Artefacts depended upon or exposed to dependents	REST API JSON message format

5.5.14 Identity Service: Identity Service and Central Identity Provider

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	NIMBLE
Base Platform Service	Identity-Service & Central Identity Provider (Keycloak)
Base Platform Service Provider	Salzburg Research (SRFG)
Expected Functionality (as-is) of Base Platform Service	User authentication & authorization, Single Sign On (SSO)
Status of Base Platform Service	Implemented Already integrated with EFPF platform and provides user authentication to EFPF portal and single-sign-on to the EFPF users across the connected base-platforms.
Base Platform Service Ownership and Provisioning	Service provided, hosted and maintained for the EFPF project by partner Salzburg Research (https://efpf-security-portal.salzburgresearch.at/)
Base Platform Security Protocols	HTTPS, security provided by Keycloak identity management server
Dependent EFPF Services	• EFPF portal login • User federation in EFPF platform

Artefacts depended upon or exposed to dependents	Identity Service REST API Keycloak integration workflow with base-platform to provide user-federation and SSO

5.5.15 Factory Connectivity: Industreweb Collect

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	SMECluster (DIGICOR)
Base Platform Service	Industreweb Collect
Base Platform Service Provider	Control 2K
Expected Functionality (as-is) of Base Platform Service	Interfaces with all automation systems within the manufacturing facility including legacy, wireless sensor, network protocols and OPC UA
Status of Base Platform Service	Implemented Validated in commercial, DIGICOR and other past EU projects. The Industryweb Collect solution is being used in the EFPF shop-floor connectivity solution
Base Platform Service Ownership and Provisioning	Service provided, hosted, and maintained by Control 2K
Base Platform Security Protocols	Integrated security based on .Net Security Provider
Dependent EFPF Services	Publishes data to EFPF broker (RabbitMQ v3.7.18 – supports AMQP1.0 and MQTT3.1) Will subscribe with EFPF Service Registry via REST interface
Artefacts depended upon or exposed to dependents	• Communication protocol is: User definable data model including OPC UA using PubSub over MQTT/MQTTs and AMQP/AMQPs or OPC UA client/server • Data format is: JSON but default but can be defined by user • Data models: Production data model (ISA-95 inspired model–used in DIGICOR)

5.5.16 Blockchain as a Service

Contract Party – Service Consumer	EFPF

Contract Party – Service Provider (Base Platform)	CERTH
Base Platform Service	Blockchain-as-a-Service (BaaS)
Base Platform Service Provider	CERTH (COMPOSITION and EFPF Partner)
Expected Functionality (as-is) of Base Platform Service	BaaS provides services related to Registration Management, Identity Management and Permission Management
Status of Base Platform Service	Implemented Tested and validated within other projects – Integrated with EFPF registration service
Base Platform Service Ownership and Provisioning	BaaS functionalities are provided, maintained and hosted by CERTH.
Base Platform Security Protocols	Security based on authentication JWT (JSON Web Token)

Dependent EFPF Services	The following EFPF tools/services make use of this service:
	• EFPF Portal – Registration process
	• It will also be used in
	  ○ Inform/Consent and Revoke Consent
	  ○ Permission management
	  ○ Circular Economy scenario

Artefacts depended upon or exposed to dependents	• Provide a REST API • Security is protocols JWT (JSON Web Token) • Communication protocol is HTTPS / RPC • Data format is JSON

5.5.17 Data Analytics: Data and Visual Analytics Toolkit

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	COMPOSITION
Base Platform Service	Data and Visual Analytics Toolkit
Base Platform Service Provider	CERTH (COMPOSITION and EFPF Partner)
Expected Functionality (as-is) of Base Platform Service	Provide analytics services to pilot partners in Circular Economy scenario. Services are: fill level sensors' monitoring and trend analysis of fill level, tonnage forecasting, price forecasting based on Deep Learning, vibration sensors monitoring and vibration profile real time analysis

Status of Base Platform Service	Implemented Tested and validated within COMPOSITION – Integrated with EFPF and it is available through EFPF portal to the corresponding users
Base Platform Service Ownership and Provisioning	Both analytics services and sensors provided and maintained by CERTH. Only price forecasting is maintained by LINKS. CERTH hosts the visual analytics platform and the sensors are deployed on KLEEMANN and ELDIA premises.
Base Platform Security Protocols	Security is based on Keycloak identity and access management
Dependent EFPF Services	The following EFPF tools/services make use of this service: • EFPF Portal – data analytics section
Artefacts depended upon or exposed to dependents	• Available to the users as a web-based UI (Chart.js and D3.js are used) • Security is based on Keycloak • Communication protocols are both HTTPS and MQTT for sensors and analytics communication • Data format is JSON for sensors and analytics communication – Interaction with end users is possible through UIs

5.5.18 Semantic Framework

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	COMPOSITION
Base Platform Service	Semantic Framework
Base Platform Service Provider	CERTH (COMPOSITION and EFPF Partner)
Expected Functionality (as-is) of Base Platform Service	Provide information about COMPOSITION companies and services. Moreover, the framework provides the matchmaking functionalities of COMPOSITION project
Status of Base Platform Service	Implemented Tested and validated within COMPOSITION – Integrated with EFPF Federated Search service by using Apache NiFi
Base Platform Service Ownership and Provisioning	Service provided and maintained by CERTH, it is hosted by FIT in EFPF Portainer/Server with other COMPOSITION components. This server is maintained by FIT as well.

Base Platform Security Protocols	Security based on authentication using OpenID Connect provided by FIT
Dependent EFPF Services	The following EFPF tools/services make use of this service: • Matchmaker component for federated search and online bidding process • Marketplace for online bidding process
Artefacts depended upon or exposed to dependents	• Provide a REST API • Security protocol is Basic Auth • Communication protocol is HTTPS • Data format is JSON

5.5.19 Factory Connectivity: Dynamic Factory Connector

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	DIGICOR
Base Platform Service	Dynamic Factory Connector
Base Platform Service Provider	fortiss
Expected Functionality (as-is) of Base Platform Service	Aggregates and maps local factory data sources to the information required/specified by the services
Status of Base Platform Service	Implemented Tested and validated within DIGICOR – Integrated with EFPF production monitoring scenario using Apache NiFi
Base Platform Service Ownership and Provisioning	Service provided, hosted, and maintained by fortiss
Base Platform Security Protocols	OPC UA security set up that can be adjusted
Dependent EFPF Services	Publishes data to EFPF broker (RabbitMQ v3.7.18 – supports AMQP1.0 and MQTT3.1)
Artefacts depended upon or exposed to dependents	• Communication protocol is: OPC UA PubSub over MQTT/MQTTs and AMQP/AMQPs, OPC UA client/server • Data format is: JSON • Data models: Production data model (ISA-95 inspired model–used in DIGICOR)

5.5.20 Supply Chain Visibility: iQluster

Contract Party – Service Consumer	EFPF
Contract Party – Service Provider (Base Platform)	iQluster
Base Platform Service	iQluster
Base Platform Service Provider	Valuechain Ltd
Expected Functionality (as-is) of Base Platform Service	Collaboration and intelligence platform that streamlines intercompany communication and securely captures multi-tier supply chain intelligence so that organisations can increase network competitiveness.
Status of Base Platform Service	Implemented Tested and validated within commercial scenarios – Integrated with EFPFSingle-Sign-On
Base Platform Service Ownership and Provisioning	Service provided, hosted, and maintained by Valuechain
Base Platform Security Protocols	Keycloak based security access and authorisation
Dependent EFPF Services	Matchmaking (Federated Search) and Business and Network Intelligence Service
Artefacts depended upon or exposed to dependents	

Annex A: History

Document History	
Versions	V1.0 • Updates based on internal and external (technical reviewers') comments and quality check. • Further extensions include Service Contract descriptions and relations to base platform services and tools & inclusion of Annex C & D V0.9 • Aggregated inputs from partners based on comments from the second internal review V0.85 • Aggregated inputs from partners based on comments from the first internal review and sent for the second internal review V0.81 • Did basic formatting, updated references and sent for the first internal review V0.8 • Aggregated inputs from partners based on review comments V0.71 • Restructured the document and added review comments V0.7 • Added contributions from SRFG (Section 3.2.5), FIT (Section 0) V0.6 • Added contributions from SRFG (Section 3.1.2, 3.1.6, 3.2.4, 3.2.5, 4.1.2) V0.5 • Added contributions from CNET and CERTH (Section 3.2.7, 4, 5.1.1) V0.4 • Consolidated contributions from FIT (Section 1, 2, 3.1 and 3.2.1), ASC (Section 3.2.2, 3.2.3 and 3.2.11), VLC (Section 3.2.6), C2K (Section 3.2.10, 3.2.10.1, 3.2.12, 3.2.12.1), NXW (Section 3.2.10.2, 3.2.12.3), ALM (Section 3.2.10.3), AID (Section 3.2.10.4), FOR (Section 3.2.12.2) V0.3 • ICE contribution under Section 3 under data analytics V0.2 • ICE contribution under Section 2, 3 and 5.4 (Section numbers from the latest version of this document) V0.1 • Base structure and initial content added by FIT

Contributions	FIT: • Rohit Deshmukh • Alexander Schneider ICE: • Usman Wajid SRFG: • Violeta Damjanovic-Behrendt • Nirojan Selvanathan • Dileepa Jayakody ASC: • Norman Wessel • Brian Clark VLC: • Happy Dudee CNET: • Mathias Axling • Matts Ahlsen CERTH: • Alexandros Nizamis • Sofia Terzi C2K: • Simon Osborne NXW: • Matteo Pardi • Alì Nejabati ALM: • Carolyn Langen AID: • Fernando Gigante FOR: • Nisrine Bnouhanna SRDC: • Senan Postaci • Yildiray Kabak LINKS: • Luigi Giugliano • Jure Rosso

Annex B: References

[Hil00] Hilliard, Rich. "Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems." IEEE, http://standards. ieee. org 12.16-20 (2000): 2000.

[IEEE 42010, 2011] May, I. S. O. Systems and software engineering–architecture description. Technical Report. ISO/IEC/IEEE 42010, 2011.

[RW05] Rozanski, Nick, and Eoin Woods. "Software Systems Architecture: Viewpoint Oriented System Development." (2005).

[Mor10] Morrison, J. Paul. Flow-Based Programming: A new approach to application development. CreateSpace, 2010.

[Shu86] Shu, Nan C. "Visual programming languages: A perspective and a dimensional analysis." Visual Languages. Springer, Boston, MA, 1986.

[KRU04] Kruchten, P. (2004). The Rational Unified Process: An Introduction. Addison-Wesley Professional.

[DP18] Dunphy, P. and Petitcolas, F. A. P. (2018): A First Look at Identity Management Schemes on the Blockchain, in IEEE Security & Privacy, vol. 16, no. 4, pp. 20-29.

[SD16] Samaniego, M. and Deters, R. (2016): Blockchain as a Service for IoT. 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, 2016, pp. 433-436.

[HF10] Humble, J., Farley, D (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional.

[ISO11] ISO (2011). ISO/IEC 25010:2011. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. Accessed Sept 2019.

[POS19] Postman Inc. (2019). Intro to Monitoring. https://www.getpostman.com/docs/v6/postman/monitors/intro_monitors. Accessed Sept 2019.

[PHP19] phpservermonitor.org (2019). PHP Server Monitor: Open source tool to monitor your servers and websites, http://www.phpservermonitor.org. Accessed Sept 2019.

[DOC19] Docker Enterprise Container Platform. https://www.docker.com/ Accessed Sept 2019.

[RAB19] RabbitMQ Open-source message-broker. https://www.rabbitmq.com/ Accessed Sept 2019.

[POR19] Portainer for Docker Management. https://www.portainer.io/ Accessed Sept 2019.

[KEY19] Keycloak Open-source identity and access management solution https://www.keycloak.org/ Accessed Sept 2019.

[EPI19] Efficient and Privacy-respectful Interoperable Cloud-based Authorization. http://tredisec.eu/content/epica Accessed Sept 2019.

[OGC19] OGC SensorThings API. http://www.opengeospatial.org/standards/sensorthings Accessed Sept 2019.

[FIP19] FIPA Foundation for Intelligent Physical Agents. http://www.fipa.org/ Accessed Sept 2019.

[MUL19] MultiChain Enterprise Blockchain. https://www.multichain.com/ Accessed Sept 2019.

[BIT19] Bitcoin Open-source P2P money. https://bitcoin.org/ Accessed Sept 2019.

[BMH18] Y. Bounagui, A. Mezrioui, H. Hafiddi, 2018. “Toward a Unified Framework for Cloud Computing Governance: An Approach for Evaluating and Integrating IT Management and Governance Models” In Computer Standards & Interfaces (2018). doi: https://doi.org/10.1016/j.csi.2018.09.001

Annex C: Platform Profiles

Platform*	Component Name*	Description	Maturity Level* (TRL 0-9)	Exposed Interfaces* (Protocol | Data Model | Data Format | Access Control | Identity Provider)	Dependency*	Programming Environment	Deployment	Partner	Contact Person	Technical Doc
Composition	Deep Learning Toolkit	Maintenance prediction using deep learning techniques	TRL4	HTTP, RPC| JSON |User Defined | |	Pyro-ns | External identity provider	Python, Keras, TensorFlow	Docker Image	LINKS	luigi.giugliano@linksfoundation.com	link
Composition	Composition Marketplace	Multi agent based marketplace	TRL 5	HTTP AMQP MQTT | JSON | User defined | Keycloak OIDC | |	Platform AMQP&MQTT Broker | External identity provider	Python	Docker Image	LINKS	jure.rosso@linksfoundation.com	
Composition	Log Oriented Architecture	Block-chain architecture for manufacturing & supply chain	TRL4	HTTP | JSON | JSON | |	Multichain, COMPOSITION intra-factory Auth	Python	Docker image	CNET	mathias.axling@cnet.se	
Composition	Big Data Analytics Tools (LinkSmart Learning Agent)	Toolset to support various business intelligence tasks	TRL4	HTTP | JSON | OGC-ST and Proprietary | OpenID Connect | Keycloak, MQTT | JSON | OGC-ST and Proprietary | MQTT Auth | Keycloak		Python, Java	Docker image, JAR	FIT		link
Composition	Semantic Matchmaking Framework	Interoperability, customers/suppliers matching, online offers' evaluation - core component of COMPOSITION Agent Marketplace	TRL5	HTTP |JSON | Proprieraty |HTTPs Basic Auth | Keycloak |JSON | Proprieraty |OpenID Connect | Keycloak |	Self-contained Ontology	Java, OWL	Docker Image	CERTH	alnizami@iti.gr	link
Composition	Forecasting Toolkit	Prediction engine for predictive maintenance and supply chain optimization(trend analysis, statistics, markov models, LOF etc.) - part of COMPOSITION data analytics tools	TRL5	HTTP | JSON | OGC Sensor Things | MQTT Auth | Keycloak, MQTT | JSON | OGC O&M | Isolated Container | None	Platform MQTT Broker	Python	Docker Image	CERTH	alnizami@iti.gr	link
Composition	LinkSmart Service Catalog	Service registry and discovery	TRL7	HTTP | JSON | Proprietary | OpenID Connect | Keycloak, MQTT | JSON | Proprietary | MQTT Auth | Keycloak	Platform MQTT Broker	Go	Docker Image, Binary Distribution	FIT	farshid.tavakolizadeh@fit.fraunhofer.de	link
Composition/ Symphony	Hardware Abstraction Layer (HAL)	Gateway for different field protocols (e.g. KNX, Modbus)	TRL 9	REST, CORBA, gRPC, raw | proprietary (oneM2M/SAREF-based) | proprietary | proprietary (OAuth) | Proprietary	Symphony base infrastructure	C++, Python, Go	Binary (Docker image coming soon)	NXW	m.pardi@nextworks.it	
Composition/ Symphony	Data storage	Data storage with pluggable back-ends (Postgres, Elastic Search, Cassandra)	TRL 9	REST, raw, AMQP, MQTT | None | SQL, JSON | proprietary | Proprietary	Symphony base infrastructure + HAL |	Python	Docker Image, Binary Distribution	NXW	m.pardi@nextworks.it	

					AMQP/MQTT broker					
Symphony	Event reactor	Event reactor and logic engine, with graphical UI	TRL 9	REST, CORBA, raw, AMQP, MQTT | proprietary (oneM2M/SAREF-based) | proprietary | proprietary (OAuth) | Proprietary	Symphony base infrastructure + HAL	C++, Python	Binary (Docker image coming soon)	NXW	m.pardi@nextworks.it	
Symphony	Visualization app	Customizable app to interact with the system (iOS, Android, web)	TRL 9	REST, gRPC, raw | - | binary | proprietary | proprietary	Symphony base infrastructure	ObjC, Java, Java/GWT	Binary	NXW	m.pardi@nextworks.it	
vf-OS	Marketplace	Multi-sided Marketplace for consumers and providers	TRL7	HTTP | JSON | JSON | Binaries | Basic Authentication |None	Ascora FIPS	PHP	Docker Image	ASC	hinz@ascora.de	link
vf-OS	Frontend Environment	Frontend Editor	TRL5	HTTP | JSON | JSON | Binaries | Basic Authentication |None	None	NodeJS, Express	Docker Image	ASC	klasen@ascora.de	link
NIMBLE	Platform Frontend	The endpoint where we can reach each platform as users	TRL4	HTTP | JSON | UBL | OAUTH | Keycloak		Java, JavaScript	Docker Image	SRFG		link
NIMBLE	Identity Service	Identity service	TRL4	HTTP | JSON | UBL | OAUTH | Keycloak		Java, Spring Boot	Docker Image	SRFG		link
NIMBLE	Catalogue Service	Management / service catalogues	TRL5	HTTP | JSON | UBL | OAUTH | Keycloak	NIMBLE scope modules	Java, Spring Boot	Docker Image	SRDC	suat@srdc.com.tr	link
NIMBLE	Business Process Service	Manages B2B collaboration	TRL5	HTTP | JSON | UBL | OAUTH | Keycloak	NIMBLE scope modules	Java, Spring Boot	Docker Image	SRDC	suat@srdc.com.tr	link
NIMBLE	Data Channel Service	Channels for enabling the exchange of sensor data		HTTP | JSON | UBL | OAUTH | Keycloak			Docker Image	SRFG		link
Valuechain	iQluster	Supply chain visualisation and intelligence generation	TRL 7	HTTP|JSON	none	.Net				
DIGICOR	Process engine	Design and execution of processes	TRL 4	HTTP | JSON | BPMN2.0 | a version w/ keycloak exists |	None	Java, Spring, Liferay	Docker Image	ICE	cesar.marin@informationcatalyst.com	
DIGICOR	Dynamic Factory Connectivity Service	Assists in connecting local site (factory floor) data to information requests by cloud tools monitoring collaborative manufacturing projects. Consists of a cloud service and edge nodes connected over a firewall friendly channel.	TRL 4	OPC UA Client-Server Binary, OPC UA PubSub over AMQP, AMQP 1.0, Eventuate REST/STOMP | JSON | Proprietary (DIGICOR Production Data Model), OPC UA	AMQP 1.0 Broker / Eventuate (cloud service only)	Java & JavaScript		FOR	bnouhanna@fortiss.org	
DIGICOR	Workflow and service automation	Design and execution of processes	TRL 4	HTTP | JSON | BPMN2.0 | a version w/ keycloak exists |	None. Standalone tool	Java, Spring, Liferay	Docker Image	ICE	cesar.marin@informationcatalyst.com	link
DIGICOR	TDMS	Tender decomposition and matchmaking service	TRL 5	HTTP | JSON	None	Java, R	Docker Image	Uni Manchester	n.mehandjiev@manc	

									hester.ac.uk	
DIGICOR	DigiGov	Creating and monitoring governance for supply chain collaborations	TRL 3	HTTP | JSON	TDMS	Java	Docker Image	Uni Manchester	n.mehandjiev@manchester.ac.uk	
DIGICOR	Opera	Generating workflows for collaborative tender preparation	TRL 1	HTTP | JSON | BPMN	TDMS, DigiGov	Java	Docker Image	Uni Manchester	n.mehandjiev@manchester.ac.uk	
SMECluster	Industreweb Collect Factory Connector	Manufacturing Data Collection and Orchestration Gateway	TRL9	OPC ClientServer | OPC DA | OPC DA OPC UA ClientServer | OPC UA binary or XML | Proprietary OPC UA PubSub | JSON | Proprietary SignalR Websockets | Websocket, JSON Payload AMQP/MQTT | JSON | Proprietary | Basic MQTT authentication	Manufacturing control systems (PLC, Robot, CNC…)	C#.Net, configuration by proprietary logic rule language	Physical deployment at machine level (edge computing)	C2K	sosborne@control2k.co.uk	
SMECluster	Flexeweb Commerce	Enterprise E-Commerce Engine	TRL9	REST Web Api | JSON | Proprietary		C#.Net, XAML Workflow design	Integrated package installer	C2K	sosborne@control2k.co.uk	link
SMECluster	SMECluster eWebCore API Services	Service library for collaboration Services within SMECluster	TRL8	REST Web Api | JSON | Proprietary				C2K	sosborne@control2k.co.uk	

Annex D: Survey of Platforms for Realising Data Spine

Platform	License	Language	Plugin/Extension Mechanism	Supported Languages for Plugins	Hot Plugin Deployment	REST/API Management	Reverse Proxy Support	Identity and Access Management	Type	Message Bus	Relation to Data Spine conceptual component
Apache NiFi	Apache License, Version 2.0	Java	Yes. Points of extension: Processors, Controller Services, Reporting Tasks, Prioritizers, and Customer Uis	Java. Scripted Processors (Clojure, ECMA Script, lua, Groovy, Python, Ruby)	No. Putting 'plugin' to the NiFi lib directory and restarting NiFi is required.	To manage NiFi instance: REST API	Yes, using reverse proxy server such as Nginx	**Authentication**: client certificates (Kerberos), username/password (LDAP), Apache Knox , OpenId Connect **Authorization**: Multi-Tenant Authorization	Data-flow management / data logistics tool based on the concepts of flow-based programming	NiFi is complementary to Messaging Queues such as Kafka	NiFi Platform: Integration Flow Engine Processors: Service Plugin Adapters
WSO2 Integration Agile Platform	Apache License, Version 2 and commercial licenses	Java	Yes	Java	No.	To manage APIs of integrated services: WSO2 API Manager	Yes, using WSO2 API Manager	WSO2 Identity Server	Integration Agile Platform to develop, reuse, run and manage integrations	WSO2 Message Broker	WSO2 Enterprise Integrator: Integration Flow Engine WSO2 ESB and WSO2 Message Broker: Message Bus
FIWARE	GNU Affero General Public License v3.0 and other	Broker: C++; IoT agents: NGSI protocol	Yes, in the form of IoT agents	-	-	NGSI protocol to interact with Orion Context Broker	No built-in reverse proxy mechanism	FIWARE Keyrock: OAuth2-based authentication and authorization security	IoT platform - Microservices architecture, with message broker in center, and IoT agents interacting with broker through NGSI protocol	The Orion Context Broker is the central message broker for all messages	Orion Context Broker: Integration Flow Engine IoT agent: Service Plugin Adapter FIWARE Keyrock: Security & Identity Management
Symphony BMS	Symphony BMS	Commercial	C++ / Python / Go	Yes, Points of extension: field-bus drivers, output data formats, data storage backends, customization of UIs	C++ / Python / Go	No	Yes (not available for all modules)	Yes	Fully fledged standalone automation platform	COBRA	-

Apache Camel	Apache License, Version 2.0	Java	Yes. Supports custom processors [c1, c2]	Java	Yes	For interacting with the Camel context, the available endpoints and routes: REST API	Yes	Route Security, Shiro Security, Spring Security, Payload Security, Endpoint Security, Configuration Security	Message-oriented middleware that supports configuration of routing and mediation rules to implement the various Enterprise Integration Patterns [c3]	Camel can be viewed as a Message Bus itself., Camel also supports various JMS providers and Message Brokers such as Kafka.	Camel Platform: Integration Flow Engine Processors: Service Plugin Adapters Talend ESB and Red Hat Fuse are both based on Apache Camel. Both have Eclipse-based development environment.
LinkSmart® Platform	Apache License, Version 2.0	Go, Java, Javascript	As new microservices	Any	As new microservices	No separate component for managing APIs	Yes, in LinkSmart® Border Gateway	Yes, using OpenID Connect (OIDC) provider such as Keycloak	IoT platform - microservices for device abstraction, security, data management (storage & visualization), service provisioning and online data mining	No message bus	Service Catalog: Service Registry, Border Gateway: API Security Gateway
Apache Flink	Apache License, Version 2.0	Java and Scala	-	-	-	-	-	-	Stream processing framework (and also a batch processing framework)	-	Flink as an option for Data Transformations in Integration Flow Engine
Confluent Platform	Apache License, Version 2.0, & commercial	-	-	-	-	-	Confluent REST Proxy	-	Event streaming platform	-	-

European Factory Platform

www.efpf.org