Collaborative Network for Industry, Manufacturing, Business and Logistics in Europe

D2.1

Platform Architecture Specification and Component Design

Project Acronym	NIMBLE	
Project Title	Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe	
Project Number	723810	
Work Package	WP2 Platform Technology Specification	
Lead Beneficiary	IBM	
Editor	Benny Mandler	IBM
Reviewers	Suat Gonul	SRDC
	Wernher Behrendt	SRFG
Contributors	EB	ALL
Dissemination Level	PU	
Contractual Delivery Date	30/04/2017	
Actual Delivery Date	30/04/2017	
Version	V1.0	

Abstract

The NIMBLE project aims to perform research leading to the development of a cloud and IoT platform specifically targeted to supply chain relationships and logistics. Core capabilities will enable firms to register, publish machine-readable catalogues for products and services, search for suitable supply chain partners, negotiate contracts and supply logistics, and develop private and secure information exchange channels between firms. The intention is to support a federation of such NIMBLE instances, all providing a set of core services, and each potentially specifically tailored to a different aspect (regional, sectorial, topical, etc.).

The main goal of this document is to explain in detail the architecture of the proposed platform including main aspects driving it and the way the proposed design will enable achieving the ambitious goals set out in the proposal. This document was thought of and written with the requirements documents (D1.1 and D1.2) in mind, i.e. it was informed by the groups working on these documents in parallel.

This deliverable describes the design, intended use, and validation of various aspects of the NIMBLE platform, via the use cases. It describes the main components along with the interactions among them. An iterative approach led to this version of the document, after laying out the foundations early on and fleshing it out in more detail as the implementation and understanding has matured.

We start from a high-level description of the architecture and then delve into more detailed explanation of the different components, and the interactions thereof. In addition, the main ideas and interactions are demonstrated through the introduction of the platform as viewed by the main stakeholders, as well as by mapping the intended use-cases to the proposed architecture.

Document History

Version	Date	Comments
V0.1	21/11/2016	Initial version and assignments distribution
V0.2	25/11/2016	Integrate ToC related comments from SRDC & SRFG)
V0.3	12/02/2017	Added more detailed IBM contributions
V0.4	28/02/2017	Semantic modelling (UB)
V0.5	28/02/2017	Data aspects (HOL)
V0.6	28/02/2017	Services discovery (UB)
V0.7	28/02/2017	Information Quality (UB)
V0.8	02/03/2017	Textile Manufacturing Supply Chain section (DO)
V0.9	02/03/2017	Registry / business processes (SRDC)
V0.10	02/03/2017	Security (SRFG)
V0.11	23/03/2017	Incorporate SRFG contributions on microservices and front-end
V0.12	03/04/2017	Incorporate security refinements (SRFG)
V0.13	12/04/2017	Version ready for internal review
V0.14	28/04/17	Incorporate final comments from Suat
V0.15	30/04/17	Incorporate final comments from Wernher Behrendt
V1.0	30/04/2017	Consolidate a final version

Table of Contents

List of Figures

List of Tables

Acronyms

Table 1: Acronyms

Acronym	Meaning
ACL	Access Control
API	Application Programming Interface
BPMN	Business Process Model and Notation
B2B	Business to Business
CA	Certification Authority
CRUD	Create, Read, Update and Delete
DoS	Denial of Service
ELK	ElasticSearch, Logstash, Kibana
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IDM	Identity Management
IIoT/ Industrie 4.0	Industrial Internet of Things
IoT	Internet of Things
IQ	Information quality
JWT	JSON Web Token
NIMBLE	Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe
PaaS	Platform as a Service
RAMI	Reference Architecture Model Industrie 4.0
SDK	Software Development Kit
SaaS	Software as a Service
SSO	Single Sign On
TLS	Transport Layer Security
UAA	User Account and Authentication
UBL	Universal Business Language
WP	Work Package
WSDL	Web Service Definition Language
XaaS	Everything as a Service

Glossary (ALL)

- Activiti - Business Process Management
- amalgam8 – service discovery and Content-based Routing Fabric for Polyglot Microservices
- Angular – develop applications for different deployment platforms
- Apache Jena - Java framework for building Semantic Web applications, by handling storage and query capabilities of RDF triplets
- Camunda - platform for workflow and business process management.
- Cloud Foundry – an open and extensible Platform as a Service offering
- Consul - Service Discovery and Configuration
- Consumable - Designing an easy to use product
- CouchBase – Scalable document oriented NoSQL data store
- ElasticSearch - high-performance indexing and search system that can be integrated with the CouchBase scalable data back-end.
- Eureka (Netflix) - Service registry and discovery component
- External applications – Independent applications that are developed and hosted outside the platform but internally make use of NIMBLE applications.
- GoRouter – component that acts as a front-end for cloud foundry for accessing internal applications
- MQTTT – A lightweight connectivity protocol that is used for bi-directional communication between the NIMBLE platform and the smart objects in the physical world.
- NIMBLE applications – cloud and IoT applications, which make use of NIMBLE core Services, and are deployed within the NIMBLE cloud platform
- Platform-as-a-Service - a cloud computing concept which offers the developer and deployer of cloud based applications the infrastructure, both HW and middleware, needed for creating and deploying successfully such applications on a cloud environment.
- RDF - A standard model for data interchange on the Web. Used in NIMBLE for storing entities description in a searchable manner.

- SPARQL – an RDF query language.
- SPARK – Open, distributed, real-time streams processing engine; used within the NIMBLE data management component.
- User Account and Authentication – The Cloud Identity Management component.
- YAML - human-readable configuration format

1 Introduction

NIMBLE is aiming at creating a B2B platform specifically geared towards improving the efficiency of supply chain creation and operations. At its core lie several services which enable companies to publish digital versions of their catalogues, containing the range of products they sell and business services (e.g. transportation, packaging and so on) they provide, and in turn enables other companies to efficiently search and find required counterparts. In addition, once potential partners find each other they are able to initiate a negotiations process through the platform and finally establish a supply chain relationship among them, including the creation of private information exchange channels.

The Industrial Internet of Things (IIoT or Industrie 4.0), including IoT in manufacturing and supply chain is expected to disrupt the global economy. This pushes companies to make their processes capable of utilizing IoT. Enhanced Supply chain services shall be IoT-driven and specialized for particular products and processes such as production facilities in a sub-contractor network. NIMBLE is creating a federated platform making entities and capabilities, discoverable and negotiable. The project will enhance existing IIoT specifications and technologies, to enable dynamic real-time IoT based services, such as water-based painting services being traded through the supply chain. The platform architecture will be demonstrated via several use cases in the field.

Ever more companies and processes are becoming internet connected and digitized, which influences both internal company processes as well as opens the door for additional efficiency of processes across entities. In the process, vast amount of data is produced that could be put into effective use via a dynamic but flexible and privacy preserving data sharing mechanism across entities. In parallel not enough attention has been placed on creating an ecosystem for enabling companies, large and small, to join such a network without heavy investments in IT. Industry 4.0 (Or IIoT – the industrial internet of things) has been coined as the term to describe such an environment. Even though there are various efforts for providing platforms for the IIoT, easy to use and federated platforms targeting small and large companies seem to be missing. Such a platform would level the playing field by enabling smaller companies to participate in advanced supply chain scenarios and encourage advancements towards incorporating digital processes in general and IoT devices in particular to participate in cloud enabled complex inter-company applications.

NIMBLE aims to tackle that ambitious goal, and along the way unleash the full potential of connected supply chain and logistics. The platform would enable entities to register with the platform, publish their capabilities and services, making them discoverable, and participate in resulting supply chain engagements. These basic capabilities will be enhanced at a second phase with more advanced capabilities such as enabling the selective sharing of data among partners.

NIMBLE aims to provide a technological platform for easily establishing various kinds of business relationships between different players. Therefore, the simplification of using and participating in the platform is a centre piece of the NIMBLE architecture. NIMBLE envisions an easy-to-use front end, which will serve as a single point of interaction for external companies, while hiding all the complexities of a cloud based platform at the back-end. Thus, users will enjoy the automation without suffering from the complexity of achieving it.

Nevertheless, NIMBLE will still be flexible to be extended by sector-specific providers of platform instances addressing particular needs of entities operating in that specific sector.

NIMBLE will ultimately provide an open platform infrastructure, which can be deployed in multiple instances, which collaborate among themselves in a federated manner. A set of core

services is to be supported by each such instance, while specific customizations and additional capabilities can be provided by each such instance.

Figure 1 depicts a high level schematic view of the main components of the NIMBLE platform. The main interaction point that external entities have with the platform is via the Front-end, which acts as a gateway for accessing internal components and capabilities. In addition, there's a programmatic way for accessing platform capabilities, both for the use of external applications, and for the federation layer of the platform itself. In addition, relevant data is fed into the platform to be processed, stored, and potentially shared via the data management component, in a programmatic manner.

At a first stage the front-end will enable external entities to interact with the platform for publishing, searching, and establishing relationships among different entities.

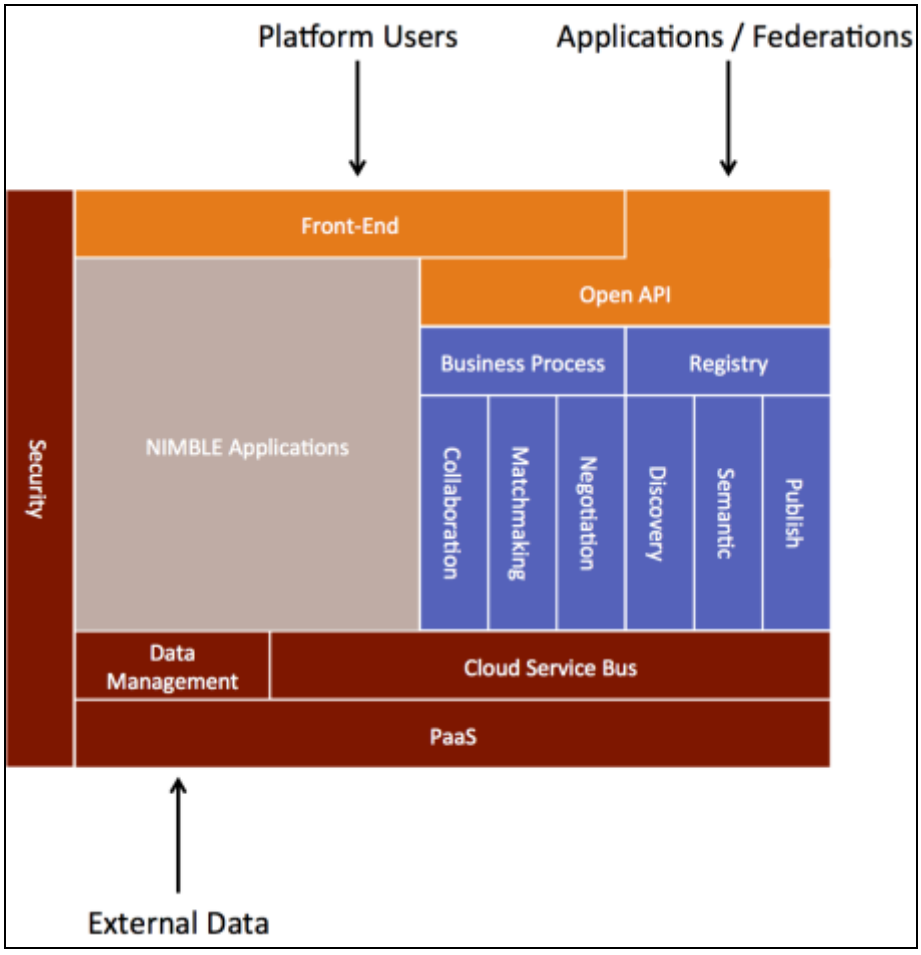

Figure 1: Main Components

The *Data Management* layer oversees the absorption of all data flowing in from external sources and performs required storage and transformations. *Security* interactions take place at different layers of the platform, such as ensuring that only the correct parties are allowed to operate within the platform and that they operate within their designated context. In addition, it makes certain that data flows only among authorized entities.

Components marked in red represent the base infrastructure on which NIMBLE entities run and communicate. The cloud-based run-time will host all NIMBLE entities, while the *Cloud Service Bus* will ensure proper communication and notification services between running entities.

A *Registry* component enables making the connection between different entities, by supporting entities publishing their capabilities to it, have these capabilities be semantically enhanced, and made discoverable via a search engine.

Once the basic connection has been established between entities, the *Business Processes* part comes into play, in which matchmaking ensures the correct relationship, followed by negotiations on the terms of the transactions, which if successful leads to a collaboration process. Each of these capabilities is represented at run-time by a separate service running within the cloud.

Key to the establishment of the NIMBLE vision are innovations spanning several areas:

- Supply chain *Platform as a Service* to provide a customized cloud based platform to ease the creation of business-to-business interactions.
- *Front-end* to ease the on-boarding of new companies into the platform and the use of available capabilities in an easy manner without vast IT infrastructure or know-how.
- Extensive use of semantic based technologies to drive the external users' experience by populating an internal registry of semantically enhanced descriptions that are exposed externally via an entity discovery mechanism.
- Security and trust to be maintained at all layers of the platform.
- Flexible communications technologies to connect between NIMBLE entities in the form of publish / subscribe, including notifications.
- Flexible and dynamic data sharing between partners.
- Data storage, processing, fusion, and sharing.

The main outcomes of the project are the architecture specification, and a reference implementation of a customized B2B Platform as a Service, including all detailed internal and external facing components. The project use cases serve for garnering requirements, providing feedback, and evaluating the platform by implementing their applications using platform provided services.

This document focuses on defining a high-level platform architecture. It will form the basis for all NIMBLE components designed and implemented in the various Work Packages (WP), as well as highlighting the interactions between the different WPs and tasks.

The rest of the document is structured as follows. Section 1.1 discusses the main set of requirements that led to the establishment of the current architecture. A high-level view of the NIMBLE architecture and its main components is presented in Section 2. Then, section 3 dives deeper into a more detailed design of the different components and layers within the platform; Section 4 introduces the platform from the point of view of the different stakeholders. Section 5 compares and contrasts the presented architecture with relevant reference architectures. Section 6 introduces the external interfaces of the platform. Section 7 provides an introduction of the planned use cases and maps their planned activities to the architecture presented, validating that the design put forward can indeed cover the envisioned pilots. Finally, Section 8 summarizes and gives arguments for NIMBLE's unique selling proposition, as witnessed by its architecture.

1.1 Major requirements leading to this architecture

NIMBLE aims to provide a wide range of ready-made and easy-to-use B2B services that can be utilised by smaller manufacturing businesses and suppliers when interacting with each other, and with customers.

Therefore, the most prominent requirement leading to the presented architecture is that of consumability, namely making the system easy to interact with for external companies. This requirement drives the way in which internal components are designed and operate, with as much automation, and XaaS (Everything as a Service) philosophy, starting from the provided services, all the way to the design and deployment of the run-time hosting the platform.

The requirement for usability leads immediately to further requirements set around semantics support. The platform intends to semantically enhance in a guided or an automated manner to the extent possible, information concerning the different entities residing in the system, so as to facilitate other entities searching for it. The semantic information will mainly be used by the search and discovery services.

Security and privacy related issues appear prominently as potential sources of concern in such B2B platforms. We intend to take a "security by design" approach rather than an after-the-fact approach and have security related aspects interleaved within the different components to reach a secure and privacy preserving platform.

Since NIMBLE intends to support a large amount of entities and data, scalability of the platform is important. Prime examples to the realization of this requirement in the NIMBLE architecture can be found in the run-time which is realized as a customized PaaS infrastructure, as well as the communication infrastructure employed to connect and provide services to different entities running within the platform, along with a scalable data management component.

A comprehensive description of the NIMBLE requirements can be found in deliverables coming out of WP1. A bird's eye view of requirements mapped to the architecture is shown below:

Table 2: Requirements-to-Architecture mapping

Requirement	**Respective architectural element**
Consumability	Platform front-end
Semantically annotated products/services	Registry
Scalability	Run-time; cloud service bus; data management
Data processing / management	Offline and online data processing
Security	Security components
Notification	Cloud service bus
Federation	Open API
Dynamic data sharing	Business process management / Cloud service bus
IoT data ingestion and processing	IoT and data analytics (advances platform services)
Security: authentication, authorization and roles based ACL	Security components

2 Architecture High Level View

At the heart of the NIMBLE vision lies the existence of many different companies that need to create their own business networks. The companies need to be able to publish their capabilities and look for corresponding capabilities they need. In addition, upon establishment of a business relation, data and notifications need to flow among entities.

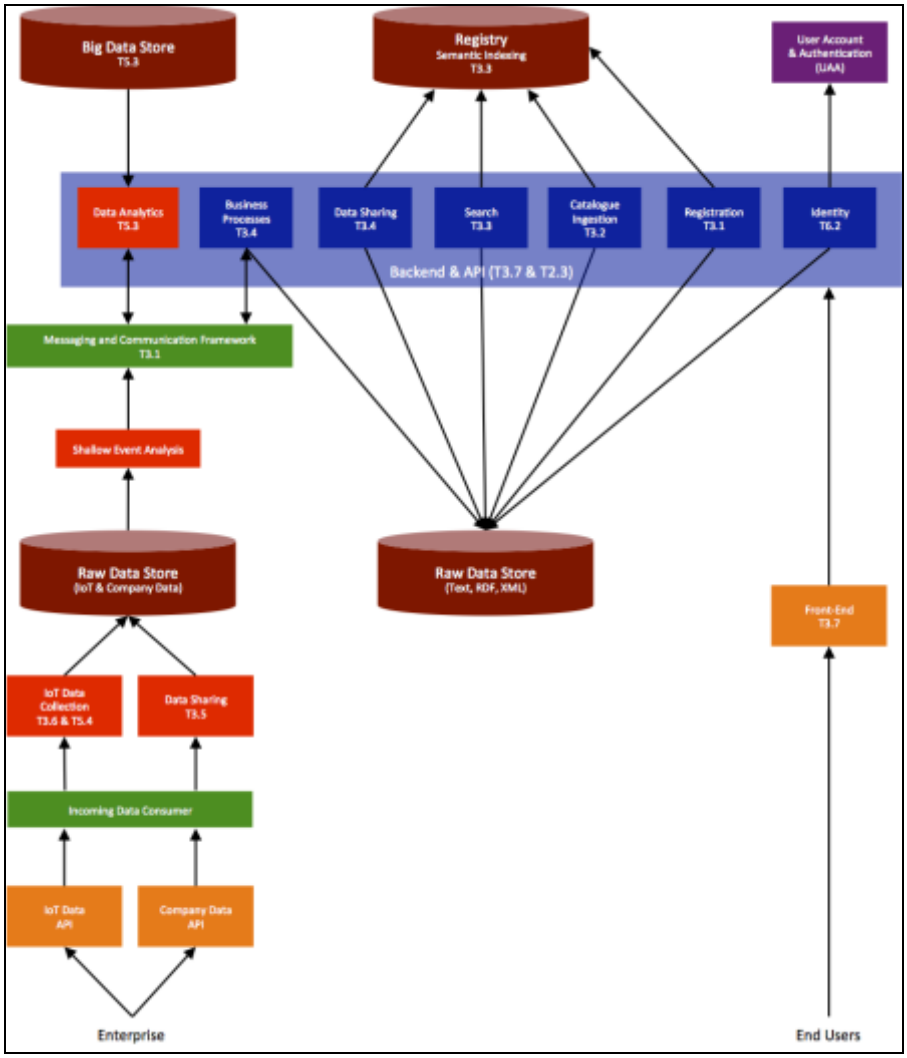

Figure 2: High level view

The high level view shows how major components relate to each other and also indicates how some of the design and development tasks of the work plan are related to the architecture.

Discovery

An internal *registry* holds semantic metadata of capabilities and services provided by the different entities hosted by the platform. This *registry* forms the basis for the *Discovery* capabilities of the platform.

Business Processes

Once entities have found each other via filtering and matchmaking, they can start negotiating integrated business processes, followed by establishing one or more private channels for them to exchange information.

External accessibility

The platform as a whole provides over-arching functionalities that can be used from the outside, via the platform *front-end* and via an *API*. Functionalities include publishing the capabilities of a company and discovering entities with capabilities you are interested in, as well as engaging in multi-party business processes. This open API is intended to enable the formation of ecosystems where a NIMBLE platform is either a central part or just a provider of certain services, to some other ecosystem. The design intention is to support a critical mass of businesses, but not to monopolise them.

Cloud run-time

Cloud run-time components host and run the core NIMBLE services. The run-time components provide direct capabilities such as flexible communication and notification via a cloud service bus, to external entities, and also to the core business services of the platform.

External data

External data fusion and integration is of significant importance to the platform as it forms the basis for the collaborative effort. Such data includes both information about the capabilities provided by different entities, as well as data from IoT devices representing in digital form within the platform the state of the relevant portions of the physical world.

2.1 Federation

The vision is for a federation of platforms for B2B operations, including a set of core services that is shared by all platforms, and optional added-value business functions that can be provided by third parties and may be available in some platforms but not in others. Thus, prospective NIMBLE providers can take the open source infrastructure and bundle it with sectorial, regional or functional added value services to launch a new platform instance in the federation.

The architecture of the platform is designed in this “federation ready” manner. Namely, a cloud based deployment pattern including a set of mandatory core services with a well-defined API. The API will enable both access to internal platform services, as well as a federation API, which will enable separate instances of the federation to collaborate. The presented architecture enables any entity which wishes to create a NIMBLE instance within the federation to deploy all core services within its own instance and complement that with specific personalized services required or requested by a specific sub-set of potential users. Such specializations may take place at the industry level, namely adding specific capabilities for a specific industry, or at a regional level for addressing specific requirements of a specific country or geographical region. The federated platform concept enables the specialization of a fixed set of capabilities for requirements stemming from different sectors or regions.

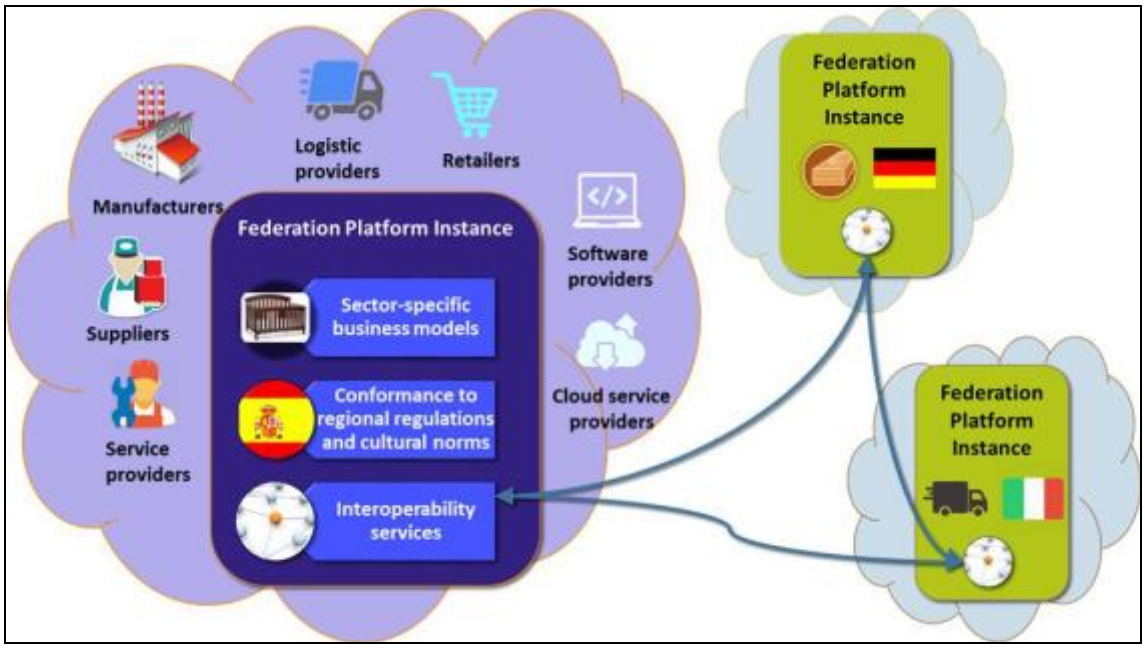

Figure 3: Platform federation

NIMBLE aspires to a federated yet interoperable eco-system of medium-sized platforms that provide B2B connectivity. Such a common, yet federated infrastructure opens the door for multiple platform providers, with a diverse set of platform instances that still can collaborate.

2.2 B2B Collaboration

As elaborated in detail in section 3.6 (Data management and analytics), one way of accessing data about a business entity is to query its data sharing services registered to the NIMBLE platform. This way of data access is convenient when the collaborating entities do not have a need for a complicated message exchange scheme for collaborating but gathering information by issuing a query like order/product id, product type, production data for a product given its id, etc. would be sufficient. On the other hand, NIMBLE also offers collaboration via business processes through which custom and more complex information exchange flows can be realized. Unlike the former method enabling a bilateral and one-way information transmission between two trading partners, business processes might involve more than two entities where information can be distributed to some or all participants simultaneously. Furthermore, business processes allow involvement of human actors who intervene in the data sharing e.g. confirmation of a delivery plan before broadcasting it to the customer and logistics service provider.

Setting up a B2B collaboration infrastructure among business entities regardless of their size, type and role in the supply chain is the main motivation behind developing a platform like NIMBLE. Figure 4 elaborates a generic way for the realization of B2B collaboration starting from the conceptualization of business models established among different types of participants. In this context, a *business model* is a conceptual agreement that is mutually agreed by the participating entities either through some formal signed agreements associated with well-defined service levels or through informal partnerships. In any case, business models are realized by exchanging information among the participants.

A set of digital information exchanges ordered as a particular sequence driven by a particular type of collaboration is represented via *business process*. In the NIMBLE context, business

processes are composed of *collaboration templates*, containing meaningfully grouped transactions. For example, assuming that a business process involves information exchange about ordering and subsequent delivery operations. In this example, we would have two collaboration templates, including one or more transactions in each, about the ordering and delivery phases respectively.

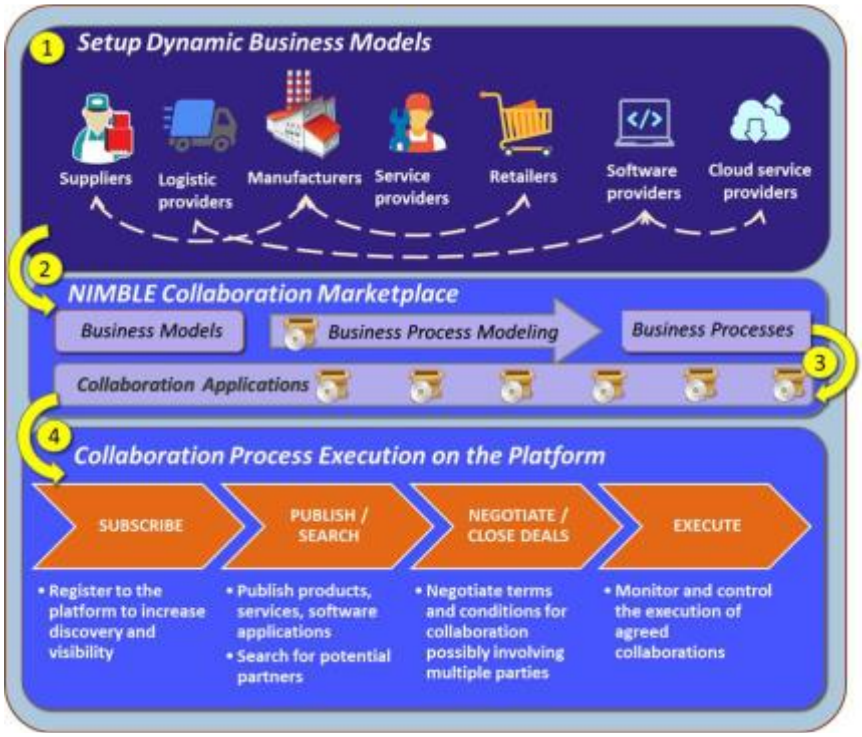

Figure 4: Realization of B2B collaboration among NIMBLE users

In the NIMBLE context, business processes in collaboration with Communication Infrastructure (c.f. section 3.8) can be seen as a backbone to which NIMBLE cloud services are connected. Business processes, on the one hand, gather data from business entities via granular data integration modules (see Section 3.6 Data management and analytics); on the other hand, transmit the data to services closer to the end users e.g. a visualization service for transportation conditions or an alert service for generating notifications to inform the entities involved in the business process, etc.

Before collaborating on NIMBLE, business entities need to register on the platform by providing metadata about the entity itself such as trading preferences, target sectors, certifications, etc. Being the first step to be visible on NIMBLE, such information makes establishing the initial agreement with new partners easier. The next step is to publish capabilities in the form of products manufactured, materials supplied and services provided. These are the major resources describing the entity on NIMBLE and make it discoverable for collaboration. Even though the common way of collaboration is envisioned as identification of new partners based on these product/service features, it is also possible for entities to collaborate with partners they already know and have come to agreements with, previously. In either case, entities might negotiate on the trading conditions as well as data to be shared based on the previous agreements or type of collaboration.

B2B collaboration through business processes is realized in three steps: design, configuration and execution. The design and configuration steps are enabled by the Business Process Design Service, which is one of the core services of NIMBLE. This is a cloud based service with user interfaces enabling specification of a concrete business process by determining the type of

entities to be involved in the process e.g. manufacturer and supplier; designing the information exchange flow among the entities and assigning the entities to data sender and recipient roles. Afterwards, the process is instantiated and executed by an open source business process engine like Camunda (https://camunda.org/) or Activiti (https://www.activiti.org/).

2.3 Micro-Services Approach

Microservices is an architecture form and methodology for breaking up a large complex system into small units, each fulfilling a small a unique well-defined task, in an independent manner. Each such microservice can be deployed separately and interacts with its peer microservices using standard communication protocols, under the provision of an application orchestrator. Such an approach eases the task of long term maintenance and evolution of a large system, when compared to a monolithic system. In addition, internal changes within a microservice do not disturb any other system component as long as the external contract of the microservice is adhered to. Moreover, each component can be implemented in a different programming language, using different middleware.

Such an architecture is distributed in nature thus there needs to be a way for one microservice to find another microservice it wishes to invoke, or for the orchestrating application to locate a specific microservice. For that purpose, a service discovery component is deployed. Such a component serves as a central registry of available microservices, responding to queries about the location of a certain microservice. In actual deployments, this component can be serviced by Netflix Eureka[1], amalgam8[2], Consul[3] or native cloud components such as Cloud Foundry's GoRouter[4], which does not respond to queries but rather immediately invokes the desired microservice by name.

[1] https://github.com/Netflix/eureka
[2] https://www.amalgam8.io/
[3] https://www.consul.io/
[4] https://docs.cloudfoundry.org/concepts/architecture/router.html

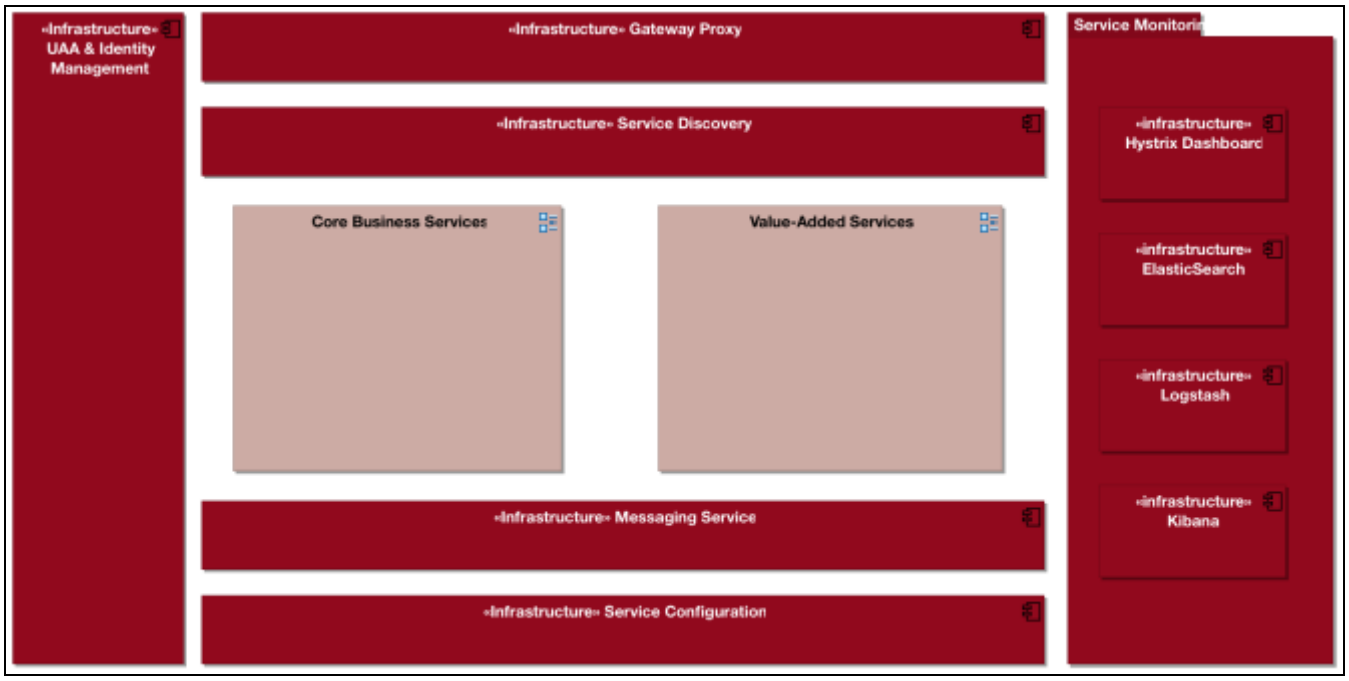

Figure 5: Components of a Microservices architecture

In addition, a gateway to the microservices is usually deployed, which acts as the public entry point to the Microservice, providing routing, filtering, and load balancing capabilities. In actual deployments, this component can be serviced by Netflix Zuul[5], or native cloud components such as Cloud Foundry's HAProxy and GoRouter.

In a microservice environment, the state of different components needs to be checked to ensure that services are up and ready to respond. As the system grows, more microservices are deployed with more instances per each microservice, and the probability for a component to be unresponsive goes up. The first stage is to perform health monitoring, which will be done via the Hystrix Library[6] or native cloud components such as Cloud Foundry's health check[7] and having the cloud infrastructure restart a failed application. Moreover, these components often interact back with the service discovery component to report instances, which are not responsive.

Communication among applications will be performed via direct REST calls or through a messaging service or a cloud bus[8]. In case of RESTful communication Netflix Feign[9] is used as technology in combination with Netflix Ribbon[10] as client side load balancer. In order to trace requests between microservices, Spring Cloud Sleuth[11] is used, which assigns each request an identifier and associates them into logical groups. Logging will be achieved using the ELK stack[12] (**E**lasticSearch, **L**ogstash, **K**ibana) or through the cloud's native logging tools. A Service Configuration can be setup via a combination of Spring Cloud Config[13], and a central Git repository.

[5] https://github.com/Netflix/zuul
[6] https://github.com/Netflix/Hystrix
[7] https://docs.cloudfoundry.org/devguide/deploy-apps/healthchecks.html
[8] https://cloud.spring.io/spring-cloud-bus/
[9] https://github.com/OpenFeign/feign
[10] https://github.com/Netflix/ribbon
[11] https://cloud.spring.io/spring-cloud-sleuth/
[12] https://www.elastic.co/webinars/introduction-elk-stack
[13] https://cloud.spring.io/spring-cloud-config/spring-cloud-config.html

Finally, security and access control are the responsibility of the User Account and Authentication (UAA) & Identity Management component which administrates identities on the platform. Cloud Foundry's UAA[14] or Keycloak[15] can be used as authentication server for the OAuth2 and OpenID Connect standards.

2.3.1 Twelve-Factor Applications

The platform will be composed of multiple microservices and will deliver services to users according to the Software as a Service (SaaS)[16] paradigm. A very recent methodology for building SaaS applications is called *twelve-factor*[17], which defines twelve guidelines for building and running applications in the cloud. Due to their affinity to cloud environments, they are often called *cloud-native* applications. In the following paragraphs, each factor will shortly be introduced and its respective implications for the NIMBLE platform stated.

I. Codebase

There must be a strict one-to-one relationship between codebase (e.g. code repository on Git) and the application. There should not be multiple deployments of the same app, but a deployment consists of a running instance of an application.

NIMBLE relevant implications: Codebases of each microservice are strictly separated. This is achieved by dedicating a distinct repository to each microservice.

II. Dependencies

Twelve-factor applications should never implicitly rely on system-wide packages or functions. All dependencies must be declared completely inside the scope of the application (dependency declaration). Additionally, no implicit dependency should leak from the surrounding system (dependency isolation).

NIMBLE relevant implications: Most microservices are built using Java and Maven. Each dependency must therefore be explicitly stated in the building manifest file (pom.xml). For dependency isolation, each microservice runs either inside an isolated Docker container or a Cloud Foundry build pack.

III. Configuration

A strict separation between configuration and code is necessary to deploy applications in the cloud, because configuration varies across deployments, whereas code does not. Relevant configuration includes credentials to third-party services and pre-deployment values (e.g. hostname).

NIMBLE relevant implications: Spring Boot applications externalise their configuration using YAML files, environment variables or a central configuration server. Implemented microservices will heavily use this paradigm in order to separate configurations from code.

IV. Backing Services

A backing service is any service the microservice uses over the network during its normal operation (e.g. data stores or messaging systems). Code from microservices should make no distinction between local and third party services. Respective configurations must be stored externally in the configuration.

[14] https://github.com/cloudfoundry/uaa
[15] http://www.keycloak.org/
[16] https://en.wikipedia.org/wiki/Software_as_a_service
[17] https://12factor.net/

NIMBLE relevant implications: Backing services (e.g. Cloud Foundry UAA or Apache Marmotta) are configured externally. This makes it possible to easily exchange backing services during the release phase.

V. Build, release, run

Twelve-factor applications strictly separate between build, release, and run stages.

NIMBLE relevant implications: The artefact of the build stage is a Java executable with a default configuration. In the release stage deployment-specific configuration (e.g. Docker or Cloud Foundry deployment) is added to the application.

VI. Processes

Similar to traditional processes, twelve-factor application must be stateless and share-nothing. Any persistent data is stored using external backing services (e.g. data store).

NIMBLE relevant implications: NIMBLE's internal implementation will strictly separate between stateless applications (i.e. microservices) and state-full services (e.g. data store as backing service).

VII. Port binding

Twelve-factor compliant applications are completely self-contained and do not rely on any injection during runtime. Services are provided to externals via a certain port bound to the application.

NIMBLE relevant implications: Microservices are built using Spring Boot and therefore self-contained Java executables (i.e. the webserver resides in the Jar file).

VIII. Concurrency

In Twelve-factor applications processes are first class citizens, similar to the Unix process model. Therefore, a microservice should easily be scaled horizontally and the workload distributed amongst multiple instances of the same microservice.

NIMBLE relevant implications: Microservices are built using the Spring Cloud framework, which includes a client-side load balancer (i.e. Ribbon). This makes it possible to distribute work amongst multiple instances.

IX. Disposability

Twelve-factor applications must be disposable. Therefore, they can easily be started and stopped without much overhead. This allows us to scale applications up or down, quickly.

NIMBLE relevant implications: Microservices will be optimised for quick start-up and graceful shutdown. Some technologies perform pre-caching of important data, which leads to additional start-up time. In NIMBLE we will disable such optimisations.

X. Development / production parity

Twelve-factor applications are designed for continuous deployment. Therefore, the gap between development and production environments should always be as small as possible. Especially in the area of backing services the development / production parity is vital.

NIMBLE relevant implications: Tools from Spring Cloud (e.g. Spring Cloud Connector[18]) will be utilised in order to keep the gap small. In addition, technologies used for production are going to be utilised during development (e.g. PCF Dev[19] or Docker).

XI. Logs

Routing or storage of log output should never be a concern of a twelve-factor application. It should rather provide a continuous stream of logging information, which is retrieved by separate backing services within the system.

NIMBLE relevant implications: Each microservice uses either an instance of the ELK stack[20] as backing service for log aggregation or relies on the underlying log management of Cloud Foundry. Microservices will never store log files in the local file system.

XII. Operational Task

Each recurring task should be automated and the resulting code added to the respective codebase of the application. Those operational tasks should also be used within release cycles.

NIMBLE relevant implications: All relevant files (e.g. Dockerfiles, Docker-compose file, Cloud Foundry manifest files) will be stored in the code base of the respective microservice. In addition, scripts for database migration and general setups will be stored in the same code base as well.

3 Platform Core Components

Platform core components represent the main aspects which are required to take a "standard" cloud environment and make it NIMBLE ready. They provide the core services which are the heart of a NIMBLE instance. Figure 6 shows the composition of NIMBLE core components depicted as individual micro-services. In the figure, micro-services with similar contexts are grouped into component groups. Details about them are provided in the subsequent subsections of Section 3. Note that these details present the envisioned functional capabilities of individual components. A concrete API to utilize those functional capabilities will be given in the deliverable "*D2.3 - Design of an Open API for the NIMBLE Platform*".

[18] http://cloud.spring.io/spring-cloud-connectors/
[19] http://cloud.spring.io/spring-cloud-connectors/
[20] https://www.elastic.co/webinars/introduction-elk-stack

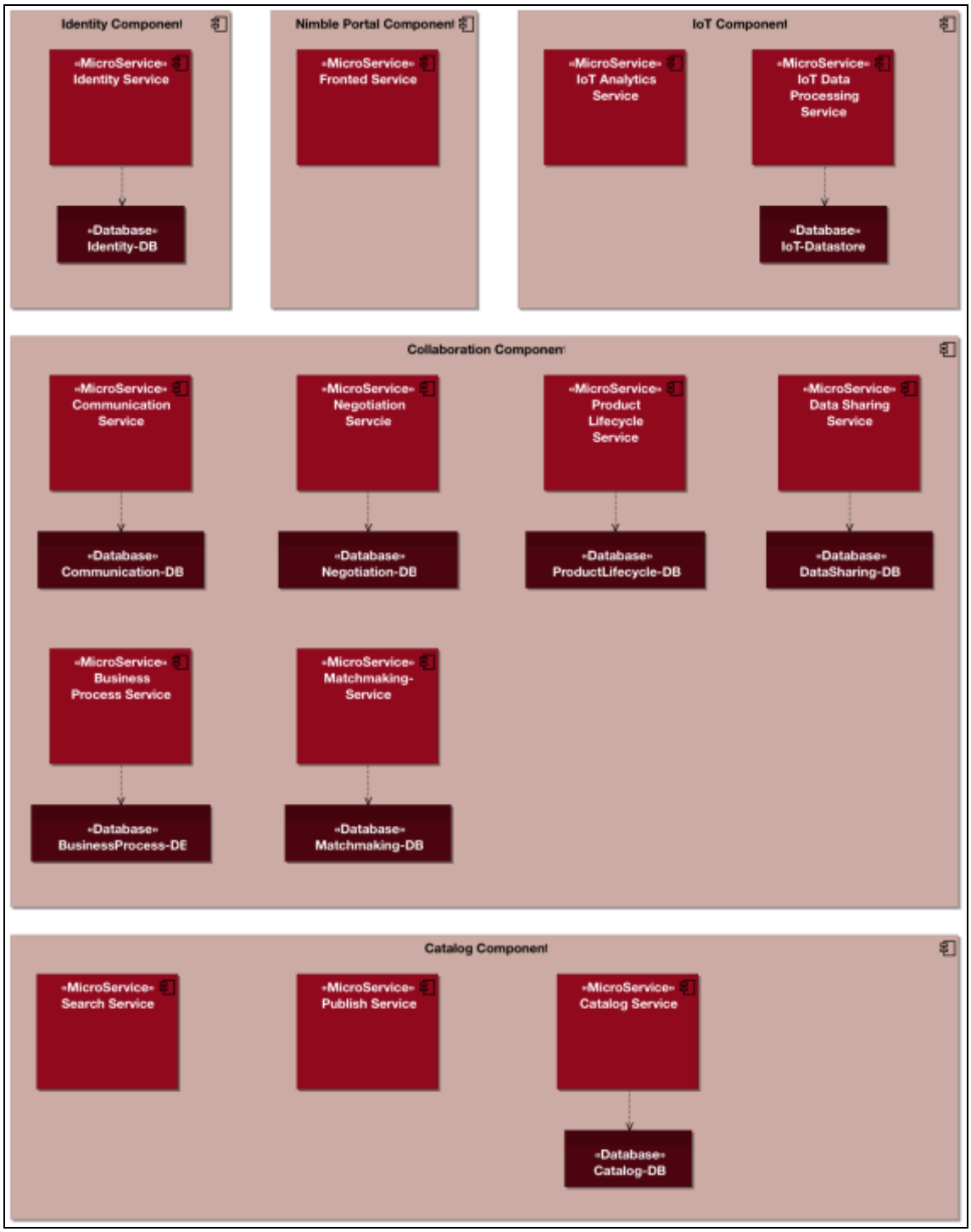

Figure 6: Core components - a microservices view

3.1 Run-time environment

The NIMBLE platform run-time will host all entities needed for the complete operation of the platform, including NIMBLE core and supporting services. In addition, it shall connect all the platform supplied middleware services such as the discovery and business processes related

services. Hosting these entities is comprised also of enabling the creation, absorption, and storage of data and metadata created and associated with various kinds of entities.

The NIMBLE platform run-time will consist of a specially customized PaaS cloud, geared towards B2B scenarios. Thus, the daily operations will be supported by a cloud environment automating many of the tasks of running such a platform in a scalable and secure manner. This B2B PaaS will coordinate and provide interfaces and hooks to additional system components, such as the front-end, to complete the cycle of aiding the users with fulfilling their tasks.

In general, NIMBLE follows a strict separation between applications and services. Applications are stateless modules, which implement NIMBLE internal business logic. On the other hand, services are used to store certain states and data in a persistent manner. Therefore, applications can easily be scaled horizontally without potential data inconsistencies arising.

An internal cloud service bus for communication is integrated within the cloud run-time environment to provide internal communication and notifications among NIMBLE components.

3.1.1 PaaS: the platform Run-Time

The run-time infrastructure is supposed to provide the basic mechanisms to deploy, host, and manage the NIMBLE platform internal components.

The NIMBLE requirements from its run-time have many parallels with current Cloud Platform as a Service (PaaS) frameworks, and can be viewed as an extension of B2B supply chain and logistics environments. In general, the mutual core requirements are:

- Reduce the complexity of writing and deploying web applications written in many languages
- Provide generic middleware services, and enable applications to easily bind with them.
- Be IaaS (Infrastructure as a Service) agnostic
- Support automated elasticity to cope with production environments

Some examples for PaaS frameworks are Cloud Foundry, Heroku, Amazon Web Services, Microsoft Azure, and IBM BlueMix. For NIMBLE, being an open and federated platform, the goal is to have the core services deployable on more than a single underlying PaaS, such that providers of a new NIMBLE instance will have the capability to deploy it on the infrastructure of their choice.

NIMBLE, being a B2B PaaS shall add the B2B required capabilities on top of an existing PaaS framework. Currently we are experimenting with two main platforms, namely Cloud Foundry and IBM BlueMix.

As an example, the Cloud Foundry (CF) architecture, shown in Figure 7, is conceptually simple. Every external entity that communicates with the platform or the entities hosted in it needs to go through a routing layer. This router is in charge of maintaining the mapping between the web address provided to external users and the real physical location in which the desired application resides. A web user will be directed to a silo of web applications, whereas a web developer that publishes or manages an application will talk to the cloud controller. Web applications may be bound to platform services, like a database, a messaging service, etc. Thus, every application deployed in CF can be divided into two components: an "application" and a set of "services". Applications are usually deployed on top of a container. The virtual machines (VMs) in which the applications run are stateless, and are managed by CF itself. Thus, an application that needs to save data uses a data storage service like a database, NoSQL store, etc. Services are either hosted on state-full VMs, or physical hardware, and can be managed by CF itself or by a third

party. CF comes with a small set of built-in services (e.g. MySQL, MongoDB), but it is rather easy to extend the set of services and add customized services.

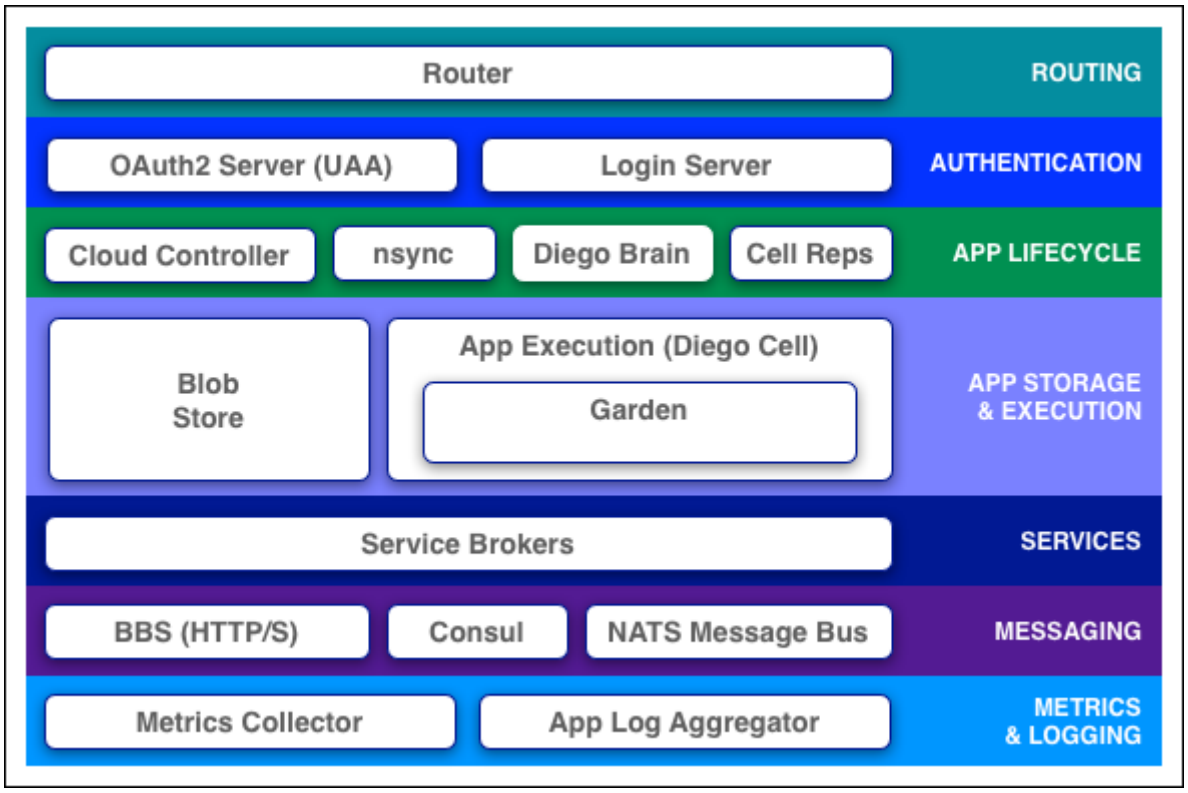

Figure 7: Cloud Foundry architectural overview

Accordingly, every NIMBLE internal component should be decomposed into parts that can be hosted as applications and others that should be hosted as services. A schematic view of the architecture is presented in Figure 2. NIMBLE core services are hosted as cloud applications and enjoy the cloud infrastructure for connecting to back-end services, elasticity, fault tolerance, etc. The NIMBLE platform augments the generic cloud installation with several specific capabilities such as data management, discovery, and notifications services.

NIMBLE Components Deployment

NIMBLE platform deployment starts from the cloud infrastructure, including base services such as security. Once the bare cloud infrastructure is in place the necessary services need to be deployed, namely for storage, communication, etc. Certain services may accompany the vanilla cloud deployment, otherwise they need to be deployed separately, such as discovery, gateway, monitoring, etc. Next, the cloud service bus can be deployed, as an application that potentially uses cloud provided messaging services. At the last stage, the NIMBLE applications are deployed, either with well-known URLs, or they get to register with a service discovery component such that the connection between different applications can be established.

3.2 Business Registry

The registry is the component to store the information that makes companies discoverable on NIMBLE. The information to be stored is grouped under two main categories: 1) Basic descriptive information about business entities, relationships between them, commercial

characteristics and trade preferences; 2) Products and services provided by business entities along with their characteristic features.

3.2.1 Semantic modelling

The semantic modelling task develops data models for representation of several types of tangible and intangible resources. Based on the data models, resources from different partners in various formats can be semantically annotated and published, so that they can be discovered later, by other partners using semantic queries. The data models are subject of separate deliverables in NIMBLE and there are at least two levels of abstraction to be considered for these models: first, the horizontal semantics that establish what an enterprise is, what kinds of business processes NIMBLE supports, etc. Second, the more specific semantics that apply e.g. in the field of textile manufacturing or furniture building. At present, some of the focus is on the topics of the first four use cases, but the semantic models need to be easily extended or tailored to other use cases.

During the semantic modelling operation, a typical ontology modelling methodology such as METHONTOLOGY and NeOn Methodology will be referenced. However, these methodologies could be possibly adapted, so that we can have at the end usable data models in the NIMBLE platform. The requirements for such data models come from WP1 which is in charge of requirements gathering form the use cases.

Semantic data model of the Registry has a number of main modules composed of fine-grained data elements. Figure 8 provides a high-level view of the semantic model of the Registry. First, the Business Entity module contains data elements to represent commercial characteristics (such as trading preferences, certifications, etc.) in addition to basic information of companies. The Business Entity module is linked to the Resource Catalog module. Main elements of both modules are represented using the Universal Business Language (UBL) standard data model. The latter module is used to represent the catalogues of products / services offered by business entities along with their detailed features. As a core capability, the Resource Catalogue module is further linked with the eClass taxonomy[21] for semantic annotation of products / services with specific features related to them.

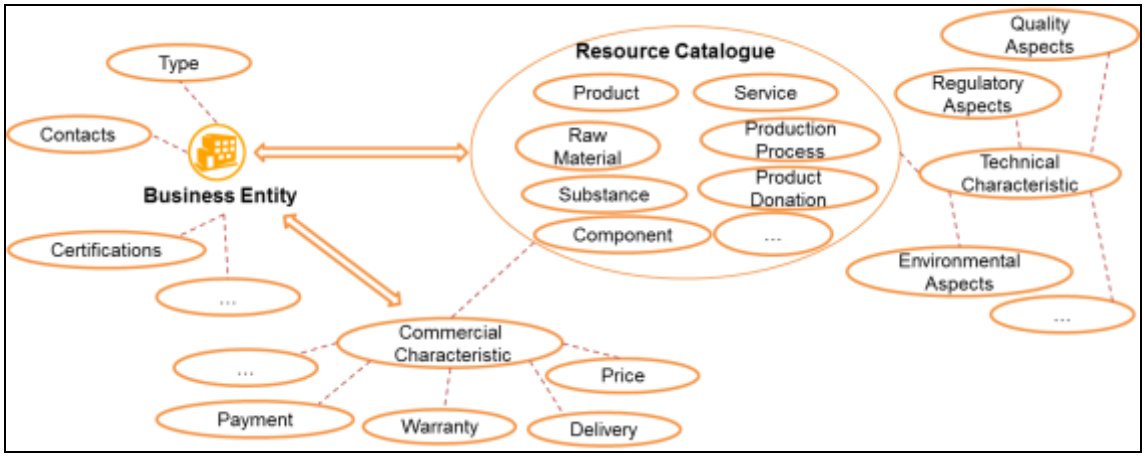

Figure 8 Modules of Registry's semantic data model

Furthermore, the customization of the NIMBLE core can be extended with additional taxonomies for proper semantic annotation of products / services with sector-specific

[21] http://www.eclasscontent.com/index.php?language=en

characteristics. As the use cases cover many different topics, various existing and widely accepted ontology resources – as well as non-ontology resources – will be checked for reusability. For example, for the use case in the textile industry, the MODA-ML ontology will be adapted and reused. Ultimately, many different ontology modules would be integrated to create semantic models meeting the requirements of use cases.

Details of the semantic modelling for the Business Registry Component is given in D2.2.

3.2.2 Company & product registry

This component provides multi-modal persistence options to address different querying requirements. On the one hand, it provides free-text indexing where all metadata for a business is indexed as key-value pairs. On the other hand, it provides persistence of relational data in a structured way inside a relational database and/or a semantic store (i.e. a triple store).

Free-text indexing is suitable for the execution of keyword-based search for getting an initial list of products or business entities as this type of search can be performed on all data fields simultaneously. Available free-text indexing tools like Apache Solr[22], Elastic Search[23] provide advanced features that are very relevant for the services that NIMBLE should offer to users. Among these features are:

- A scoring mechanism to evaluate the relevancy of indexed items with the query term
- A search mechanism that generates the related features with the search results. Such a capability enables filtering the initial results further according to particular features.
- A dynamic indexing mechanism that enables associating features for products that have not been considered before. This would allow indexing new sector-specific features without extra effort.
- Field boosting to indicate the importance of a field either at query time or at indexing time. This allows users to specify the features of interest that have higher priority and thus, find relevant items more quickly.

In addition, it will also be possible to query the persisted information in a structured way that would allow custom queries that may be needed by NIMBLE's internal components or user/sector-specific applications.

3.2.3 Catalogue ingestion

In NIMBLE, we develop a generic catalogue ingestion mechanism for business entities that do not use a specific representation format for their products / services. On the other hand, we also aim to support entities performing within a particular sector using sector-specific representation formats / data models such as MODA-ML[24] in the textile sector, or ISO 10303[25] in the child furniture sector.

Looking from the querying perspective, however, NIMBLE should provide a generic discovery mechanism regardless of a specific sector. Such a capability, requires transformation of any ingested item (i.e. product / service) into a common representation to a certain extent. As it would be infeasible, probably impossible, to align each data model with other models

[22] http://lucene.apache.org/solr/
[23] https://www.elastic.co/
[24] http://www.moda-ml.org/
[25] https://www.iso.org/standard/42340.html

completely, we introduce a simple and extensible indexing mechanism to deal with this heterogeneity problem. We see each item as a set of key-value pairs to be persisted in a free-text indexing tool as introduced in the previous section. Therefore, we would need individual adapters extracting such pairs from sector-specific representations to be stored in the registry in a unified way. Nevertheless, it is still possible to keep the original representation of the ingested item that could be utilized by interested users / applications. Figure 9 provides an overview of the ingestion procedure along with the technical components underneath. *Feature Extractors* are used to extract the product-specific features from an external taxonomy like eClass and embed them into the main item possibly using a dereferenceable link to the root category to which the product-specific features are associated. On the other hand, *Storage Configurations* provide ways to access the original product description.

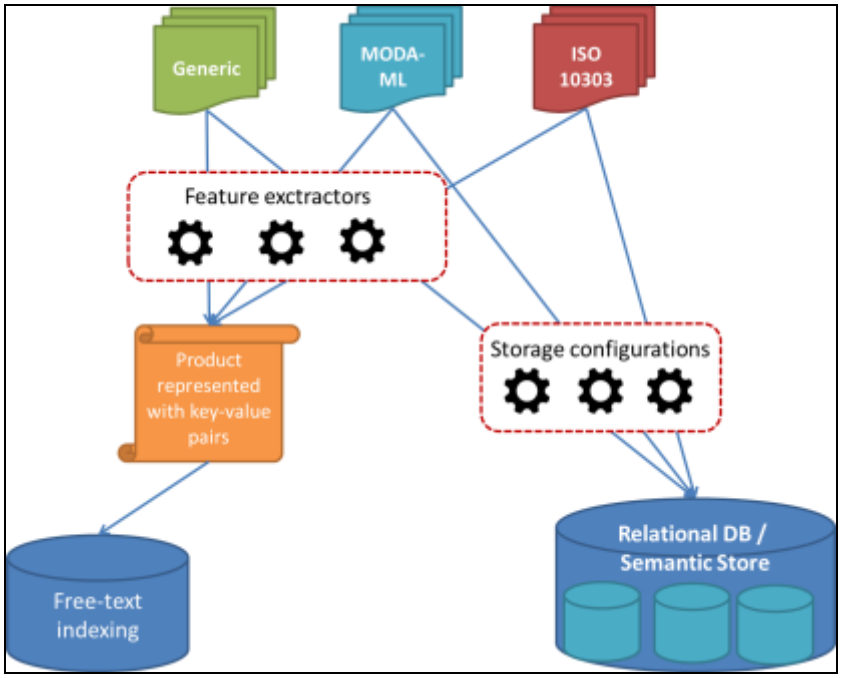

Figure 9 Catalogue ingestion procedure

In addition to tradable products and services that can be published into the product registry, IT-driven data sharing capabilities of entities will also be stored. These capabilities will be extracted automatically from the data sharing components associated with entities.

NIMBLE uses the eClass release 10.0, which is a cross-sector product data standard for classification and clear description of products and services as well as declaration of product / service specific features. NIMBLE offers publishing products and services in two ways. First, after selecting a product category from the eClass hierarchy, users can publish products either one-by-one by specifying values for the features of the selected category for each product. Or, users can download a template generated again based on the selected category and provide values for the features for as many items as desired. Both types of publishing capability will be available via the open API to be provided. Users will also be able to publish sector-specific products if there exists a registered feature extraction adapter for the custom data model.

The ingestion component in collaboration with the company/product registry component will also deal with product variations by enabling maintaining of multiple values for individual product features e.g. colour range for bathroom tiles, sizes of a baby cot, etc.

3.3 Service / Partner Discovery

The service discovery component deals with search capabilities for catalogues. The different distributed catalogues/registries contain searchable service descriptions.

The different search modalities of users require different use case specific workflows. The search methods will cover keyword-based search, faceted search and semantic search. The main goal is to provide semantic search capabilities through intuitive user interfaces for the various search functions.

All three kinds of search functions will apply ontologies and corresponding query languages like SPARQL to perform the search functionality. Hereby, each kind of search will provide its common look and feel and is going to hide the underlying ontological complexity. For example, a user will not notice in a keyword based search that his keywords will be mapped to concepts and axioms of similarity internally.

The most relevant data sources will be provided from the catalogue services. To achieve a robust search service, a configurable mapping between the data schema of the catalogue and the ontology driven search approach is going to ensure the interoperability.

The 'service discovery' service will be provided as a micro service. Spring will be used to provide the micro service architecture. Therefore, it will offer a REST interface in which search queries could be invoked. Each search query will include at least the user specific input data as well as the applied search function. Based on both kinds of information, a corresponding SPARQL query including assumptions about missing information will be generated and used to extract the necessary information from the catalogue service. The assumptions are necessary to fill the gaps which arise with not so formal search functions (like keyword based search). The results will be provided as JSON or XML and will be used to allow an explorative search approach.

3.4 Matchmaking

Unlike a standalone service that is utilized by users directly, matchmaking is a set of supporting functions embedded within the main search and discovery features of NIMBLE. These embedded functions assist users in finding suitable services, products and other business entities and discover the most relevant results faster. These supporting functions are context-aware and take into account any trade preferences of the business entity that is using NIMBLE in a particular activity.

Matchmaking is performed based on different criteria in different contexts. For example, when a keyword-based query is initiated for getting a particular product; the original query can be enriched with additional query parameters so that the results include only the products supplied by entities that satisfy the trading criteria required by the current entity. Another way for this can be post-processing the search results so that they are ordered according to supplier firms' trading characteristics.

Several criteria can be used for matchmaking such as specific supply chain need (speed, reliability, flexibility), geographic alignment, expected capabilities from partners (volume, financing, market access, etc.), supply chain segment (raw material provider, manufacturers with specific capabilities, etc.).

3.5 Business process and negotiations

NIMBLE aims for users to be able to utilize business process services to exchange information with several partners easily; a task that cannot be realized by simple querying of a data sharing service as explained in the next section. However, a set of activities should be performed before utilizing these services. First, the skeleton of the business process should be constructed. This skeleton specifies the entities involved, types of messages to be exchanged among these entities along with the sequence of transactions. Figure 10 shows an example skeleton involving three types of entities: a manufacturer, a supplier and a logistics service provider where the manufacturer entity receives production information about a specific order and sends shipment schedule to inform the supplier as well as the transport service provider.

Second, each participant entity in addition to the one initiating the business process should be visible on NIMBLE. In other words, each entity should be registered to NIMBLE with basic company information at least. An entity aiming to initiate a business process has two main ways of discovering other entities on NIMBLE: 1) The target entity is known beforehand as it is an existing trading partner with previous collaborations. 2) The target entity is discovered via a search process initiated with a keyword query and selected amongst the initial list of entities based on specific features of the products / services it provides. In addition to domain-specific characteristics of products / services e.g. yarn manufacturing, entities are filtered according to their capabilities of data sharing e.g. ability to provide real-time data about the yarn production processes. It is also possible to filter entities according to any other information related to that entity such as role in the supply chain, performance indicators in terms of cost, quality speed, geographic location, trading preferences, etc.

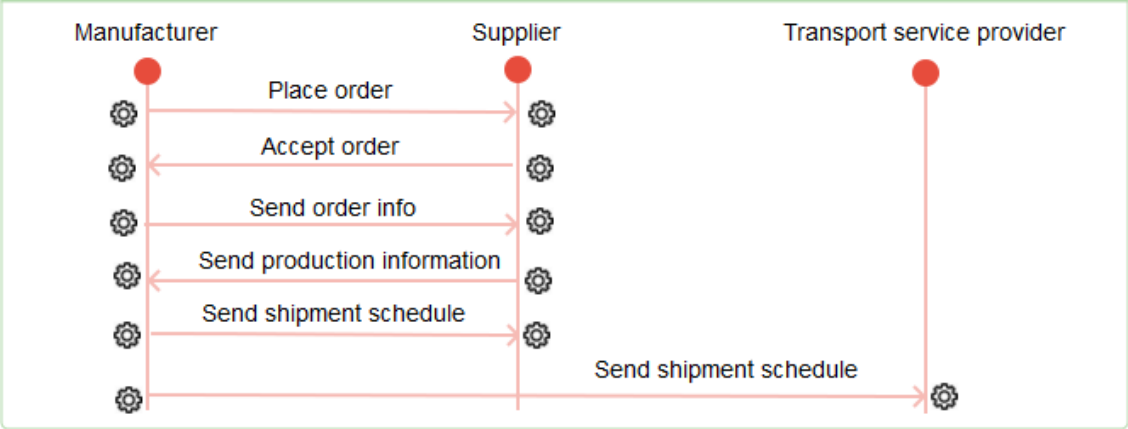

Figure 10 An example business process skeleton as a sequence of information exchanges

Each arrow in the figure means a transaction through which a particular type of message is sent to the recipient. The next activity is the technical configuration of these transactions on one side to gather / generate data to be shared with the target entity (i.e. the beginning node of the arrow), on the other side to receive and process the incoming data (i.e. the ending node). Each data sending node makes use of a data sharing service of the relevant entity based on the information to be sent. Similarly, each data receiving node should be configured as required by the task performed in that node. In this respect, NIMBLE will make use of Business Process Model and Notation (BPMN) 2.0[26] for business process representation. BPMN provides different types of tasks that can be performed in the nodes e.g. *Service Task* for invoking a

[26] http://www.omg.org/spec/BPMN/2.0/

service executing a custom business logic, *User Task* for involving a human actor for decision making, *Receive Task* for waiting a particular type of message, etc. Business Process Design Service allows configuration of nodes with BPMN constructs via its user interfaces which saves users from learning the details of BPMN specification and the technical details to utilize it.

As introduced earlier, business processes can be divided into templates that group logically related information exchanges together. NIMBLE will provide built-in templates and business processes for generic supply chain operations e.g. ordering, fulfillment, etc. We will benefit from Universal Business Language (UBL)[27] for these built-in templates where relevant. See the business processes defined by UBL at [28]. We will keep the granularity of the templates low so that users would be able to deal with smaller parts of complex business processes and perform integrations progressively. In addition to sector-independent processes introduced by UBL, NIMBLE also will support sector-specific processes e.g. defined by MODA-ML for the textile industry.

Once all nodes of a business process are configured, the process becomes ready to be executed. For execution of business processes, NIMBLE will make use of existing open source business process management tools implementing BPMN 2.0.

3.6 Data management and analytics

The data management component deals with data ingestion, storage, and potentially some level of processing for shallow data analysis. Moreover, data based notifications can be supported by connecting the data pipe to the cloud services bus which serves as the messaging pipe.

First and foremost, this component is in charge of ingesting heterogeneous data types into the platform. Based on the requirements of the entity connecting a data source, data flowing into the platform can take several routes. The simplest one is for incoming data to be stored for future use, including offline analytics. Furthermore, data can be passed through a real-time pipe for performing transformations on the data including filtering, potentially targeting notification, and finally storage of transformed data items.

Of specific interest is data generated from IoT devices. That data will be brought into the platform for enabling the interaction of IoT data with other kinds of data, processes and applications running within the platform. For example, that will enable tying in real-time information from the real-world to trigger business processes steps among different entities.

IoT data poses numerous challenges to traditional data storage and processing systems due to the unique characteristics exhibited by this kind of data, such as scale and heterogeneity. Proper solutions need to address data collection and fusion, in addition to enabling actionable insights out of the data streams and data oceans. The Data management component must address data warehouse issues as well as real-time aspects.

Ultimately the data management component will be provided as-a-service to the rest of the NIMBLE components internal and external to the platform. The Big Data Toolset will be made available through the PaaS model, enabling developers to deploy, host, and manage applications that can handle data coming from IoT-enabled devices. Thus, any NIMBLE component will be able to invoke data management procedures such as storage and analytics. NIMBLE will make use as much as possible of existing open source tools for portions of the framework, such as ElasticSearch or CouchBase for storage and indexing and Apache Spark for real-time processing.

[27] http://docs.oasis-open.org/ubl/os-UBL-2.1/UBL-2.1.html
[28] http://docs.oasis-open.org/ubl/os-UBL-2.1/UBL-2.1.html#S-OVERVIEW-OF-BUSINESS-PROCESSES

Overall these data management services will be elaborated, further designed and implemented in later stages of the project, and more detailed information will be provided in the relevant deliverables. In particular enclosed are some initial design goals for two of the main corresponding components:

Product data sharing throughout the supply chain

Data can be provided in two ways:

1. Store data over the NIMBLE platform: Saving data and metadata that can be indexed via ElasticSearch, to be easily found and retrieved; the metadata must be provided along with the data.
2. Register data retrieval services in the platform, with their metadata to find them. These services are run by the registering company and are called every time in which the data is needed. These data retrieval services will be stored registered in the Netflix Eureka Service Registry and Repository for later discovery. When a service is returned by Eureka, it will be called to retrieve data and return them after a mapping to a canonical format (RDF is the preferred one).

In addition, IoT data could be stored in the same way over the platform or provided via real time like services.

Lifecycle data management

According to the use cases, the lifecycle data could be stored on the NIMBLE platform or provided by third party services that must be called when a set of data is required. The data and services must be accurately described with metadata to quickly find, retrieve and update them. The lifecycle data management could store or provide data in various steps and the users could retrieve them upon need.

The NIMBLE platform will interact with the lifecycle data at different times and in different ways, because the platform could store data about a specific production step, then retrieve them to contribute to another step. This kind of interaction is heavy and the use of ElasticSearch and Eureka will contribute to improving the performance.

A product avatar is contemplated, in which all the information concerning a specific product is concentrated and easily parsed to provide a 360 degrees view of the product. This will be described in D3.6.

3.6.1 Information quality

The topic of information quality (IQ) has been intensely discussed for at least two decades and several definitions for ‘information quality’ exist. The following description of information quality is published in [9]. From a general perspective, the quality of information can be defined as the degree that the characteristics of specific information meet the requirements of the information user (derived from ISO 9000:2005 [10]). Based on this understanding, Rohweder et al. propose a framework for information quality that is an extension of the work conducted by Wang and Strong [11] – it contains 15 information quality dimensions that are assigned to four categories as summarized in **Table 3**.

Table 3. Dimensions of information quality [11]

Quality category	Scope	Quality dimensions
Inherent	Content	Reputation; free of error; objectivity; believability
Representation	Appearance	Understandability; interpretability; concise representation; consistent representation
Purpose-dependent	Use	Timeliness; value-added; completeness; appropriate amount of data; relevancy
System support	System	Accessibility; Ease of manipulation

The selected definition of information quality is split into four categories, i.e. inherent, representation, purpose-dependent and system support. Each category has dimensions that characterize information by two to five dimensions. A description of some definitions of these quality dimensions is provided in Table 4.

Table 4. Selection of quality dimensions and their description (based on [12])

Quality dimension	Description
Reputation	Credibility of information from the information user's perspective
Free of error	Not erroneous; consistent with reality
Objectivity	Based on fact; without judgement
Believability	Follows quality standards; significant effort for collection and processing
Understandability	Meaning can be derived easily by information user
Interpretability	No ambiguity concerning the actual meaning; wording and terminology
Concise representation	Clear representation; only relevant information; suitable format
Consistent representation	Same way of representing different information items
Accessibility	Simple tools and methods to reach information
Ease of manipulation	Easy to modify; reusable in other contexts

In order to receive a specific statement about the actual quality of an information item, the as-is characteristics of the item must be compared with the required characteristics. The better the matching is, the higher the information quality is considered to be.

For a number of activities in NIMBLE, quality of information will be a key issue (e.g. information that is used to assess the quality of a supplier). The quality metrics will come from a quality management concept (guide book) developed in T6.4 and integrated into metrics collection of the platform. The concept foresees that quality indicators are identified and relevant formulas developed giving an indication about the quality of the information exchanged via the NIMBLE platform. Quality dimensions follow the common approach proposed by Wang and Strong (see above). In some cases, the information quality management expects that meta-data about information sources are added (e.g. the type of sensor used for a monitoring service of a machine). The architectural structure and positioning of the usage quality metrics component(s) will be further elaborated, designed, and implemented as a part of the advanced

version of the platform. This task will feed off experimentation with the four use cases and their respective requirements.

3.7 Security

NIMBLE will make the users aware that they are the first line of defence to protect their own data. Such awareness will be realised through the user's registration agreement on the platform and through fine-grained authorization and access controls.

The state-of-the-art security and privacy methods in NIMBLE will be achieved by relying on security standards, recommendations and open source tools. This approach avoids the "security by obscurity" trap in which security is achieved by hiding the internal mechanisms of a system. This is an important design decision that complements the microservice approach, where each service is publicly accessible by default.

Standard security mechanisms will be used to accomplish confidentiality, integrity and availability in the microservice architecture.

Confidentiality

Measures to ensure confidentiality prevent unauthorised entities from accessing sensitive data on the platform. Therefore, users must be authenticated and authorised on the platform in order to access sensitive resources. In NIMBLE, we will apply methods from the Spring Cloud Security project to ensure a high level of confidentiality on the platform. The Spring Cloud Security framework is built on top of the Spring Security with the focus on securing distributed architectures. Open standards such as OAuth2 and OpenID Connect are fully supported by this framework. The feature set of Spring Cloud Security includes:

- Support of common Single-Sign-On (SSO) patterns;
- Relay of security tokens between individual microservices;
- Out-of-the-box integration with already used technologies (e.g. Cloud Foundry and Spring Cloud);
- Transition between frontend and backend technologies;
- Standardised interaction with authentication servers.
- The confidentiality can be solved using the following three security mechanisms: authentication, authorisation and identity management.

Authentication

Authentication is the process of securely identifying acting entities on the platform. Challenging these entities by asking specific questions is a common technique for proving the identity of actors. This mechanism is called *challenge-response authentication* and is a common procedure for authentication. A concrete realisation of this technique includes prompting the user for their credentials (i.e. username and password), which are only known to the user. A standardised protocol for this workflow is OpenID Connect, which is built on top of OAuth2.

After a successful authentication, the identity is encoded in a JSON Web Token (JWT). This token is digitally signed by the authentication server and attached to each request between communicating microservices. Identities are determined through validation of the digital signature and decoding of data stored in the token.

Authorisation

Authorization is the process of granting access to protected resources. Authorisation handles access rights and ensures that resources are only retrieved and altered by legitimate parties. An open standard for realising authorisation on the platform is the OAuth2 protocol, which defines four different roles:

- The resource server holds protected resources;
- The resource owner is able to grant access to the protected resources;
- The client is an application that wants access to the resource;
- The authorisation server authorises the client to access the protected resource.

In the most trivial case the resource owner is a user of the platform. In a scenario where service A want to access data from service B, service A is the client and service B is the resource server holding protected data. In a more complex scenario the client might also be represented as an agent on the platform, which works on behalf of the user. The authorisation server is often combined with the Identity Management server in order to limit the complexity of the system. After appropriate authentication, the resource owner grants the client permission to work on its behalf by issuing an access token. This token is attached to each request and authorises access to protected resources. The OAuth2 standard provides different workflows for obtaining and distributing access tokens.

Identity Management

Managing identities in software projects is a recurring problem. One of the possible solutions for Identity Management in NIMBLE is called *Keycloak*, addressing SSO, identity brokering, user federation and a web-based admin console. Keycloak supports standard protocols such as OpenID Connect, OAuth2 and SAML and is distributed under the Apache 2.0 license. In addition to Identity Management it can be used as authorisation server. Another solution to Identity Management in NIMBLE is to use the open source tool – Identity Manager (IDM) with dynamic and static policy enforcement.

Integrity

Data integrity refers to protecting data from unauthorised modification and use by third parties. Accuracy, consistency and trustworthiness in the entire data lifecycle are achieved by utilising existing standards and common workflows. The Transport Layer Security (TLS) is used to ensure data integrity on a communication level. Asymmetric cryptography is applied to encrypt exchanged messages, while integrity is checked using a keyed-hash message authentication code (HMAC). HTTPS is built on top of TLS and is the underlying technology for the applied RESTful communication between services.

Integrity on a data store level is a major challenge, due to highly diversified technologies in the architecture. Best practices for designing microservices suggest creating a separate data store for each microservice. In order to keep track on changes on the separate storage systems, there are tools that perform Master Data Management (MDM), which examines every database created under a specific reference, e.g. IDs of the user or device that created an entry into a data store. MDM operates in the background in order to find and fix inconsistencies.

Availability

The platform must be protected against downtimes caused by malicious attackers. Therefore, measures against denial-of-service (DoS) attacks and network intrusion should be applied.

The NIMBLE system should be up and running even if single microservices become unreachable, when fall-back mechanisms must be activated. Netflix Hystrix provides fault

tolerant mechanisms and avoids failure propagation in the system by providing mechanisms for fall-back procedures, in case of errors. Netflix Hystrix is part of the Spring Cloud technology stack and can therefore be easily integrated into the existing technology stack. Intrusion detection systems mainly perform log analysis and policy monitoring in order to detect unauthorised intruders in the system. In NIMBLE we will use the OSSEC intrusion detection system to detect and react on malicious behaviours on the platform.

Data Security Management

Data security challenges target protection of private user data and IoT information stored on the devices and/or in WiFi data storages allowing hackers to gain access to other systems (e.g. via SSID, a Service Set IDentifier, "a network name"). Cybersecurity technologies that can be used to prevent data leakage are focused on either securing IoT gateways or securing the products. IoT gateways aggregate sensor data, translate between sensor protocols and connectivity models, filter and process sensor data before sending it onward, perform updates, security, pass through the alerts, run rules engine and take independent actions, e.g. turning on the air conditioning and more.

In 2016, most Internet disruptions were caused by DDoS (Distributed Denial of Service) attacks, like the Mirai botnet that brought down Twitter, the Guardian, Netflix, Reddit, CNN and many other sites in Europe and the US, in October 2016. The Mirai botnet is largely made up of IoT devices such as digital cameras and DVR players. The same botnet was used in September 2016, in an attack on the information security blog "*Krebs on Security*".

Some security practices to minimize the negative impacts of DDoS include:

- Performing security assessments to look for DDoS related vulnerabilities
- Applying network security controls, including services from cloud-based vendors for DDoS
- Software updates (patches) and bug fixes
- Proactive network monitoring and alerting.

Therefore, data security management in NIMBLE needs to provide data integrity and to make sure that data cannot be corrupted, e.g. data are not accessible by unauthorized parties. Some of the viable solutions for the data security management are:

- Data encryption that allows only an approved party to decode data, using a specific key;
- Data authentication that allows access to data only to the users with the proper key or password (see Section 6.1 for more details on how it is planned to be done in NIMBLE);
- Data protection software that keeps unauthorized parties from accessing private data and can be also used for giving different access permissions to different people;
- Data protection while in transit, which is usually based on IP security.

NOTE: A complete list of security methods to be used in NIMBLE will be elaborated in D6.2 "Design and implementation of Security and Privacy for Core Business Services".

3.8 Communication infrastructure

The main purpose of this component is to provide generic communication capabilities among different NIMBLE components and participants. It can be used to send messages and notification among components or to share information among various companies. The dominant paradigm shall be the publish/ subscribe pattern leading to event-based communication among collaborating partners by registering interest in particular events. Event-based communication is also recommended by IIRA and RAMI architectures as it decouples

any producer and consumer in terms of location and time (asynchronous communication). Using this paradigm producers and consumers do not have to be aware of each other and need only to agree on the topic via which they are going to communicate. We will employ open source tools such as Apache Kafka, ActiveMQ, Apollo for the Messaging and Communication Framework.

The communication infrastructure is meant to provide group communication and membership services to the entire cluster including all its internal components. That infrastructure can be used internally by components to support their own operations, for example for passing information from catalogue related application to business processes. Moreover, applications running within the platform can make use of these services to communicate between themselves.

Data management notifications can use this mechanism for notification of subscribed events to be dispatched and shared with interested subscribers. The offered services will enable higher layer capabilities, such as fault tolerance and application integration and cooperation.

3.9 Front-End

The front-end provides a graphical user interface for accessing the NIMBLE capabilities. It serves as a central receiver for user interactions/requests and delegates those requests to the appropriate services (identity, search, publish, communication, negotiation, matchmaking, IoT analytics).

The design is focused on creating a modern responsive web based application that is accessible on literally any device. In order to meet this design goal and create a uniform look and feel the frameworks "Bootstrap"[29] and "jQuery"[30] will be used. The resulting front-end should be intuitive and easy to use by the intended end-users.

Since the front-end will be managed within one central service, "Angular"[31] will be used to facilitate modular development, templating and code splitting.

Furthermore, Angular enables useful features such as IDE integration, unit testing, dependency injection and deployment of native mobile and desktop applications.

In general, the front-end will follow a "one-click-to-get-it-done" philosophy focusing on the ease of use. Detailed requirements will be documented throughout T2.4 "User Experience Design for Fast System Adoption".

4 A view of the architecture from the stakeholder's perspective

The front-end is the main component which external users encounter when dealing with the NIMBLE platform. The main task of the front-end is to help users access internal capabilities of the platform.

[29] http://getbootstrap.com/
[30] https://jquery.com/
[31] https://angular.io/

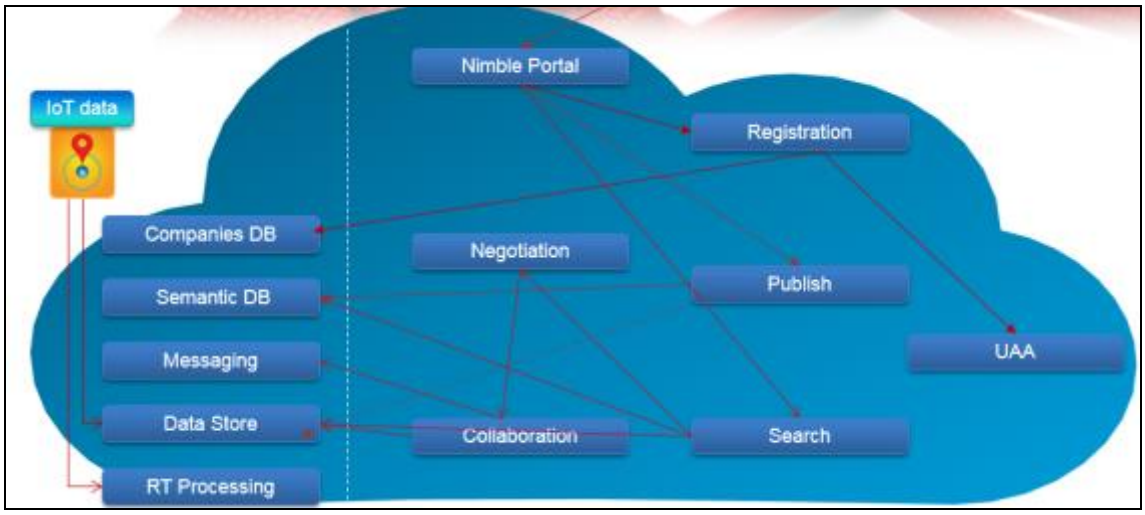

Figure 11: NIMBLE core platform - a high-level view

The first action to be taken by an end user would be to *register* his company with the platform. The registration process will involve security and identity management aspects tightly integrating with the UAA component. Moreover, several different roles within a company can be registered, such that different people would be able to perform different level of tasks. The registration process interfaces with a database component to track entities and their related metadata.

The following task to be performed by an external user would be to *publish* a catalogue including capabilities and services they can provide. The catalogue would be stored in an internal registry which will be used later for retrieving relevant information. A complementary task would be for a company to *search* for supply chain partners. In that case, a query would be invoked from the front-end to the NIMBLE platform. These tasks are aided by a service discovery component which in turn relies on a services registry.

Finally, a *business process* is initiated after going through a matchmaking and negotiation process, and the relevant flow of information and transactions can begin for the two entities aided by the cloud service bus, which can connect different entities and companies in the platform.

An additional function facilitated by the front-end application is that of *IoT data flow* registration and manipulation. Once that is done the NIMBLE data management component comes into play which serves as the point of entry for all related data, providing advanced processing capabilities, and providing interfaces to access that data. That data in turn may serve as an integral part of the business processes.

The main or characteristic excepted flows within the platform are presented to cover the different roles that exist around the NIMBLE platform. First, there exists a **platform provider** which is in charge of deploying and hosting the NIMBLE platform and its daily operations as well as sector-specific extensions requiring ICT expertise.

End-users interact with the system via the front-end to carry out all the different capabilities provided by the platform.

A separate role is that of an **IoT device data provider**, who wants to expose data to the NIMBLE platform to be used, operated on, or exposed by NIMBLE applications for business collaborations.

4.1 NIMBLE Platform Provider: Out of the box deployment

The NIMBLE platform consists of a specifically tailored customization of a cloud environment, for the B2B realm NIMBLE targets. A platform instance being deployed is federation-ready by including an API covering the core services to be provided by each individual NIMBLE instance.

Along with the platform, several NIMBLE related middleware services are deployed and made accessible to platform users. Externally accessible functionalities for registered and authenticated / authorized users include publishing and searching through catalogues, while internally used functionalities include data processing services such as a NoSQL DB or a streams processing engine. A data store supporting semantic queries will be deployed as well, even though in first versions free-text based searches will be supported and evaluated. In addition, a cloud service bus will be made available to internal components for communication, data sharing and notification purposes. Some such capabilities, such as notification, may be available for external components as well. Internal applications will be available to orchestrate the available microservices to provide a cohesive set of capabilities.

NIMBLE provides a programmatic API which will enable users to access capabilities programmatically, and this will form the basis for the federation capabilities.

The platform out of the box is a B2B tailored PaaS, thus it exhibits a cloud environment taking care of internal management of resources needed to run the platform, transparently to the end-users. The platform comes with readily available software and hardware ready to host the platform internal services.

4.2 IoT Objects Provider

IoT devices may be manually registered with the platform through the front-end or through the API. At registration time, object metadata needs to be provided. The registration will take care both of the data and metadata portions of the device in question.

Once the device has been registered, the object is ready to be found and becomes operational, and reflects the data from the external world. Data updates will be performed programmatically by the device, and will be processed and stored by the platform.

Storage and processing of incoming data will be offered as services to incoming IoT data. Notifications based on data state will be made available as well to be used internally by NIMBLE components.

Data and metadata may be stored and indexed via ElasticSearch to retrieve them quickly. The IoT services may be registered on Netflix Eureka to grant a quick discovery and retrieve.

4.3 End-user: Interaction with the platform

As already mentioned, the main component dealing with end-user interaction is the platform front-end. Nonetheless, some functionality shall also be directly accessible via API endpoints in order to facilitate integration with existing platforms and legacy systems.

Generally, one can break the interaction capabilities down into the following main features:

- Registration of users and companies with different roles and affiliations (UI only)
- Product and service catalogue publishing (UI and API)

- Establishing a business process; includes searching for supply chain partners, matchmaking and negotiation (UI and API)
- Managing data flows and processing capabilities (UI and API)
- Integration of "value added" services (depends on the specific implementations)

Overall, different access policies can be applied to every interaction and some functionality can be limited to specific user roles (e.g. company details shall only be editable by an eligible representative).

Thus, the UI views will be created dynamically based on the user role and access rights. Corresponding features will be implemented throughout T3.7 "User Front-End Prototyping for Fast System Adoption".

5 Relationship with reference architectures

The two reference architectures that correspond to NIMBLE are the Industrial Internet Reference Architecture (IIRA) and the Reference Architecture Model Industrie 4.0 (RAMI).

The IIRA aims to serve as a template refence architecture for the creation of standards based Industrial Internet of Things systems. It comes to unify the language and the concepts used. RAMI shares some of the same aspirations as the IIRA by attempting to establish a unified vocabulary among different stakeholders, attempting to promote a structured way for targeting Industrial IoT systems. At its core, RAMI is based on SOA principles, and strives to break down the complex system into the individual components and establish proper relationships among them.

Figure 12 depicts the NIMBLE technical architecture along with the main technical components. The NIMBLE architecture is aligned with the three-tier implementation pattern of the IIRA. The components interacting with IoT-enabled devices are situated in the **edge tier**. Under-the-hood enablers like data management are aligned with the **platform tier** and enterprise applications from supply networks and user applications are situated at the **enterprise tier**.

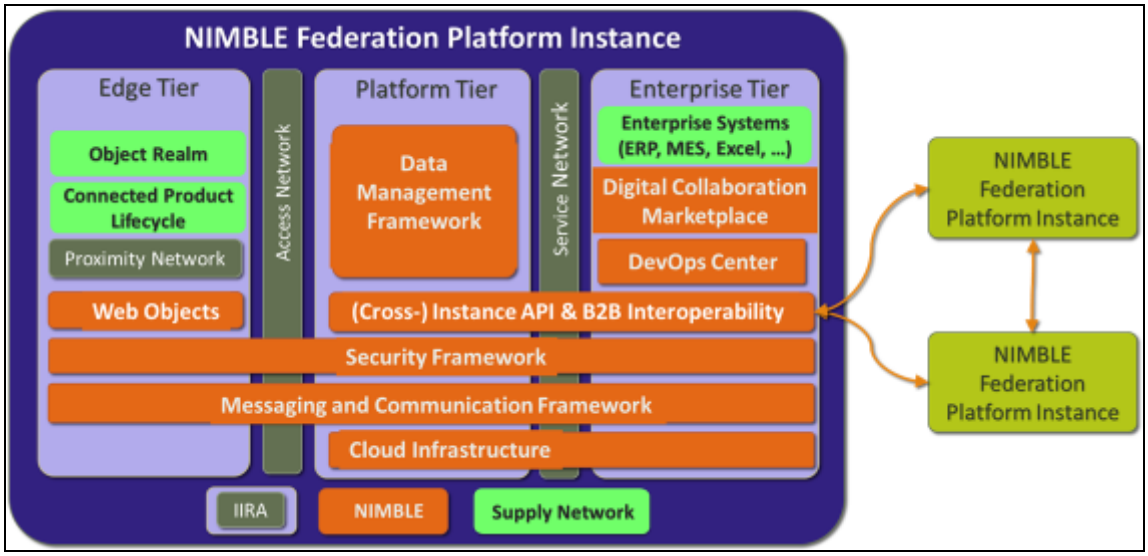

Figure 12: NIMBLE and IIRA

The edge tier represents elements of the real world which interact with the platform, mainly to push data along with sending or receiving commands or instructions. The heart of NIMBLE lies in the platform tier, in which all relevant capabilities are developed and deployed. The enterprise tier in turn will consist of applications running outside the platform and using the platform infrastructure and services to achieve their own goals.

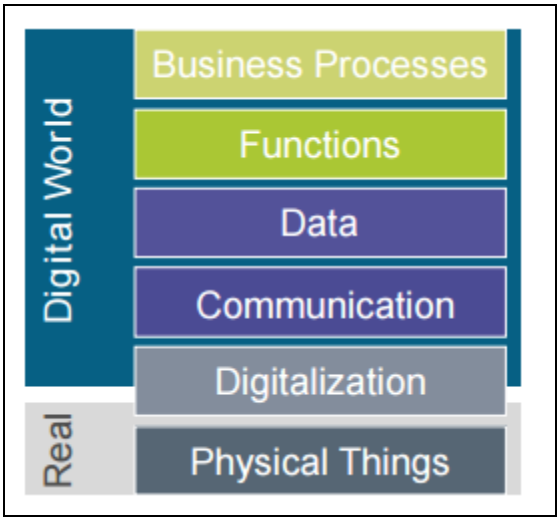

Figure 13: RAMI architecture

Similarly, Figure 13 provides a simplified view of the RAMI proposed architecture, which corresponds very well to the NIMBLE proposed architecture by having the real world communicate with the platform, having its data pumped in, which forms the basis for internal platform functions, which in turn feeds the user defined business processes.

The main motivation behind both reference architectures is the lack of a common understanding of what the Internet of Things means for the industrial sector, especially from an architectural perspective. Despite the great progress in the availability of devices and systems in this field, not so much progress has been made in the definition of common reference models within industrial settings. In order to deal with the lack of clarity, these efforts propose a Reference Architecture, which represents the basis for designing any concrete architecture. This is combined with a Reference Model, which stresses a common language and understanding. Both these efforts have been used as a reference and inspiration for the design of the NIMBLE system.

Table 5: Mapping concepts of RAMI/IIRA to NIMBLE components

RAMI / IIRA	**NIMBLE**
Administration shell	Data management
Events based communication	Cloud service bus
Data analytics patterns	Big data toolset
Three tiers architecture	Similar

Once the concrete NIMBLE architecture has been validated in use, we will publish our lessons learned by comparing it with respect to the IIRA and RAMI architectures.

6 External interfaces and technologies to access NIMBLE

As discussed in the previous sections, NIMBLE will provide a set of services (core services and value added services, cross-instance federation), which should be accessible in a programmatic way via an API and a human-interactive way via a GUI. Target users for the APIs are application developers, while target users of the GUI are usually domain experts from specific industrial sectors. Also, the application developers may have domain knowledge of a specific sector.

In most cases, the GUIs will simply use the APIs for accessing the functionalities of the NIMBLE services. As the User Interfaces will be responsive Web UIs (see Section 3.12 "Front-End"), the services should provide a RESTful API based on HTTP for its public services. Data should be exchanged in JSON-Objects, following a documented structure. A short documentation of each API shall be provided to allow other developers to make use of it without major difficulties. It is recommended to use the Swagger Framework[32] for the API documentation. Using this YAML-based specification format would allow to create documentation as well as an implementation of client libraries in more than 30 different programming languages. In addition, it can also be used to generate server stubs in different formats. This enables the platform to provide documentation as well as downloadable example client implementations without any additional effort.

Some industry sectors / application domains may require SOAP/WSDL services to allow seamless integration with legacy applications. Using SOAP is therefore an option for the implementation of value-added services. Still, all core services must at least provide a RESTful API Endpoint.

In both cases HTTP(S) will be the primary choice for data transport. Session management and security features from HTTP can be used as needed by the services. For the create/read/update/delete data manipulation operations, the HTTP methods for GET/POST/PUT/DELETE will be used.

All services should be published via a central access point, called "Gateway Proxy". Although the loads can then be distributed to multiple instances in case of high performance requirements, the address of the gateway proxy – and therefore the exposed API address – should be stable. The Gateway Proxy (e.g. Netflix Zuul, HAProxy, GoRouter) provides routing, filtering, and load balancing capabilities. For doing an API-level request logging and rate limiting, the use of an additional framework, such as Kong[33] is considered.

For testing of the NIMBLE APIs command-line tools such as cURL[34] or Postman/Newman[35] are recommended.

Finally, we propose the use of the following URL naming pattern for all NIMBLE services:

https://<nimble-platform-instance-address>/<service-id>/[<version>/<subservice/.../>/]

with the following components:

- <nimble-platform-instance-address>: this is the DNS-name and port number to access the platform instance from the public internet

[32] https://swagger.io/
[33] https://getkong.org/
[34] https://curl.haxx.se/
[35] https://www.getpostman.com/

- <service-id>: instance-unique id or name of the service. This part is managed by the Gateway Proxy, which will forward requests to the actual service in the microservice architecture.
- <version>: Optional version number of the service, if different version must be maintained for backwards compatibility. If no version number is used by the request, the service should automatically provide its latest API.
- <subservice>: this parameter directs the call inside the service to several sub-services or simple html-pages. The subservice name "**api**" is reserved and should point to the documentation of the service. This could be the swagger specification (api.yml) or the auto-generated HTML-documentation.
- The definition of service parameters (e.g. GET or POST parameters) as well as response types is up to the service itself, and will be defined in the API documentation of each service.

7 Mapping the use-cases to the architecture

This section discusses how the use cases for each Project Pilot utilize the NIMBLE core platform in order to deliver the envisioned functionalities. The Pilots consist of several use cases that demonstrate and exercise different platform capabilities. The mapping of the use cases to the NIMBLE architecture requires us to:

- Identify the NIMBLE core components that are used in each use case.
- Specify the interaction with various NIMBLE components and services.

The use case mapping will assist in identifying how the proposed architecture will be utilized for both the developed Pilots but also for future developments based on the NIMBLE platform.

7.1 Use Case 1: White Goods Service Supply Chain

This use case revolves mostly around company data, which is distributed in many locations and systems, and the objective is to distil a unified overarching view based on distributed data. There are two distinct scenarios, the first in which data sharing should be achieved between different entities to improve overall processing and future design of the product. A second scenario centres on the concept of a product avatar, which means being able to view a product from 360 degrees throughout its lifecycle, regardless of the location of the information.

NIMBLE is expected to provide unified access to information about a specific product combining data coming from different and separated data sources. Search and data analytics services are expected to accompany this capability.

Security and roles based access is expected on all scenarios.

The two detailed scenarios are expected to be a) the correlation of issues from field service to manufacturing process, and a Product Avatar (to easily access a product history throughout its lifecycle).

Collect and correlate data coming from different sources, to enrich relevant information, and ensure proper dispersal of that information. Lifecycle Management of data and the Product Avatar services will mainly be exercised by this use case.

7.2 Use Case 2: Eco Houses Supply Chain

The main motivation for this use case is to improve the supply chain by connecting stakeholders and data from throughout the process. Lindbacks need to keep track of the original design of all produced units, along with the individual changes applied to each individual unit. The main purpose would be to add quality and IOT-data along the supply chain throughout the lifecycle, from design through to maintenance. All data and changes should be tracked, and proper notifications should be sent when changes occur. All access to data should be governed by proper authentication and authorization.

7.2.1 Product configurator

A customer can ask for changes in the properties of a bathroom of his future eco house. The catalogue component enables specification of a product configuration by selecting desired modifications among the available options for different components composing the bathroom. The customer sends his modification request to Lindbacks, thus initiating a new business process, through which the first stage would be looking for a sub-contractor that can satisfy the new request. Once, such an entity is located, a negotiation process is initiated in which both parties agree on the terms of the engagement. Once an agreement is reached, Lindbacks can reply to the customer with the details of the offer to realize the desired changes. Upon authorization from the customer, a new order is being sent to the appropriate sub-contractor.

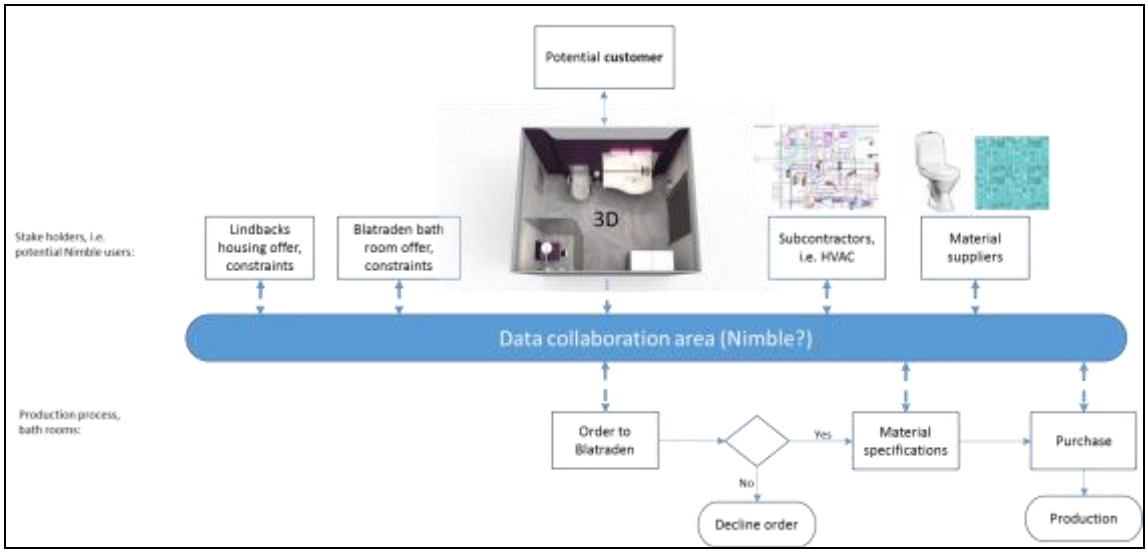

Figure 14: Product Configurator

This scenario will exercise in particular the search, business processes, and communication services of NIMBLE.

7.2.2 IoT Measurements in bath rooms

Various sensors are installed in some of the bathroom units, to track environmental conditions such as temperature, humidity, water leakages and energy consumption. These measurements should help both maintenance as well as future, improved design of similar units.

Water leakage and its associated damage is a recurring issue. This process aims to help better understand whether the issues are related to the design of the units, to installation procedures or

to the actual usage. The connection of relevant IoT devices to the data ingestion portion of the platform, along with appropriate data processing, including data in-flight and historical data, will play a central role to realize this scenario.

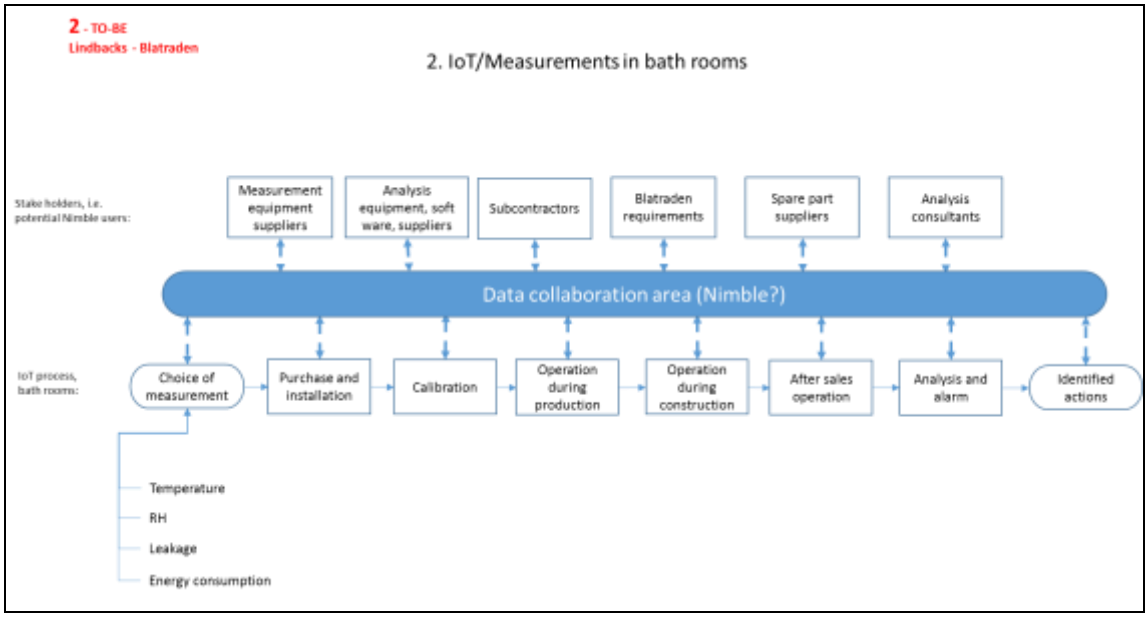

Figure 15: Bathroom IoT measurements

This scenario will exercise in particular the IoT data ingestion and processing services.

7.2.3 Tracing components and Quality Control Information

With the ability to trace individual parts forming constructed modules the supply chain should become more efficient, being able to trace also individual parts throughout the lifecycle of the module, from production to integration, transportation, and final assembly.

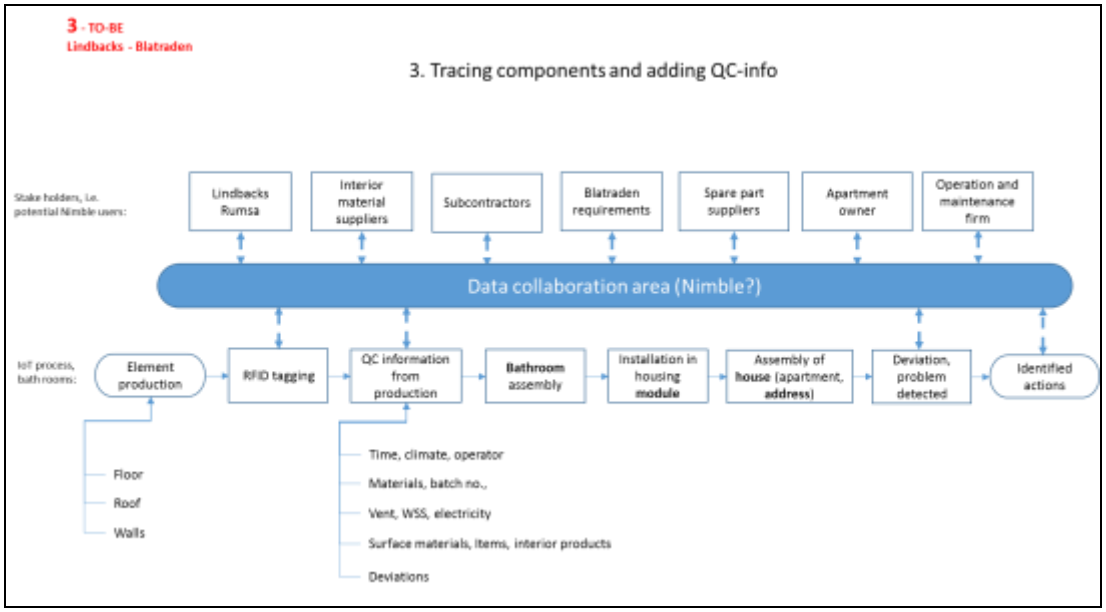

Figure 16: Tracing components

7.3 Use Case 3: Textile Manufacturing Supply Chain

There are four target use cases for NIMBLE in textile manufacturing. As described in D1.1 the scenarios are *Design Collaboration, Order aquisition, Order production Monitoring, and Certificate of Origin generation.*

The work done in the MODA-ML project created on the one hand a list of documents useful for a B2B collaboration in the textile sector, and on the other hand has prompted companies to have an infrastructure to exchange documents. The documents are available under the MODA-ML archive that is currently maintained by ENEA and we propose the use of this standard for this use case. The infrastructures unfortunately are not standard and sometimes are tailored software solutions for each company, thus it may happen that a company sends a document by email or similar. At this moment the most accepted solution, already in place in the companies and already implemented in the product of Domina DotRunner, supports web services, for the exchange of documents between companies.

The NIMBLE platform suggests the publisher/subscriber pattern for a B2B comunication, which is an appropriate solution for most of software infastructure, where a component should communicate with others, but the presence of an existing method of comunication requires the introduction of an adapter.

Figure 17 shows the first pattern *"Direct Pattern"* where the Supplier needs to send a new document to the Customer, the message is sent to the Nimble platform through the sendMessage service, the platform looks for the same service of the Customer and forward the message to it. This pattern is easiest to integrate in NIMBLE, where each partner subscribes its sendMessage service filtering by recipient, the most important point is the subscription approval by the publisher in the negotiation phase.

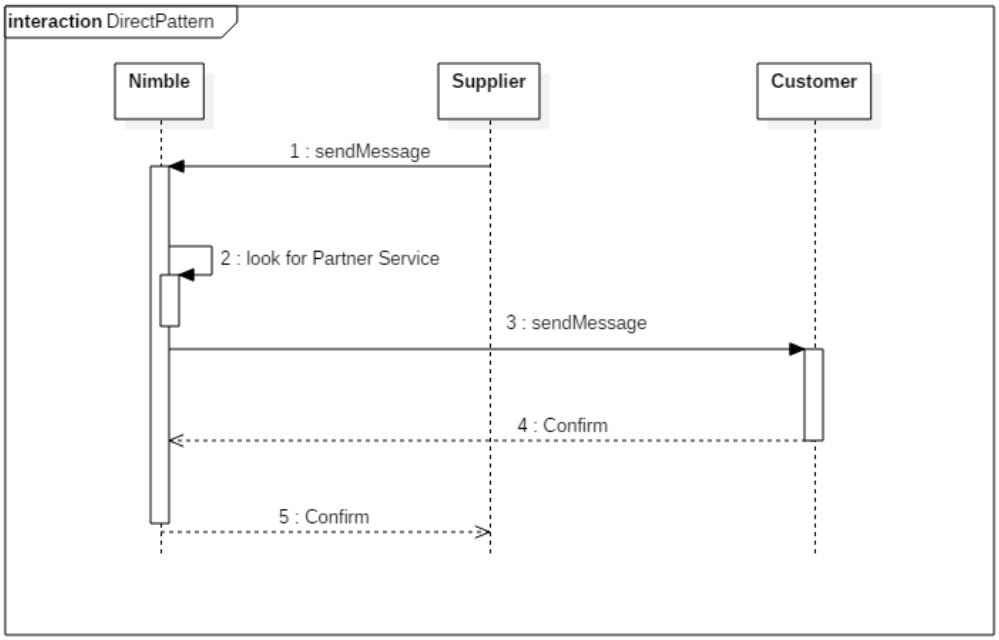

Figure 17: Direct Pattern

Figure 18 shows instead the second pattern *"Polling Pattern"* whereas like in the previous example the supplier needs to send a new document to a customer that has no fixed infrastructure for receiving messages (i.e. a laptop). In this case, the NIMBLE platform receives

the message for the recipient and adds it to a queue. Once the Customer is able to receive the messages he invokes the getNextMessage service that returns the queued message for him. In order to integrate in NIMBLE, we could use the same approach described before, but replacing the Customer sendMessage service with a queueing service instance that receives the messages for the partner and forwards them once its getNextMessage is called. Also in this solution, the negotiation phase is very important, to avoid sending messages to unintended third parties. Thus, an authentication service is required

In both patterns for each single service call a confirm response is received, to ensure the correct execution of the service or the error number in case of a failure.

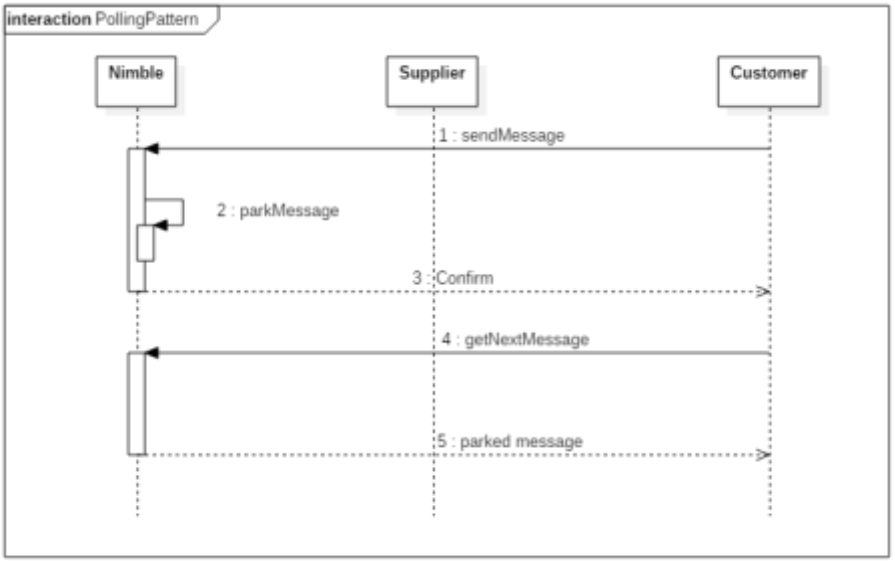

Figure 18: Polling Pattern

This pattern is very important, for the collaboration scenario, where the Customer has its own account, linked to a company and doesn't need to import data in the legacy system of the company view only the article prototype. In general, this pattern is useful for all cases where you need to manage special company where you have only one person (sales agent, designer freelance, etc...). The last consideration to bear in mind and respect is to maintain compatibility in the implementation of the WSDL that describe the two services.

The other communication service that the use case needs to adopt for the *Monitoring Scenario* is the IoT functionality of NIMBLE. In the FITMAN project Piacenza implemented a monitoring system for some departments and machines integrating the FITMAN component respect the FIWARE NGSI [36]Context Management specifications.

SCENARIO IMPLEMETATION

The following part describes the communication between the partners and the NIMBLE platform for each scenario, in all diagrams the *Direct Pattern* is used, but all scenarios should support both patterns. The first scenario described in the D1.1 is the *Collaboration Scenario*.

[36] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/NGSI-9/NGSI-10_information_model https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.IoT.Gateway.DataHandling.Api

The scenario starts from requiring a specific article design of the customer. This will be translated into a document and sent to the NIMBLE platform with a specified supplier as recipient. Under a permission supervision the document with the specified information is forwarded to the supplier, who makes stylistic changes to the model and responds back the new article offer. This loop could be made several times until the customer's requirements are met, then an approval of the article is done in order to proceed to the order acquisition scenario.

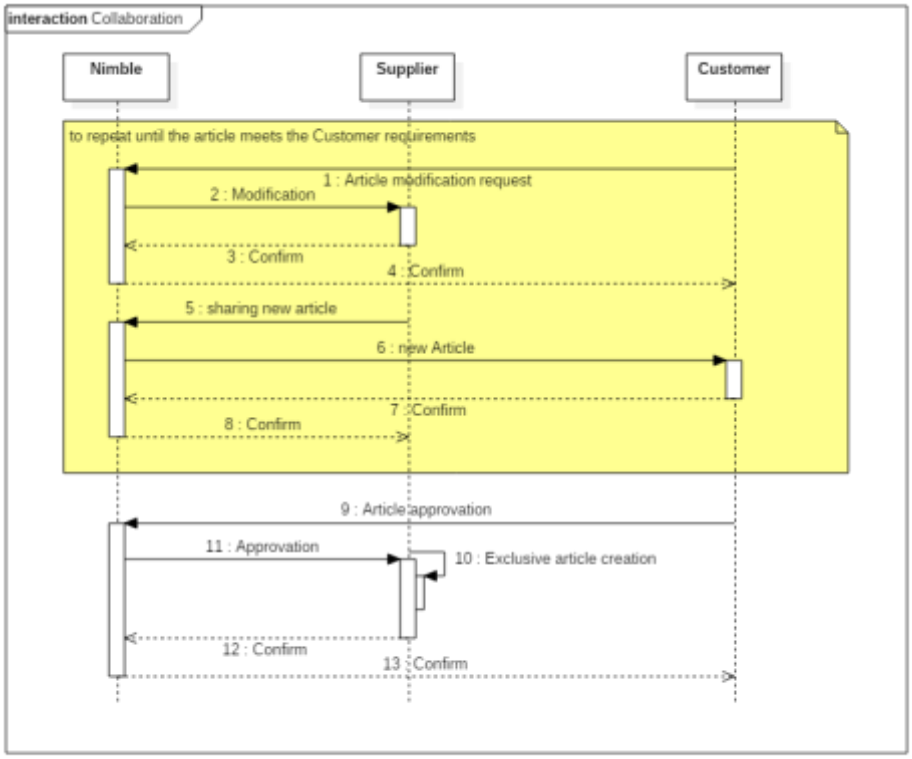

Figure 19: Collaboration Scenario

The *Collection Presentation* is similar to the previous scenario, the main difference being that the communication remains only in one direction, from the supplier to possible customers. Also in this case, the permission control of the messages is important, because the seasonal catalogues preview is not public, but is made available only for a specified distribution list of customers.

The scenario as previously, starts with a customer request, in this case the request is sent to all catalogue publishers and each one responds to the request with the relative documents that contain the data for him. This workflow ensures the most flexibility to the supplier to offer the best solution for the customer, based on his own rules that could often change for a single request.

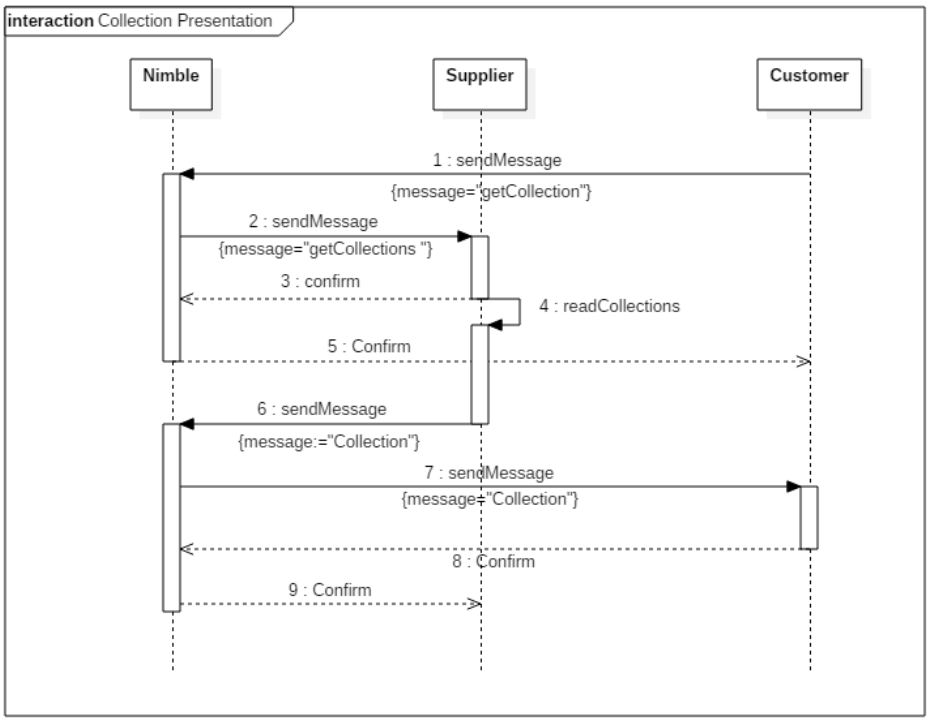

Figure 20: Collection presentation

The Order acquisition follows the same pattern as described above apart from the documents that are specific for this scenario.

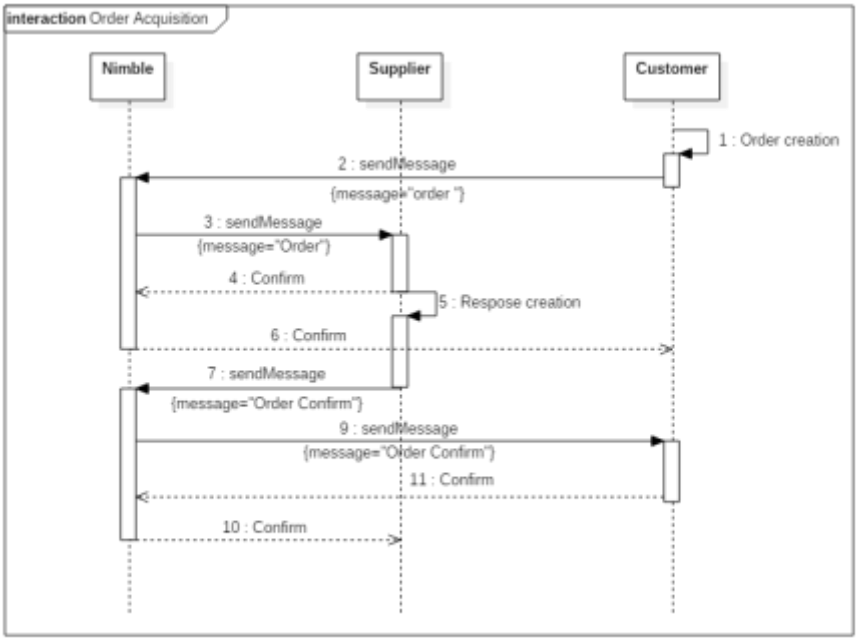

Figure 21: Order Acquisition

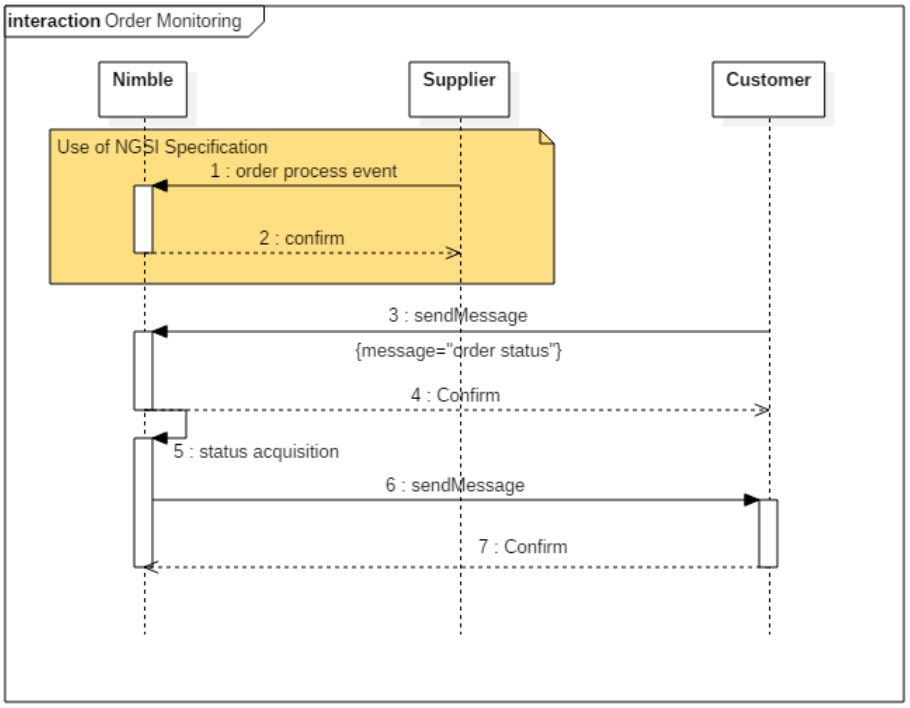

Figure 22: Order Monitoring

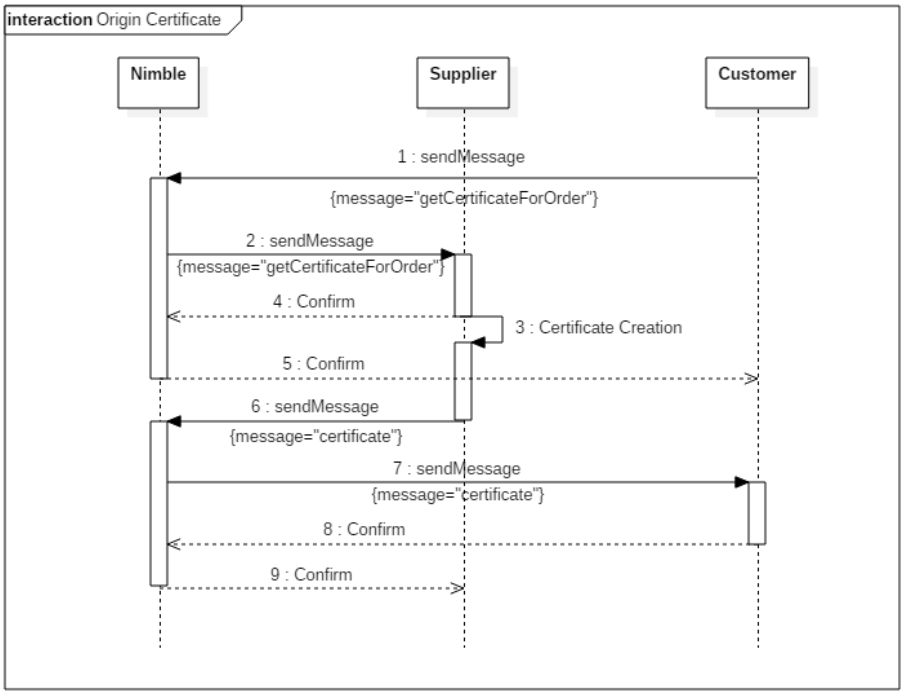

Figure 23: Origin Certificate

7.4 Use Case 4: Child Furniture Supply Chain

The main capabilities required from this use case revolve around publishing catalogues and being able to search through existing information in a helpful manner. The expectation is for companies to be able to publish their catalogue dynamically through the NIMBLE platform, and have other entities being able to search and browse through published catalogues in a guided manner. The guided manner should support for example relevant taxonomies and ontologies, potentially presented as multiple facets through which a user can interactively narrow his search space. There should be various characteristics by which to filter or narrow the search scope, such as reputation of the counter-party.

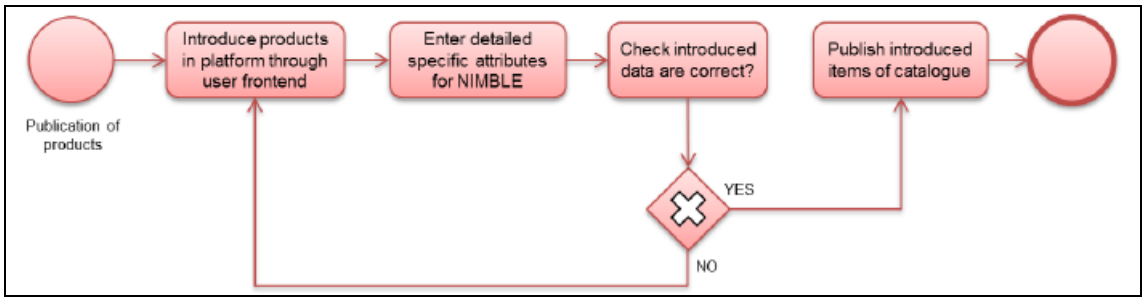

Figure 24: Publish catalogue

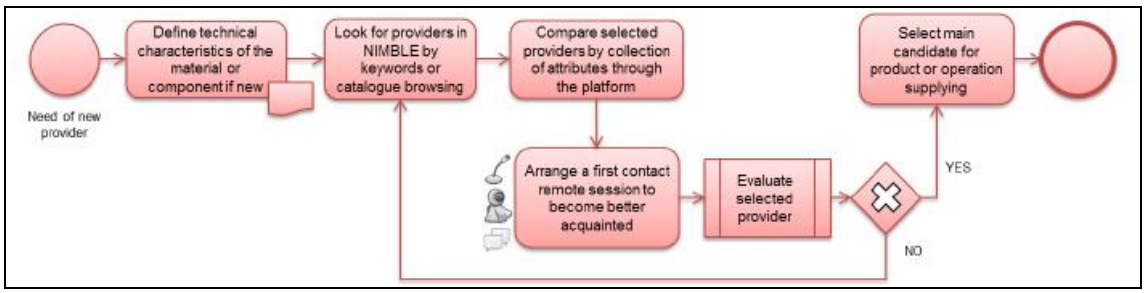

Figure 25: search for partners

In addition, once a potential company has been found there is a need for establishing a private communication exchange mechanism among the interested parties, to form a step in the supply chain and to support the process through its different stages.

Security aspects, such as access to data storage or to the data exchanged via the communication channels, with proper authentication and authorization is paramount to this scenario.

The business process components of NIMBLE shall be used as well, for negotiating with known and new supply chain partners.

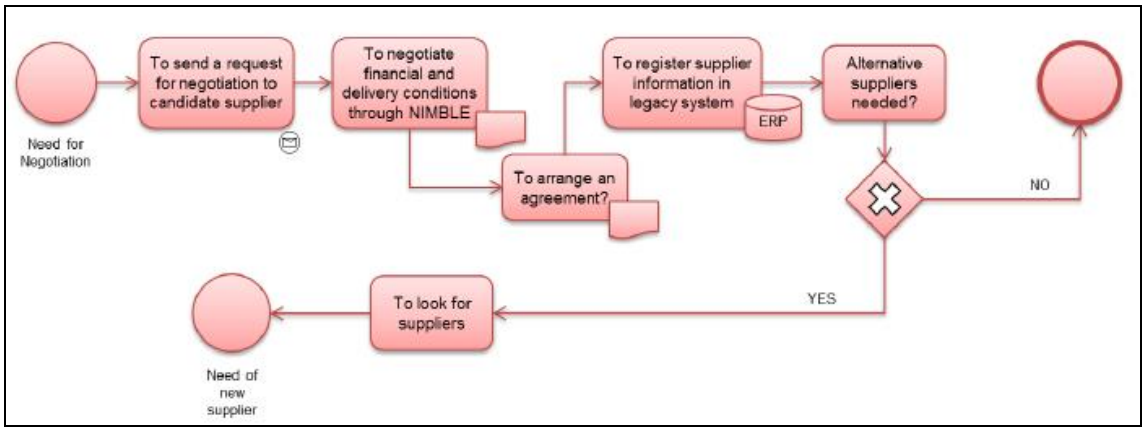

Figure 26: negotiations process

This scenario will exercise in particular the NIMBLE capabilities for publishing catalogues, search through them, and establish communications channel with counterparts.

8 Summary

The quest leading to the current architecture document has started off with the requirements document to ensure that the architecture proposed covers the stated requirements and used the pilots to further drive the requirements as well as the validation scenarios. We provide an overview of the different platform components and highlight the interfaces and dependencies between them.

This architecture document frames the high-level overview of the NIMBLE platform, its components, and the main interactions between different components. Thus, at this stage we can break up the large overall NIMBLE architecture and assign components to WPs and tasks, to drive the individual internal low-level design and development of each component.

WP1 works on the requirements that feed WP2 and especially this task and document. WP3 shall implement a core version of the vision laid out in this document, to include and demonstrate base capabilities and interactions thereof. Capabilities include publishing capabilities, to be found by other companies to foster inter-company relationship. An intrinsic part of collaboration may be the flexible and dynamic sharing of data. WP4 shall use the initial platform developed in WP3 to initiate an initial version of the use cases, thus providing early feedback and added requirements for the second wave of platform development. The second wave shall be realized in WP5, as an enhanced version of the platform, focussing on an additional, advanced set of capabilities. WP6 takes an overall view of security aspects of the platform from all directions, horizontal and vertical. WP7 will enhance and validate the use cases running on the enhanced platform release coming out of WP5.

The resulting platform is targeted mainly towards companies seeking collaboration, via which companies can find suitable partners, establish business relationships with them, and monitor the joint venture.

8.1 NIMBLE Unique Selling Proposition

After presenting the NIMBLE architecture along with the introduction of the different components participating in the platform, and the interactions between them we established how companies will be able to take advantage of such a platform.

The platform's uniqueness does not rely on a single component or capability but rather on a tight combination of complementary capabilities in an inter-disciplinary manner. The NIMBLE platform enables the achievement of various complementary and intertwined capabilities.

The first set of capabilities revolves around the ease of registering a new entity and publishing its capabilities, along with complementary capabilities of searching through the published information. That set of capabilities is strongly tied to a second set concentrating on different aspects of business processes, which define what activities companies will perform once they have found each other using the first set of capabilities. Third, data aspects come into play, by which the sharing of relevant data serves as the oil and the glue of the business processes established with the second set of capabilities. All described capabilities are a part of a secure cloud deployment, specially customized for B2B relationships, in which security aspects need to be present at all layers to provide users with the level of trust required for them to make use of such a platform. Finally, all capabilities are presented in an easy-to-digest manner, via a web based front-end.

In a B2B context, the first argument differentiates NIMBLE from B2B messaging / business process management platforms, which provide data exchange between partners that know each other from previous business activities. The second and third arguments differentiate NIMBLE from B2B e-commerce platforms and tendering platforms, which do not provide further structured B2B communication once a buyer identifies a seller on the platform or vice versa.

Moreover, platform instances may work in a federated environment via which different providers may deploy their own instance of the platform, enhancing it for their own target audience, while having all instances capable of collaborating with each other.

Most of the existing and planned components will be released as open source to the community.

9 Bibliography

[1] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapè . On the Integration of Cloud Computing and Internet of Things. Proceedings of the 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014). 27-29 August 2014, Barcelona, Spain.

[2] Apache Jena http://jena.apache.org/

[3] Apache SPARK http://spark.apache.org/

[4] Cloud Foundry http://en.wikipedia.org/wiki/Cloud_Foundry

[5] CouchBase http://www.couchbase.com/

[6] MQTT. http://mqtt.org

[7] Node-RED http://nodered.org

[8] SPARQL http://www.w3.org/TR/rdf-sparql-query/

[9] Wellsandt, S.; Wüst, T.; Hribernik, K. A.; Thoben, K.-D. "Information Quality in PLM: A product design perspective" In: Umeda, S; Nakano, M.; Mizuyama, H.; Hibino,

H.; Kiritsis, D.; v. Cieminski, G. (Eds.): Advances in Production Management Systems: Innovative Production Management Towards Sustainable Growth; Proceedings of the International Conference of Advances in Production Management Systems (APMS); Volume 460; pp. 515-523

[10] International Organization for Standardization: ISO 9000:2005. Quality management systems – Fundamentals and vocabulary.

[11] Wang, R. Y., Strong, D. M.: Beyond Accuracy: What Data Quality Means to Data Consumers. Journal of Management Information Systems, 12(4), 5–33, (1996)

[12] Rohweder, J., Kasten, G., Malzahn, D., Piro, A., Schmid, J.: Informationsqualität – Definitionen, Dimensionen und Begriffe. In: Hildebrand, K., Gebauer, M., Hinrichs, H., Mielke, M.: Daten und Informationsqualität. Springer, 25–45, (2008)