PUBLIC

| Feedback mechanisms Across the Lifecycle for Customer-driven Optimization of iNnovative product-service design |
|---|
| Acronym: FALCON<br>Project No: 636868 |
| FoF-05-2014<br>Duration: 2015/01/01-2017/12/31 |

# PROJECT DELIVERABLE 1.2:

# FALCON Virtual Open Platform Architecture and Interfaces

Content : This report defines the overall architecture of the Virtual Open Platform. The documentation of the architecture includes all its components and defines technical quality standards regarding performance, reliability and efficiency. In addition, state-of-the-art APIs and standards to ensure the ease of use for all interfaces (APIs) between the partners' components and especially all APIs defined for 3rd parties are documented briefly.

Copyright ©FALCON Consortium 2015 - 2017

Versioning and contribution history

| Version | Description | Contributors |
|---|---|---|
| 0.1 | Draft outline | Karl Hribernik (BIBA) |
| 0.2 | Input according to the FALCON modules (Section 2) | Simone Parotta (Holonix) |
| 0.3 | Input according to the FALCON modules (Section 2) | Sang-Je Cho (EPFL) |
| 0.4 | Input according to the FALCON modules (Section 2) and Technical Quality Standards | Marco Franke (BIBA) |
| 0.5 | Input according to the FALCON modules | Wilfred van der Vegte (Delft) |
| 0.6 | Contribution to Technical Quality Standards | Karl Hribernik (BIBA) |
| 0.7 | Input according to the FALCON modules | Patrick Klein (BIBA), Christian Melchiorre (SOFTECO), Panagiotis Gouvas (UBITECH) |
| 0.8 | Input according to software quality and architecture | Marco Franke (BIBA)<br>Christian Melchiorre (SOFTECO) |
| 1.0 | Input according to the FALCON modules | Simone Parrotta (Holonix)<br>Panagiotis Gouvas (UBITECH)<br>Marco Franke (BIBA) |
| 1.1 | Edits for coherency | Karl Hribernik (BIBA) |
| 1.2 | Update FALCON VOP architecture and minor changes | Marco Franke (BIBA) |
| 1.3 | Update harmonization of D1.2 & D2.1 | Marco Franke (BIBA) |

Reviewer

| Name | Affiliation |
|---|---|
| Simone Parrotta | Holonix |
|  |  |

# Table of contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Objective of WP1

Within work package 1 the development of the FALCON Virtual Open Platform (FALCON VOP) will be managed and one major objective is cohering the diverse software components developed in the FALCON project. Establishment and enforcement of suitable methodologies and techniques for evolving, maintaining, and applying the FALCON architecture internally and externally is one of the main tasks of this WP. Along with this WP a requirements analysis within the Business Cases (WP5-WP8) has been conducted in order to reflect end-user needs within FALCON VOP. Focusing on a system of system approach, the definition of external interfaces to the FALCON architecture with potential users (customers) is crucial for the success of the project. The results of the requirement analysis will directly influence the definition of the FALCON VOP interfaces to external systems with particular reference to product-service design and development. Here the integration of feedback acquisition module, modules for PUI collection and interchange of multi-domain knowledge developed in RTD WPs is part of the work. In addition, the architecture team is responsible for ensuring the quality of all aspects of the architecture, including performance, maintainability, and most importantly the usability of internal and external interfaces. State-of-the-art principles of measuring and ensuring i.e. the usability of the developed programming interfaces will be applied. The quality control process will be executed continuously throughout the project and applied to public programming interfaces as well as to the development process of internal components of the FALCON VOP.

In summary, the main objectives of WP1 are:

- to specify, analyze and evaluate technical and functional requirements of the FALCON VOP
- to develop, evaluate and maintain the architecture of the FALCON VOP
- to implement the different modules required for the FALCON VOP

## 1.2 Objective of T1.2

In Task 1.2, in cooperation with Task 1.1, the architecture team will define the overall architecture of the FALCON VOP together with all partners. This task also includes monitoring and control of the evolution of the platform architecture and its building blocks. The architecture team will ensure that the architecture including all its components fulfils technical quality standards regarding performance, reliability, and efficiency and ensure the application of suitable software development, management, and evaluation techniques throughout the lifecycle of the project. The team will define interfaces to ensure the ease of use between all partners' components.

## 1.3 Content of the document

Within this document, Chapter 2 defines the overall architecture of the FALCON Virtual Open Platform as well as the chosen technology. Then, a detailed overview of the defined FALCON VOP modules is given. For each FALCON VOP module, a description, preliminary interface definitions and mock-ups of the graphical user interfaces are presented. Subsequently, the addressed technical quality standards of the FALCON development process are described in detail. The deliverable closes with a summary and an outlook.

# 2 FALCON Virtual Open Platform Architecture

This section documents the specification of the FALCON VOP architecture, and explains the decisions taken by the architecture team regarding architecture model and platform composition.

## 2.1 FALCON VOP Architecture Model

In the initial phases of the architectural analysis activities, different models for application integration were taken into consideration, namely classical "monolithic" client-server architecture and a so-called micro-services-based architecture. In the first model, each module is implemented as a component whose life cycle is managed by some enterprise container, for instance in a Java Enterprise (JEE) architecture or as Java Spring beans in a lighter weight Java Spring implementation.

The second model taken into consideration, and which in the end was selected as a basis model for the FALCON VOP, consisted in an architecture where each module is implemented as a (REST-like or similar) independently deployable service.

In the last few years, a set of patterns and best practices for the design of software applications as collection of independently deployable services, particularly aimed at developing applications which adapt well to cloud-based architectures, have come to be known under the term "micro-services" architectural patterns. Though the term is used to mean many different things, basic concepts underlying these architectural patterns are based on well-known Service Oriented Architecture (SOA). While trying to overcome most of the problems that these kind of architectures and related frameworks have shown over the years, basically less coupled and lighter weight approaches (e.g. using protocols such as REST vs. SOAP etc.) are proposed. A detailed discussion on micro-services architectures and their differences with classical SOA architectures is not in the scope of this document. Please check the resources and bibliography for links to different sources of information on the web.

In general terms, components in the micro-services-based architecture are modelled as independently deployable processes which access each other remotely through well published interfaces and via some common protocol (e.g. the already mentioned HTTP-based REST messaging or asynchronous messaging through message brokers like RabbitMQ or similar ones). A FALCON specific specialization is shown in Figure 1.

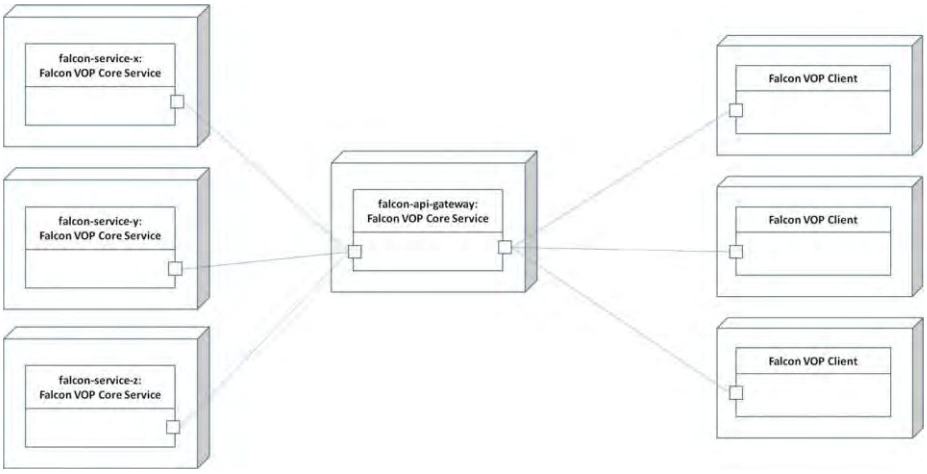


**Figure 1 FALCON modules as independently deployable services**

As we will see, the micro-services model provides many advantages but at the same time impose more strict requirements in implementing best practices and design patterns in order to be successful, and needs specific infrastructure services to replace container middleware functionality like component discovery and wiring, configuration etc., which is shown in Figure 2.

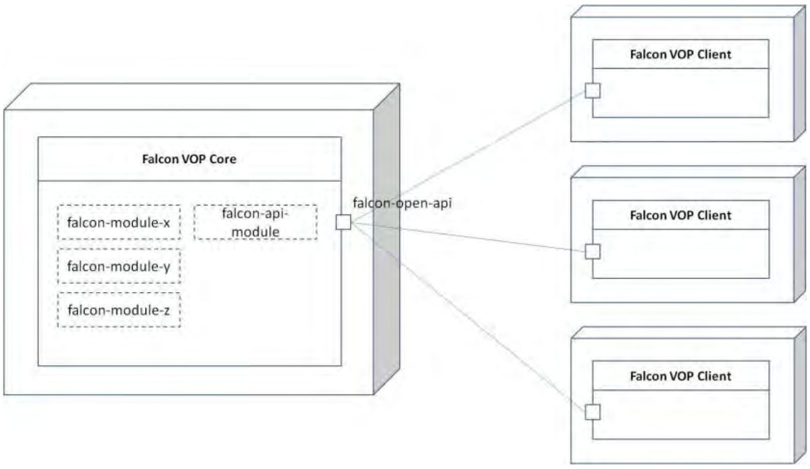


**Figure 2 FALCON modules as components in a client-server model**

To begin with, below a list of advantages vs. disadvantages of the micro-services architectural model (with respect to the monolithic client-server model) is summarized.

**Table 1 Advantages and disadvantages of micro-service architecture**

| Advantages | Disadvantages |
|---|---|
| Simplification of independent development by different, spatially separated teams (as is the case in the FALCON consortium). Each team is responsible for the maintenance of the code base of a single service. Communication between modules (and teams) occurs only through well-defined and well documented interfaces (APIs). | Higher complexity and architecture topology: Currently, one application is composed by a great number of independent "moving parts" that need to be coordinated and orchestrated. |
| Inherent low-coupling between services (but only if design "best practices" are followed: See the corresponding entry in the disadvantages column). This also favours service reuse in different applications. | If the model is not implemented correctly and best practices are not followed, this architectural can lead to higher instead of lower coupling between components.<br>(e.g. by passing as message payloads complex json-encoded data structures that directly reflect services' internal data structures, increasing instead of decreasing modules' coupling). |
| Greater scalability, availability and tolerance to failover: Under the right conditions, multiple equivalent instances of the same service can be ac- | The lack of a shared container imposes the need for more infrastructure ("middleware") services, which must be supplied separately. Service registration, discovery service, intelligent routing, configuration management, message brokers etc are |

| | |
|---|---|
| tivated and load-balanced by the platform framework in a transparent way for service clients and the service itself.<br>This is one of the points that favours micro-services-based architectures for the development of “cloud-ready” or “cloud-native” applications. | good examples. In a monolithic application all these functionalities are provided by the container, while at present specific micro-services supplying each of these capabilities must be deployed together with the application domain specific services.<br>See section 2.3.1 on supporting technologies for infrastructure services. |
| The model is a great enabler for continuous delivery, allowing frequent releases (new versions of a service, which don’t change its external interface, can be redeployed whilst keeping the rest of the system available and stable). | Significant operations overhead: Deployment of the set of developed services must be automatized due to the difficulty of managing a potentially large number of separate services installations, which may not be as straightforward as deploying a WAR in a container.<br>See section 2.3.1 on supporting technologies for platform deployment. |
| The model allows for “polyglot” implementation (when agreed on common communication mechanisms, e.g. JSON over REST), meaning there’s no need to implement all the services using the same technologies or programming languages.<br>This point is relatively of little interest in our case, since all partners seem to be comfortable with java/spring platform. | By offering a wider “surface of attack” security concerns become more critical. |
| Following from the previous point: Easier integration with external legacy systems, wrapped in a service layer that makes these systems act like just another service in the platform. | N/A |

In the above introduction, we have made reference to the fact that such an architecture poses greater challenges, and needs of greater rigour and coordination by module developers, in order to be successful. As mentioned, best practices and specific patterns must be followed, and a number of infrastructure services must be supplied to tackle the management of the complex application topology and to replace the services usually made available by the enterprise application container in the monolithic case.

Example of such services are the following:

- A service registry service, where business application services can register and discover each other.
- A configuration management service, handling externalized configuration for services.
- Services for dynamic routing, load balancing, etc. if required
- Others

Besides middleware and infrastructure services to deploy with the application, there's also the need for tools to handle deployment, configuration and orchestration of a potentially high number of independent services. Having to deploy and manage the configuration of many parts manually quickly becomes unmanageable without the support of the right tools and technologies, more so in case the different services are implemented through different technologies and language stacks (a possible scenario, though not the case for the FALCON project). To address this latest problem, a containerization solution based on docker and related container orchestration technologies (docker-compose) were selected for the FALCON VOP.

Containers acts as a sort of very lightweight virtual machines providing the minimum necessary environment to run the service. Moreover, a container-packaged application presents a standardized interface to the external environment, through which it can be configured and managed. The use of container orchestration and management tools adds to this the ability to configure and manage the whole set of containers composing the application as a whole.

## 2.2 FALCON VOP Final Draft Architecture

This section presents the current status of the FALCON VOP architecture. Due to the agile software engineering approach adopted by FALCON, this architecture is considered a final draft, in order to be able to accommodate future developments in the project. However, any changes to this final draft can be expected to be minimal and to only affect minor details.

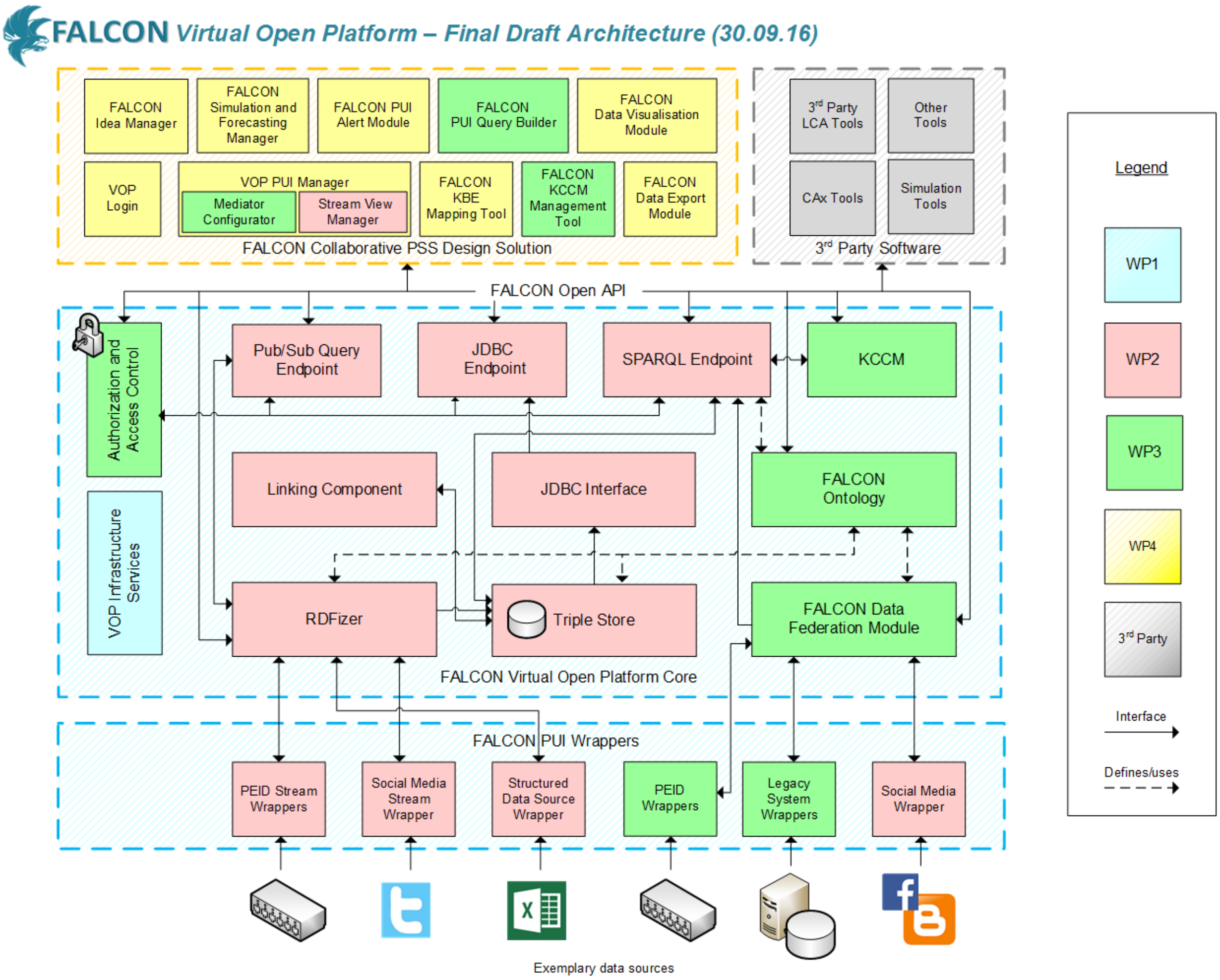


**Figure 3 FALCON VOP Final Draft Architecture**

As shown in Figure 3, the proposed FALCON VOP architecture includes four groups of components: the FALCON Virtual Open Platform Core, FALCON PUI Wrappers, FALCON Collaborative PSS Design Solution, and 3rd Party Software that are divided into three layers. In the bottom layer, FALCON PUI Wrappers wraps a diverse set of data sources e.g. sensor data and social media data to provide a solid ground for interoperability between different systems. In the middle layer, FALCON Virtual Open Platform Core provides core services to manage and integrate the knowledge and data from different sources, and make them public accessible through FALCON Open API. In the top layer, FALCON Collaborative PSS Design Solution and 3rd Party Software includes services to help the management and utilization of the interoperable knowledge and data for carrying out individual tasks depending on their own application scenarios.

## 2.3 FALCON VOP Components

As mentioned in the above section, the modules in the FALCON VOP architecture are grouped into four groups: FALCON PUI Wrappers, FALCON Virtual Open Platform Core, FALCON Collaborative PSS Design Solution, and 3rd Party Software. In this section, the groups and assigned micro-services/module are listed.. The grouping is based on micro-service functionality due to its readability.

### 2.3.1 FALCON Virtual Open Platform Core

FALCON Virtual Open Platform Core includes core modules for managing interoperable product usage information for the virtual open platform. In this section, the modules in the group FALCON Virtual Open Platform Core are presented and explained in detail.

#### 2.3.1.1 VOP Infrastructure Services

**FALCON Service Archetype**
(falcon-service-archetype)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This is not an individual VOP module in itself, but rather a Maven prototype used to allow the automatic generation of a skeleton Maven project for other VOP core modules pre-structured to integrate with the Spring Cloud environment defined for the FALCON platform.

The modules generated with this archetype will include configuration of references to the maven Nexus repository where the various FALCON platform artefacts will be uploaded and shared with the project partners, that will be made available through the URL http://repository.falcon-h2020.eu/nexus and other configuration of necessary maven plug-in(s) for connection and automatic deployment to the above configured nexus repository.

Moreover, the VOP Core services generated through this archetype will contain references in code and configuration to connect to the cloud infrastructure services distributed as part of the VOP infrastructure, such as the service registry and the cloud configuration service (see related module description forms)

**Interfaces:**

As mentioned, this module, not being a properly running VOP service, in this section in place of a functional interface is described how this module is used to generate the skeleton project for other modules. This is done through the invocation of a Maven command as illustrated in the following:

*Methods:*

1. *Generate*

```
mvn archetype:generate -B \
 -DarchetypeGroupId=eu.falcon-h2020.platform \
 -DarchetypeArtifactId=falcon-service-archetype \
 -DarchetypeVersion=0.0.1 \
 -DgroupId=eu.falcon-h2020.platform \
 -DartifactId=$artifactId \
 -Dversion=0.0.1 \
 -Dpackage=eu.falcon.services.$packageName \
 -DmoduleClassName=$moduleClassName
```

More detailed instructions on how to use the falcon-service-archetype have been distributed by SOFT among partners and summarized in an internal technical working document, FALCON_T1.3_Integration_Handbook.docx

**GUI Mock-ups:**

Not Applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Christian Melchiorre <christian.melchiorre@softeco.it> (SOFT): Nexus(X), GIT(X), Jenkins(X)

## FALCON VOP Service Registry (falcon-service-registry)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

Contrary to a classic enterprise client-server application environment, where usually a container manager takes care of handling modules instances and lifecycle, in a micro-services-based architecture (like the one selected for the FALCON VOP) a specific external service allowing modules to register itself so that its references can be retrieved by other modules in the system is required.

A service registry is basically a store of services references, their instances and locations. Service instances are registered with the service registry on start-up and deregistered on shutdown. The clients of the service and/or routers query the service registry to find the available instances of a service, either directly by name or by querying specific service properties.

A service registry is useful as it decouples service providers from consumers without the need for DNS, and can provide further additional functionalities, like enabling client-side load-balancing.

The implementation of the FALCON VOP service registry module (falcon-service-registry) is based on the Open Source Eureka service registry implementation.

**Interfaces:**

The functionality provided by this module is made available to other VOP modules not through the invocation of specific API interface functions but through specific configuration of the module code. The structure already provided within the falcon-service-archetype module is being leveraged.

*Functions:*

1. *Service Registration*

   This is the way a module can register itself on the service registry with a given name.

   As long as Spring Cloud and Eureka Core are on the classpath, any Spring Boot application annotated with `@EnableEurekaClient` (as happens for modules generated through the provided application) will try to contact a service registry and register itself with it. The address where the service is expected to run and the name the micro-service uses to register itself are provided, respectively, with two system properties, namely `eureka.client.serviceUrl.defaultZone`, and `spring.application.name`

2. *Service Retrieval*

The symmetric functionality provided by the service registry allows a client micro-service to retrieve references to another micro-service in order to invoke latter's functionality.

There are various ways to retrieve services references. The immediate one is exploiting the Spring-Cloud aware RestTemplate class, which takes the name of selected micro-service as part of the invokation URL. Note that the URI uses a service ID (the name with which the micro-service providing the invoked function registered with the registry), not an actual hostname.

```
String result = restTemplate.getForObject("http://falcon-service-
  xxx/func", String.class);
```

A second way to retrieve service instances is to explicitly query the service registry through the Spring Cloud provided class `DiscoveryClient`, as shown in the following code snippet:

```
@Autowired

private DiscoveryClient discoveryClient;

…

discoveryClient.getInstances("falcon-service-xxx")
```

(More details on the RestTemplate and DiscoveryClient classes' interface can be found on Spring Cloud online reference documentation[1])

[1] http://projects.spring.io/spring-cloud/spring-cloud.html

**GUI Mock-ups:**

This is a VOP Core infrastructure micro-service, hence does not provide a User Interface.

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Christian Melchiorre <christian.melchiorre@softeco.it> (SOFT): Nexus(X), GIT(X), Jenkins(X)

## FALCON VOP Externalized Configuration Service (falcon-service-config)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

A configuration service in a micro-services-based architecture provides server and client-side support for externalized configuration in a distributed system. With the configuration service, modules have a central place to manage external properties across all environments, instead of relying on settings hardcoded in property files distributed with the deployed module code, or having to manually provided

FALCON VOP Externalized Configuration service implementation is based on the Spring Cloud Config[1] implementation provided with the Spring Cloud platform.

The concepts on both client and server map identically to the Spring Environment and PropertySource abstractions, so they fit very well with Spring applications, but can be used with any application running in any language. As an application moves through the deployment pipeline from dev to test and into production, you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate. The default implementation of the server storage backend uses git to easily support labelled versions of configuration environments, as well as being accessible to a wide range of tooling for managing the content. It is easy to add alternative implementations and plug them in with Spring configuration.

[1] http://projects.spring.io/spring-cloud/spring-cloud.html#_spring_cloud_config

**Interfaces:**

The functionality provided by this module is made available to other VOP modules not through the invocation of specific API interface functions but through specific configuration of the module code, also leveraging the structure already provided with the falcon-service-archetype module.

*Functions:*

3. *Connection to the config service*

   FALCON VOP modules generated via the provided archetype already have the necessary configuration to connect to the running VOP configuration service. The connection of the client module to the config service is configured by setting two specific environment properties `cloud.config.discovery.enabled` and `cloud.config.discovery.serviceId`

4. *Retrieval of Configuration properties*
   As mentioned, Spring Cloud config concepts maps to the Spring environment and `PropertySource` abstractions, so they fit very well with Spring applications

such as the FALCON modules. If the connection to the config service is configured at startup (see previous point), environment properties values can be accessed by the modules' code by binding them to variable like in the following example:

```
@Value("${falcon.service.sample.property:default}")
String message;
```

Spring Cloud Config also provides mechanisms to dynamically refresh property configuration information in case this is changed at runtime.

5. *Retrieval of Configuration properties (REST interface)*

   Besides the mechanisms described in the previous point, the configuration service also provide a REST-based interface to access configuration information. Such information can be retrieved by invoking URLs such as

   ```
   http://localhost:8888//{module name}/{profile}[/{label}]
   ```

   Returning a JSON representation of the set of properties associated to the given module. It is likely this way of accessing properties will not be used by VOP modules, but can be a useful tool for monitoring or debugging instead.

(More details on the Spring Cloud service implementation can be found on Spring Cloud online reference documentation[1])

[1] http://projects.spring.io/spring-cloud/spring-cloud.html#_spring_cloud_config

**GUI Mock-ups:**

This is a VOP Core infrastructure micro-service, hence does not provide a User Interface.

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Christian Melchiorre <christian.melchiorre@softeco.it> (SOFT): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.1.2 Authorization and Access Control

### Authentication and Access Control Module
(falcon-service-auth&accesscontrol)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module will be developed in the context of Task T3.5 of WP3 – Data Security and Privacy. Since the related task is not yet active, but will start in M22, the following module description should be intended as general approach to authentication and authorization module specification, and will be subject to change.

The FALCON Authentication and Access Control module implements the identity management and access control functionalities.

The identity management grants FALCON users and associate them with corresponding roles. Access control verifies compliance with security rules defined for roles, activities and resources within the ontology in use.

The module can be configured and enabled to collaborate with legacy resource servers (e.g. using standard technologies like OAuth2) and will reuse architectural components envisaged by currently widely used specifications and protocols (e.g. RFC2904, XACML, SAML, etc.) to improve interoperability, reuse of components and integratability.

The FALCON login page uses the authentication manager. Access control will be exploited by FALCON VOP micro-services before providing a user with the requested functionality/resource.

**Interfaces:**

*Interfaces specification will be provided within T3.5 activities.*

**GUI Mock-ups:**

This is a VOP Core infrastructure micro-service, hence does not provide a User Interface.

- Rita Pasini <rita.pasini@softeco.it> (SOFT): Nexus(X), GIT(X), Jenkins(X)
- Christian Melchiorre <christian.melchiorre@softeco.it> (SOFT): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.1.3 FALCON Data Federation Module

## Data Federation Module (falcon-service-semanticfederationmodule)

**module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

It offers a layer for semantic, virtual interoperability and integration specifically of item-level product lifecycle data. It provides semantic interoperability for different kinds of common data sources like social media, databases and file based repositories. It introduces a layer of semantics on top of existing syntactic data structure descriptions to avoid semantic integration conflicts and allows a scalable, efficient and comfortable interoperability of product data across all of the stakeholders and IT systems involved in VOP platform.
This module communicates with the *Linked Sensor Data Access Mechanism* to read and write ontology triples as well as to read ontology triples from the *Triple store*. Moreover, this module will be accessible by the web application *Semantic Mediator Conf Administrator.*

**Interfaces:**

*Methods:*

1. *Query*
   a. **Input**: JSON string including a SPARQL query and a namespace
   b. **Output**: String representing the query result as ontology
2. getConnectedDataSources
   a. **Input**: Nothing
   b. **Output**: XML String describing the connected Data Sources
3. getOntologyConcepts
   a. **Input**: Nothing
   b. **Output**: XML String describing the covered ontology concepts by the connected data sources

**GUI Mock-ups:**

This module is a micro-service and offers only a REST interface. The corresponding GUI mock-ups to configure this micro-service is located in VOP PUI Manager

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Marco Franke <fma@biba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Quan Deng <dqu@biba.uni-bremen.de> (BIBA) : Nexus(X), GIT(X), Jenkins(X)

#### 2.3.1.4 FALCON Ontology

**FALCON Ontology**
(No corresponding micro-service)

**Module type:**

☐ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☒ Other

**Description of Main Responsibilities of this Module:**

It plays a pivotal role, offering a semantic model for the PSS knowledge domain. It will serve as a common reference model for the annotation and description of the entire Product-Service Life Cycle. In addition, it will provide a unified and all spanning semantic model covering domain knowledge of product-services to build a bridge between end users of a to-be-designed software platform and the software platform implementation. It involves all of the concepts, which represent the basic entities of the project and provides a linked data integration framework.

It consists of two levels which are an upper ontology and specific ontologies reflecting the functionalities of VOP required by each business partners. Meanwhile, it represents a base layer for the semantic reasoner.

**Interfaces:**

The FALCON ontology is a model and therefore, it has no software interface

**GUI Mock-ups:**

The FALCON ontology is a model and therefore, it has no GUI mock-ups

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Sangje Cho <sangje.cho@epfl.ch> (EPFL): Nexus(-), GIT(-), Jenkins(-)

### 2.3.1.5 KCCM

**Knowledge Consolidation & Cross sectoral Management Module** (falcon-service-KCCM)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The Knowledge Consolidation & Cross sectoral Management Module is foreseen to provide features (functions) for transformation of accessed data. These features can be shared between VOP modules, thus allowing a kind of service orchestration in preparation of front-end modules.

The following features are planned:

- features referring to statistical/mathematical operations enabling to aggregate information from data sets
- features to transform state or appearance of information such as one currency into another, or percentage of a single item to a set of items
- features to conduct automated filter operations such as derived queries: (data of “today” => “today” will be transformed in actual date)
- features relating to semantical neighborhood of entities, exploring the benefits of the contextual relevance. (e.g. cashing of “tomorrow’s data”)
- features relating to semantical neighborhood of entities, relating information across PLM boundaries (e.g. date of purchase => guaranty Yes/No => maintenance strategy => different data sets)

**Samples**:

- ***Statistics:*** Calculation of Arithmetic mean of sensor values;
- ***Transformation:*** Adding a column with Fahrenheit temperatures, which are calculated based upon an existing column with Celsius temperatures;
- ***Math:*** Add a column with weight values, which are calculated based upon a volume column,
- ***Filters:*** Find amount of “warm colours” (if warm is defined with red, yellow; beige …);

**Interfaces:**

*Methods:*

a. **Input**: The definition of input methods is currently under preparation, but the methods will be named in reference to the mathematical, statistical or logical operation, e.*g.: Get_average_for_period(x,y,z);*

b. **Output**: A JSON Object including inputs and the output arrays

**GUI Mock-ups:**

The KCCM is foreseen to become a backend module providing bundled features without an own Graphical User Interface. The features will be encapsulated as services and provided to the system accessible with the FALCON protocols.

However a User Interface for KCCM is provided by KCCM Management Tool (ref. to section 2.2.3.4)

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Patrick Klein <klp@bba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Marco Franke <fma@biba.uni-bremen.de>

### 2.3.1.6 RDFizer

**RDFizer**
(falcon-service-rdfizer

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The RDFizer Component is responsible for transforming data into information in RDF format. As a secondary function, it also delegates the RDF triples produced by the Semantic Enrichment of data extracted from Social Networks to the RDF storage, which is the FALCON Triple Store. It is part of the RDFizer specific development activities (see for more details D2.1).

**Interfaces:**

*Methods:*

1. ***RegisterStream***
   a. **Input**: A JSON containing the Stream URI
   b. **Output**: A JSON Object with Success/Failure Enumeration Code
2. ***CreateFacetedView***
   a. **Input**: A JSON containing the Stream URI and the facets that have to be rendered
   b. **Output**: A JSON Object with Success/Failure Enumeration Code
3. ***SaveView***
   a. **Input**: A JSON containing the state of the current view
   b. **Output**: A JSON Object with Success/Failure Enumeration Code
4. ***DisposeStreamView***
   a. **Input**: A JSON containing the Stream URI that has to be disposed
   b. **Output**: A JSON Object with Success/Failure Enumeration Code

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.1.7 Linking Component

**Linking Component**
(falcon-linking-module)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The scope of the linking component is to align two (or more) semantic concepts that derive from two distinct ontologies using a relationship that is declared either in an upper ontology or in one of the two ontologies. This process ends up in a linked data dataset

**Interfaces:**

*Methods:*

2. *RegisterOntology*
   a. **Input**: A JSON Object that included an Ontology along with the Ontology format
   b. **Output**: A JSON Object with Success/Failure Enumeration Code
3. *AddMapping*
   a. **Input**: A JSON Object that includes two semantic concepts from Ontologies that are already registered
   b. **Output**: A JSON Object with Success/Failure Enumeration Code
4. *getMappigns*
   a. **Input**: A JSON Object that includes the request
   b. **Output**: A JSON Object that contains a Success/Failure Code along with an array of the mappings

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.1.8 Triple Store

## Triple Store
(falcon-triplestore-component)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module is responsible for the persistency of the RDF triples for knowledge representation in the frame of FALCON. Also, the persistency engine will be used for knowledge inferencing since the engine will be able to respond to semantic queries. Apache Fuseki will be used as a substrate technology.

**Interfaces:**

*Methods:*

1. *addTriple*
   a. **Input**: A JSON Object that contains an RDF triple along with the format-metadata of the triple
   b. **Output**: A JSON Object that contains a Success/Failure Code of the insertion
2. *executeSPARQLQuery*
   a. **Input**: A JSON Object that contains a SPARQL-properly formated query
   b. **Output**: A JSON Object that contains a Success/Failure Code along with a list of the RDF results

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.1.9 JDBC Interface/JDBC Endpoint

## JDBC Endpoint (JDBC_Interface)
(falcon-jdbc-module)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module exposes basic relational database interface on top of any relational database that is registered on the platform. It allows dynamic on boarding and graceful deprovision of any relational database. The JDBC Endpoint is a concrete implementation to the JDBC_Interface which will be a utilisation of the existing technology Apache FUSEKI (see for more details D2.1).

**Interfaces:**

*Methods:*

5. ***Register Relational Database***
   a. **Input**: A JSON file containing the Database Registration Context
   b. **Output**: A JSON Object with Success/Failure Enumeration Code
6. ***Remove Relational Database***
   a. **Input**: A JSON file containing the Database Identifier assigned during registration
   b. **Output**: A JSON Object with Success/Failure Enumeration Code

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

Copyright ©FALCON Consortium 2015-2017

#### 2.3.1.10 Pub/sub Query Endpoint

## Pub/sub Query Endpoint
(falcon-pubsub-component)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This component will be used in order to facilitate all the asynchronous communication within the project. It will act as a queue and will allow subscribers to send and receive messages through predefined topics. ActiveMQ will be used as a substrate technology (see for more details D2.1).

**Interfaces:**

*Methods:*

1. *CreateTopic*
   a. **Input**: A JSON Object that contains a new TOPIC that will be supported by the queue
   b. **Output**: A JSON Object that contains a Success/Failure Code
2. *RegisterToTopic*
   a. **Input**: A JSON Object that contains the desired TOPIC that the invoker wishes to register along with the invoker metadata
   b. **Output**: A JSON Object that contains a Success/Failure Code
3. *UnRegisterFromTopic*
   a. **Input**: A JSON Object that contains the desired TOPIC that the invoker wishes to unsubsribe along with the invoker metadata
   b. **Output**: A JSON Object that contains a Success/Failure Code
4. *PublishToTopic*
   a. **Input**: A JSON Object that contains the message that will be put on the queue
   b. **Output**: A JSON Object that contains a Success/Failure Code

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.1.11 SPARQL Endpoint

## SPARQL Endpoint
(falcon-service-sparql-module)

**Module type:**

☒ VOP Core REST service

☐ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The specific module offers augmented semantic query capabilities on top of the triple store. As already noted, the triple store already offers SPARQL querying capabilities; however the existing module offers reasoning extensions. This module will be a utilisation of the existing technology Apache FUSEKI (see for more details D2.1).

**Interfaces:**

*Methods:*

1. *ExecuteReasoningQuery*
   a. **Input**: A JSON containing the query in a SQRQL-conformant format
   b. **Output**: A JSON Object with Success/Failure Enumeration Code

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

### 2.3.2 FALCON PUI Wrappers

FALCON PUI Wrappers is a collection of wrappers whereby a wrapper aims to extract information of a specific kind of data source. The objective is to achieve the interoperability in the overall FALCON VOP. The contained wrappers are presented and explained in detail.

#### 2.3.2.1 PEID Wrappers

**PEID Wrapper**
(Part of falcon-service-semanticfederationmodule)

**Module type:**

☐ VOP Core REST service

☐ Application/UI level tool

☒ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The PUI Wrapper collects data from sensors, which are provided as not streaming data and maps it onto the FALCON Ontology. Thus, it creates an Abox for new available data and forward this Abox to the overall Data Federation Module which enables the commit to the Triple Store and achieve therefore, the digital continuity for all different FALCON modules. In doing so, the existing autonomy of sensors will not be violated and changes in sensors can be compensated.

**Interfaces:**

*Methods:*

1. *Initialize*
   - *a.* **Input**: Java InputStream of the property file
   - *b.* **Output**: void
2. *query*
   - *a.* **Input**: Java Object containing current Abox and requested concepts and properties
   - *b.* **Output**: Abox
3. insert
   - *a.* **Input**: Abox
   - *b.* **Output**: void

**GUI Mock-ups:**

This module is a micro-service and offers only a REST interface. The corresponding GUI mock-ups to configure this micro-service is located in VOP PUI Manager

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Marco Franke <fma@biba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)

- Quan Deng <dqu@biba.uni-bremen.de> (BIBA) : Nexus(X), GIT(X), Jenkins(X)

#### 2.3.2.2 PEID Stream Wrappers

**PEID Stream Wrappers**
(falcon-streamwrapper-module)

**Module type:**

☐ VOP Core REST service

☐ Application/UI level tool

☒ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module will be used in order to semantically uplift an existing sensor stream to a semantic stream which is aligned with the FALCON Ontology. The result of the 'uplifting' procedure will be propagated to the Triple Store that was mentioned above. It is part of the DataRefinement specific development activities (see for more details D2.1).

**Interfaces:**

*Methods:*

1. *RegisterStream*
   a. **Input**: A JSON Object that registers a sensor-binarystream which will be processed
   b. **Output**: A JSON Object that contains a Success/Failure Code
2. *ProvideMapping*
   a. **Input**: A JSON Object that provides a mapping which will transform the binary stream to proper CSV file format
   b. **Output**: A JSON Object that contains a Success/Failure Code
3. *getBufferedOutput*
   a. **Input**: A JSON Object that contains transformed CSV payload
   b. **Output**: A JSON Object that contains a Success/Failure Code

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.2.3 Social Media Wrappers

| **Social Media Wrapper**<br>(Part of falcon-service-semanticfederationmodule) |
|---|
| **Module type:**<br>☐ VOP Core REST service<br>☐ Application/UI level tool<br>☒ PUI Wrapper<br>☐ Other |
| **Description of Main Responsibilities of this Module:**<br>The SocialMedia Wrapper collects data from **social media**, which are provided as not streaming data and maps it onto the FALCON Ontology. Thus, it creates an Abox for new available data and store this data in a Triple Store. Incoming information request will be replied on basis of the already extracted knowledge and will be forwarded as Abox to the overall Data Federation Module. Then, it enables the commit to the FALCON Triple Store and achieves the digital continuity for all different FALCON modules. In so doing, the existing autonomy of social media will not be violated and changes can be compensated in social media. |
| **Interfaces:**<br>*Methods:*<br>1. *Initialize*<br>a. **Input**: Java InputStream of the property file<br>b. **Output**: void<br>2. *query*<br>a. **Input**: Java Object containing current Abox and requested concepts and properties<br>**b. Output**: Abox |
| **GUI Mock-ups:**<br>This module is a micro-service and offers only a REST interface. The corresponding GUI mock-ups to configure this micro-service is located in VOP PUI Manager |
| **Responsible Partner(s)/Developer(s):**<br>*Developer:*<br>• Marco Franke <fma@biba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)<br>• Quan Deng <dqu@biba.uni-bremen.de> (BIBA) : Nexus(X), GIT(X), Jenkins(X) |

#### 2.3.2.4 Social Media Stream Wrappers

## Social Media Stream Wrappers
(TBD)

**Module type:**

☐ VOP Core REST service

☐ Application/UI level tool

☒ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module is responsible for the usage of public information that exists in social media in order to perform NLP analysis on top of it. The output of this analysis is the generation of RDF triples that will be propagated to the Triple store. It is part of the DataRefinement specific development activities (see for more details D2.1).

**Interfaces:**

*Methods:*

1. *RegisterSocialMediaSource*
   a. **Input**: A JSON Object that contains information about a social media source which will be processed
   b. **Output**: A JSON Object that contains a Success/Failure Code
2. *AddProcessConfiguration*
   a. **Input**: A JSON Object that contains NLP configuration (e.g. keywords) that will be used to perform sophisticated analysis
   b. **Output**: A JSON Object that contains a Success/Failure Code
3. *getProcessedOutput*
   a. **Input**: A JSON Object that contains the output of an analysis in a CSV format
   b. **Output**: A JSON Object that contains a Success/Failure Code

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

#### 2.3.2.5 Legacy System Wrappers

**Legacy System Wrappers**
(Part of falcon-service-semanticfederationmodule)

**Module type:**

☐ VOP Core REST service

☐ Application/UI level tool

☒ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The Legacy System Wrapper collects data from legacy systems, which are provided as not streaming data and maps it onto the FALCON Ontology. Thus, it creates an Abox for new available data and forward the resulting Abox to the Data Federation Module. It allows committing the aggregated version of the legacy system information to the Triple Store and achieves the digital continuity for all different FALCON modules. In so doing, the existing autonomy of legacy systems will not be violated and changes in legacy systems can be compensated. The Legacy System Wrapper represents not only one kind of wrapper, but rather selects different kinds of wrappers, which focus on the interoperability of legacy systems.

**Interfaces:**

*Methods:*

1. *Initialize*
   a. **Input**: Java InputStream of the property file
   b. **Output**: void
2. *query*
   a. **Input**: Java Object containing current Abox and requested concepts and properties
   b. **Output**: Abox
3. *insert*
   a. **Input**: Abox
   **b.** **Output**: void

**GUI Mock-ups:**

This module is a micro-service and offers only a REST interface. The corresponding GUI mock-ups to configure this micro-service is located in VOP PUI Manager

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Marco Franke <fma@biba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Quan Deng <dqu@biba.uni-bremen.de> (BIBA) : Nexus(X), GIT(X), Jenkins(X)

#### 2.3.2.6 Structured Data Source Wrappers

## Structured Data Source Wrappers (falcon-service-structured-data-source-wrappers)

**Module type:**

☐ VOP Core REST service

☐ Application/UI level tool

☒ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module will semantically uplift an existing structured data source to a semantic stream, which is aligned with the FALCON Ontology. The result of the 'uplifting' procedure will be propagated to the Triple Store that was mentioned above. It is part of the DataRefinement specific development activities (see for more details D2.1).

**Interfaces:**

*Methods:*

1. *RegisterDataSource*
    a. **Input**: A JSON Object that registers a structured datasource which will be processed
    b. **Output**: A JSON Object that contains a Success/Failure Code
2. *ProvideMapping*
    a. **Input**: A JSON Object that provides a mapping which will transform the elements of the datasource to proper CSV input
    b. **Output**: A JSON Object that contains a Success/Failure Code
3. *getStructuredOutput*
    a. **Input**: A JSON Object that contains transformed CSV output
    b. **Output**: A JSON Object that contains a Success/Failure Code

**GUI Mock-ups:**

Not applicable

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X)

### 2.3.3 FALCON Collaborative PSS Design Solution

FALCON Collaborative PSS Design Solution includes a set of modules to help the management and utilization of the interoperable knowledge and data for carrying out individual tasks. In this section, the modules in the group FALCON Collaborative PSS Design Solution are presented and explained in detail.

#### 2.3.3.1 Virtual Open Platform Login Widget

| **Login Service**<br>(TBD) |
|---|
| **Module type:**<br>☐ VOP Core REST service<br>☒ Application/UI level tool<br>☐ PUI Wrapper<br>☐ Other |
| **Description of Main Responsibilities of this Module:**<br>This module allows the access to the Virtual Open Platform. It communicates with the authorization and access control service. Different users will have different degree of access in the system. This authorization will be managed by the authorization and access control service. |
| **Interfaces:**<br>*Methods:*<br>1. *getUserId*<br>a. **Input**: JSON user credentials<br>b. **Output**: - |
| **GUI Mock-ups:** |

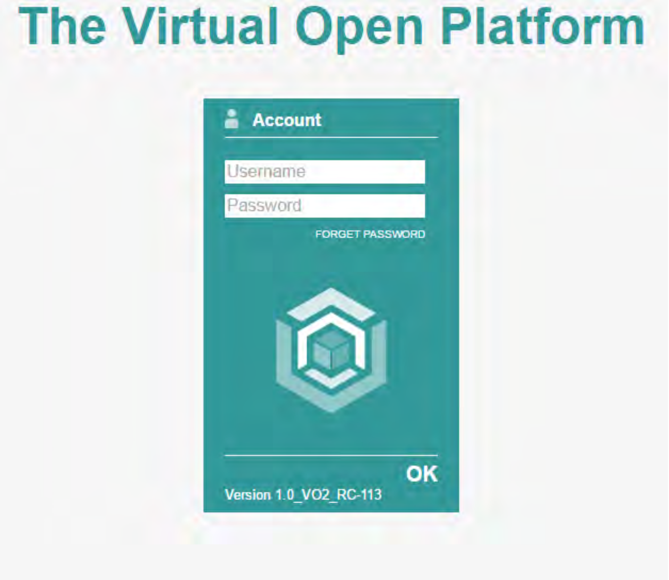


**Figure 4 Login View of FALCON VOP**

Reaching the address http://demo.holonix.it/vofalcon/index.zul the user with the access right can accede at the system.

**<u>Responsible Partner(s)/Developer(s):</u>**

*Developer:*

- Weian Xu <weian.xu@holonix.it> (HOLONIX): Nexus(-), GIT(X), Jenkins(-)

#### 2.3.3.2 Collaborative Environment

### Collaborative Environment
(TBD)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The Collaborative Environment is a dynamic system that allows improving collaboration and information sharing between two or more actors in a physical context (a meeting room) and/or in a virtual way (between two or several meeting rooms or single computers). The system allows the creation of virtual rooms with dedicated topics. Each of these rooms can have specific participants with different access rights. Within each room, many applications can be embedded to provide a wide set of services. In addition, web-based tools provided by third parties can be embedded into rooms set up using the Collaborative Environment. It supports concurrent and asynchronous work in the same virtual rooms, and collaboration with team members both in the same physical place and at a distance. Its key feature is the implementation of visual management which is a clear, simple and effective way of organizing and presenting work. Its user interface is designed to be user-friendly and to clearly organize and visualize many documents, applications, tools and objects in the same room. Moreover, users can customize how tools and applications are visualized in their individual rooms. This includes removing objects, adding new tools and applications, changing the position of available tools by simple drag and drop and inviting new participants to join the room. Moreover, the Collaborative Environment is able to interact with 3rd party System using open APIs. The Idea Manager, Data Analysis module and other FALCON web-based tools can be used within it.

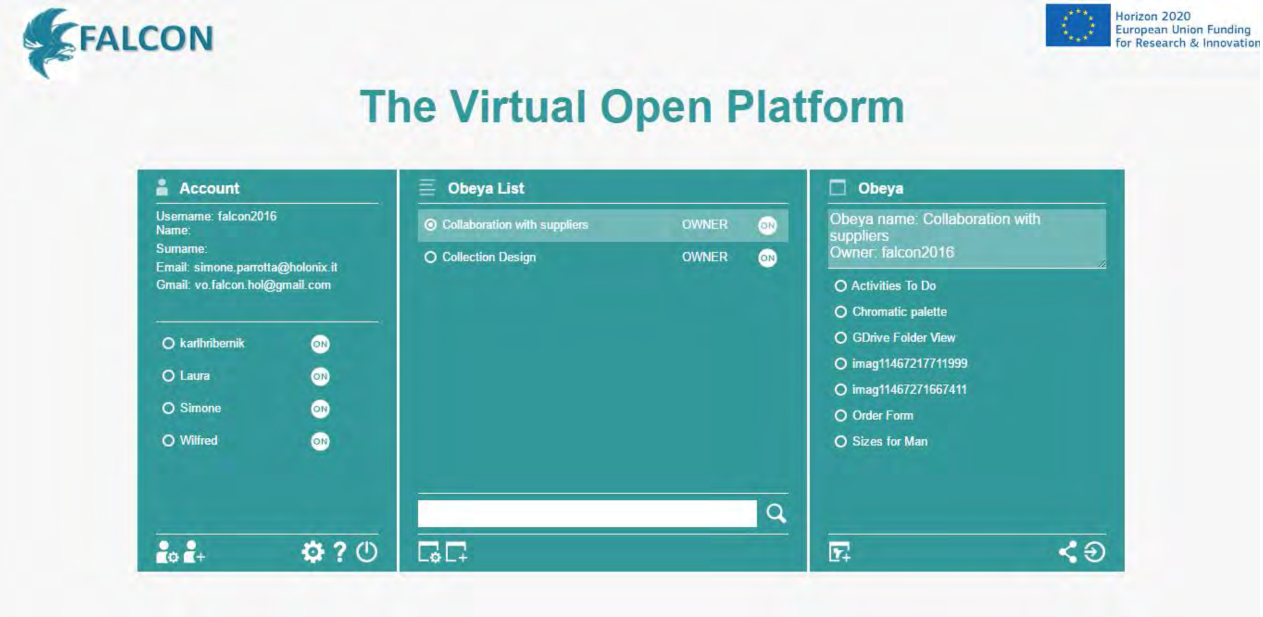
**Interfaces:**

*Methods:*

1. *PostCreateNewWidget*
   a. **Input**: JSON specifies widget parameters
   b. **Output**: -
2. *PostCreateNewObeya*
   a. **Input**: Obeya name as parameter
   b. **Output**: -
3. *PostObeya-WidgetAssociation*
   a. **Input**: Obeya Name and Widget Name as parameters
   b. **Output**: -

**GUI Mock-ups:**

In the central column of the main page, the Collaborative Environment shows the list of dashboards on which the user can accede. Selecting one dashboard the user can visualize the list of widgets available in the selected room. Moreover, it's possible to create a new dashboard, modify the contents of the already existing one (including adding or removing widgets), as well as authorizing or denying access to specific users.



Username: falcon2016
Name:
Surname:
Email: simone.parrotta@holonix.it
Gmail: vo.falcon.hol@gmail.com

○ karlhribernik
○ Laura
○ Simone
○ Wilfred

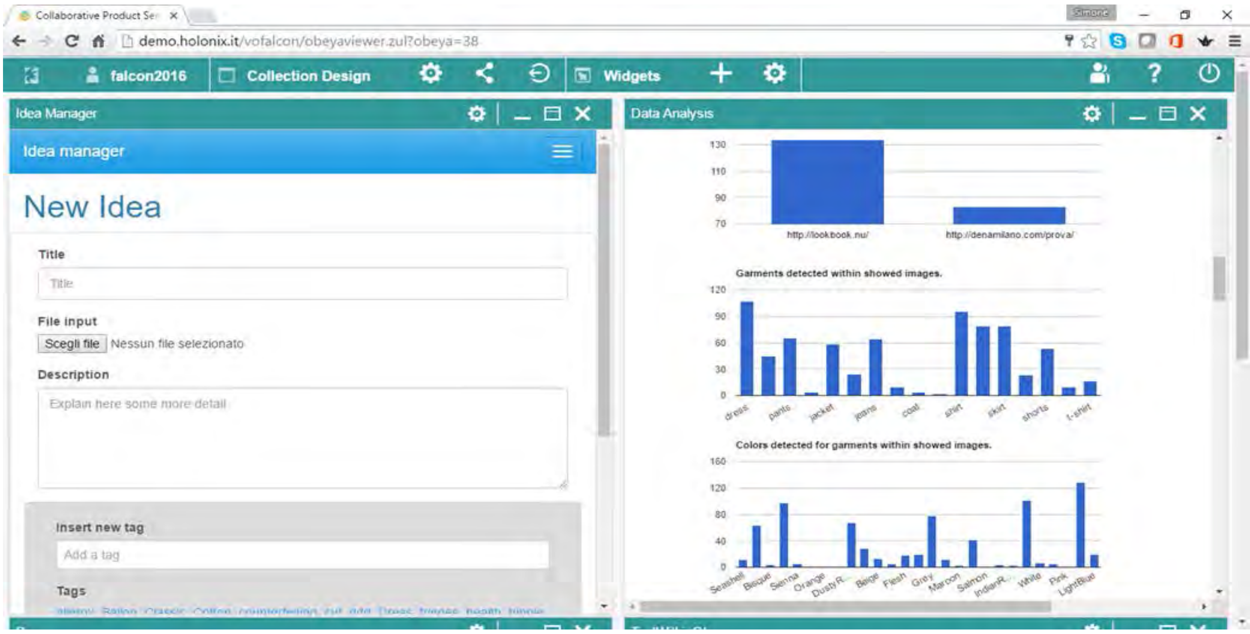
◉ Collaboration with suppliers OWNER
○ Collection Design OWNER

Obeya name: Collaboration with suppliers
Owner: falcon2016

○ Activities To Do
○ Chromatic palette
○ GDrive Folder View
○ imag11467217711999
○ imag11467271667411
○ Order Form
○ Sizes for Man

**Figure 5 Dashboard of the FALCON VOP**

Once the user accedes one specific dashboard, he/she will be able to interact with tools placed in that room, modify the dashboard visualization structure (e.g. number of columns, widget visualized, dimensions of each tool), and see the people connected to the room during the same time.



New Idea

Title

Title

File input

Scegli file Nessun file selezionato

Description

Explain here some more detail

Insert new tag

Add a tag

Tags

**Figure 6 Example widgets of FALCON VOP**

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Weian Xu <weian.xu@holonix.it> (HOLONIX): Nexus(-), GIT(X), Jenkins(-)

### 2.3.3.3 VOP PUI Manager

**VOP PUI Manager**
(falcon-ui-puimanager)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The objective of the PUI Manager is to enable the configuration of a FALCON module, which stores its settings within the FALCON VOP. This module implements the basic functionality to create and load configurations of PUI wrappers. The specific configuration wizards will be implemented in other modules (2.3.3.3.X). The objective of these wizards is to support the user in configuring and maintaining the tools.

**Interfaces:**

*Methods:*

1. *updateConfiguration*
   a. **Input**: Java object holding the settings
   b. **Output**: The information whether it could be updated

**GUI Mock-ups:**

This module has no graphical user interface

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Marco Franke <fma@biba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Quan Deng <dqu@biba.uni-bremen.de> (BIBA) : Nexus(X), GIT(X), Jenkins(X)

### 2.3.3.4 Mediator Manager

| **Mediator Manager**<br>(falcon-ui-mediatormanager) |
|---|
| **Module type:**<br>☐ VOP Core REST service<br>☒ Application/UI level tool<br>☐ PUI Wrapper<br>☐ Other |
| **Description of Main Responsibilities of this Module:**<br>This module is a specialization of the general VOP GUI manager and focusses on the configuration of the Data Federation Module, PEID Wrapper, Social Media Wrappers and Legacy System Wrapper by the user. For that purpose, module specific widget will be developed and offered in the overall VOP platform |
| **Interfaces:**<br>*Methods:*<br>1. *updateConfiguration*<br>a. **Input**: Java object holding the settings<br>b. **Output**: The information whether it could be updated |
| **GUI Mock-ups:**<br>The following screenshots show how a wrapper configuration could be added. For that purpose, the user first selects the type of data source to be configured. Each kind of data source requires different amount of information. Then, all data source specific information by the user will be collected with a wizard. An example of the wizard during the configuration of a social media related data source is shown in the second screenshot. |

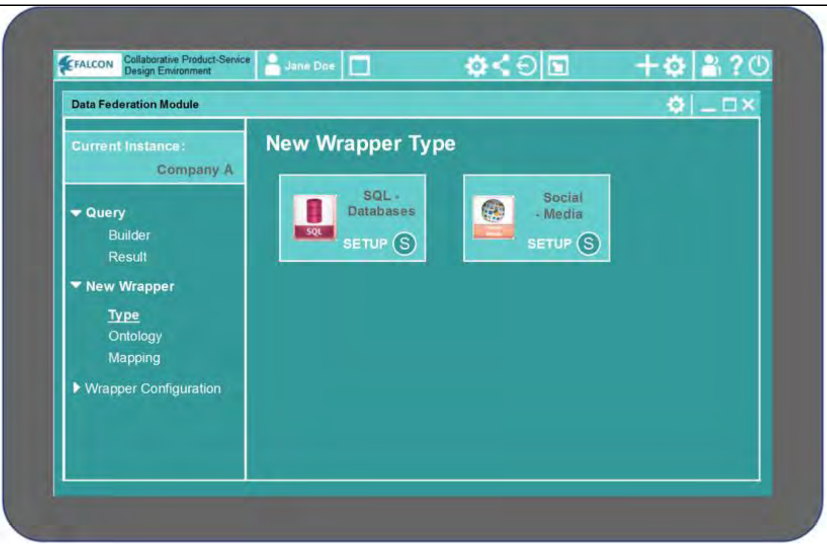
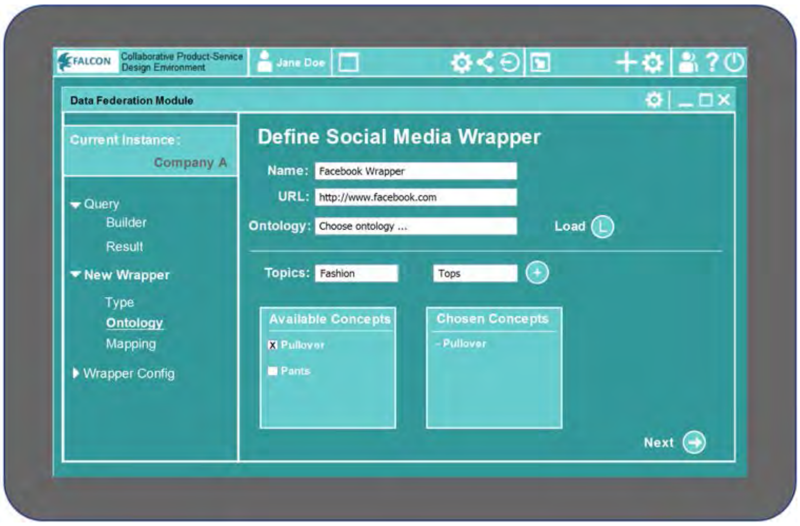


**Figure 7 Widget for wrapper selection**



Name: Facebook Wrapper
URL: http://www.facebook.com
Ontology: Choose ontology ...
Topics: Fashion Tops

Available Concepts: ☒ Pullover □ Pants
Chosen Concepts: - Pullover

**Figure 8 Configuration widget of social media wrapper**

**<u>Responsible Partner(s)/Developer(s):</u>**

*Developer:*

- Marco Franke <fma@biba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Quan Deng <dqu@biba.uni-bremen.de> (BIBA) : Nexus(X), GIT(X), Jenkins(X)

#### 2.3.3.5 Stream View Manager

| **Stream View Manager**<br>(falcon-stream-view-manager) |
|---|
| **Module type:**<br>☐ VOP Core REST service<br>☒ Application/UI level tool<br>☐ PUI Wrapper<br>☐ Other |
| **Description of Main Responsibilities of this Module:**<br>The stream manager is responsible to manage the streaming sources that are registered at FALCON VOP. It is the module that along with the RDFizer manage holistically the streaming sources and enables the extraction and storage of information in the Triple Store. It is part of the FALCON Daemon specific development activities (see for more details D2.1). |
| **Interfaces:**<br>The Stream View Manager uses the interfaces from the RDFizer to configure the stream data sources. |
| **GUI Mock-ups:**<br>Not applicable |
| **Responsible Partner(s)/Developer(s):**<br>*Developer:*<br>• Panagiotis Gouvas <pgouvas@ubitech.eu> (UBITECH): Nexus(X), GIT(X), Jenkins(X) |

### 2.3.3.6 FALCON KBE Mapping Tool

**Mapping tool for PUI related Design**
(KBE PUI Mapping service)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

Several proprietary KBE frameworks exist on the market, which enable an easy and fast creation of design variants (CAD models) based on a variety of input parameter sets. But due to lack of a framework independent standard itself, a direct linkage of Lifecycle related information such as usage parameters is not feasible.
With this background the FALCON team is developing and promoting KbeML, a modelling language derived from SysML (as an extension), which will allow a formal mapping of usage information to the product model dependencies, if the KbeML elements are enhanced accordingly.
This module defines a mapping tool to link PUI to product development knowledge. The tool enables to create linkages between parameters/values of commercial tools such as CAD and a KbeML model, which has been designed in a modelling framework (and stored as XMI file). The KbeML model itself provides the capabilities to handle the usage information. A selection of appropriate bearings may serve as a sample: The bearing itself is typically chosen by some geometry related parameters and the bearing-constant. The bearing constant can be expressed by an equation having an “environmental temperature” as one of several input parameters. In line with FALCON objectives users may want to map this “environmental temperature” to a data set of a sensor (“Temp-Sensor_01”). The tool allows a manual mapping between both types of parameters. The tool relies on XML/XMI files, to enable the data exchange.

**Interfaces:**

*Methods:*

*n.a. => The module will be based upon handling of XML-files*

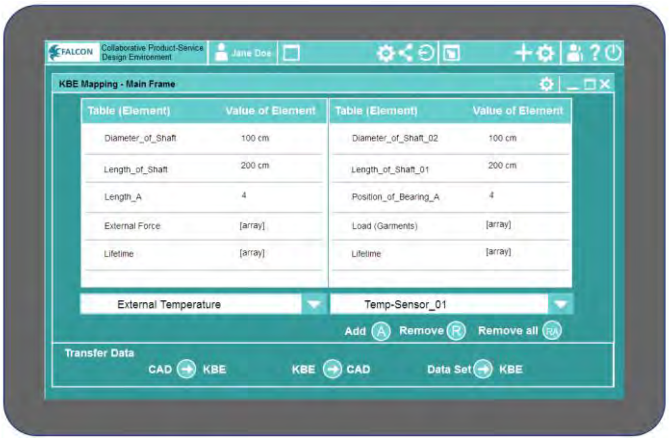
**GUI Mock-ups:**

The mapping tool will allow a manual assignment between the development related parameters and the PUI information. As illustrated below a parameter such as the “external temperature” can mapped to a specific sensor called “Temp-Sensor_01” and the values can be exchanged based upon this definition.

Using KbeML the “external temperature” parameter can become part of equations and rules (e.g. to reflect a thermal expansion of parts/materials). The mapping itself will allow the automated transfer of values. For this purpose, the data transfer can be controlled by three options: 1. CAD data will be transferred into KbeML models; 2. KbeML

Copyright ©FALCON Consortium 2015-2017

data will be transferred into CAD files; 3 Product usage data will be transferred into KbeML models.



| Table (Element) | Value of Element | Table (Element) | Value of Element |
|---|---|---|---|
| Diameter_of_Shaft | 100 cm | Diameter_of_Shaft_02 | 100 cm |
| Length_of_Shaft | 200 cm | Length_of_Shaft_01 | 200 cm |
| Length_A | 4 | Position_of_Bearing_A | 4 |
| External Force | [array] | Load (Garments) | [array] |
| Lifetime | [array] | Lifetime | [array] |

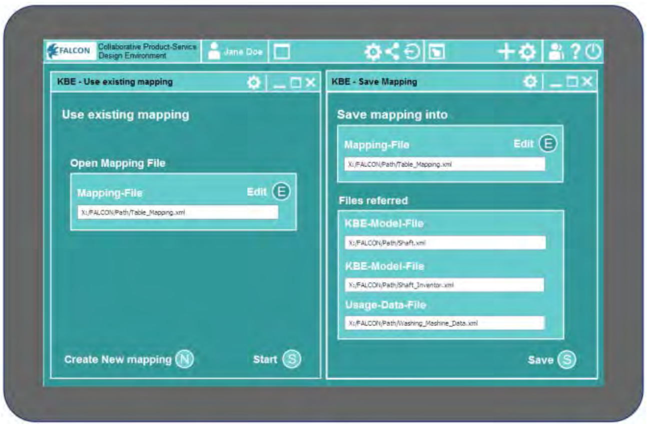
External Temperature ▼ Temp-Sensor_01 ▼

**Figure 9 KBE parameter mapping widget of FALCON VOP**

The GUI will provide the capability to manage the parameter mappings (to avoid double work). The mapping can be stored in so called mapping files, which include a reference to a source file (e.g. a CAD parameter file) and the mappings itself.

Users can create, open adapt and store mapping files. If engineering knowledge has been formalized in KbeML beforehand, the user can easily switch the mapping to adapt the product usage information:

1. Open the mapping file;
2. Search/select “external temperature”;
3. : adapt the mapping for “external temperature” from “Temp-Sensor_01” to Temp-Sensor_02”.
4. Initiate data transfer from “Data set to KBE”
5. Save the file for future use.



Use existing mapping

Open Mapping File

Mapping-File: X:/FALCON/Path/Table_Mapping.xml

Save mapping into

Mapping-File: X:/FALCON/Path/Table_Mapping.xml

Files referred

KBE-Model-File: X:/FALCON/Path/Shaft.xml

KBE-Model-File: X:/FALCON/Path/Shaft_Inventor.xml

Usage-Data-File: X:/FALCON/Path/Washing_Mashine_Data.xml

**Figure 10 Open mapping file in KBE widget of FALCON VOP**

**<u>Responsible Partner(s)/Developer(s):</u>**

*Developer:*

- Patrick Klein <klp@bba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Marco Franke <fma@biba.uni-bremen.de>

#### 2.3.3.7 FALCON KCCM Management Tool

## KCCM Management Tool
(falcon-service-KCCM)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The KCCM Management Tool is a user interface for KCCM configuration and status monitoring. The core idea is to provide an environment which allows to define and process a sequence of functions provided as methods by KCCM (ref. to 2.2.1.5 section).
To enable a clearly arranged User Interface the processing sequence is envisioned to be implemented in form of configuration-files (e.g. “KCCM_*sample*_config.xml), which can be uploaded and processed afterwards. The user will have a status window showing the process steps and will be able to pause/stop the process. The processing output will be stored by the KCCM in the triple-store and thus provided to other FALCON modules afterwards.
The configuration-file syntax will be based upon the defined KCCM methods. Sequences will be defined by word wraps and semicolon (similar to script languages); e.g.:

```
Get_average_for_period(x,y,z);
Get_average();
```

For data access the module will offer two options: selection of data-sets before processing the config-file or inclusion of SPAQL queries directly into the configuration-files. In this case the methods will include the required SPARQL syntax, which may be defined and tested with Query Builder in advance (2.2.3.5).

As outlined in section 2.2.1.5 the KCCM module will encapsulate output results in a json object and store results in the triple store at the same time. To enable users a first look into the results the KCCM Status Console will provide a result view on the data (ref. to mock-up below).

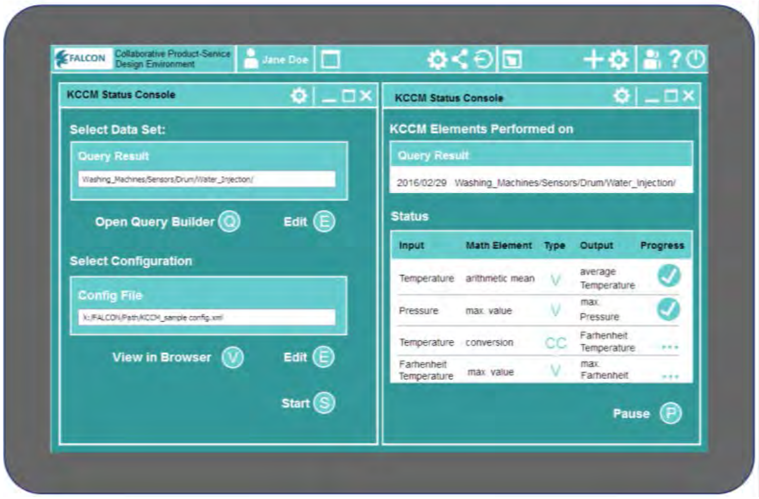
**Interfaces:**

The KCCM Management Module will not have an API (it will not provide methods to be used by other services). However, the module will of course have a direct connection to KCCM itself and use the methods provided by KCCM (section 2.3.1.5).

**GUI Mock-ups:**

Left part of the mock-up enables the user to select the data-set and the configuration-file. The Start button will initiate processing and activate the status monitoring. The latter is positioned on the left side. The progress is visualized by checkmarks. The pause button on the bottom left will interrupt the process.

Copyright ©FALCON Consortium 2015-2017



KCCM Status Console

Select Data Set:

Query Result

Washing_Machines/Sensors/Drum/Water_Injection/

Select Configuration

Config File

X:/FALCON/Path/KCCM_sample config.xml

KCCM Status Console

KCCM Elements Performed on

Query Result

2016/02/29 Washing_Machines/Sensors/Drum/Water_Injection/

Status

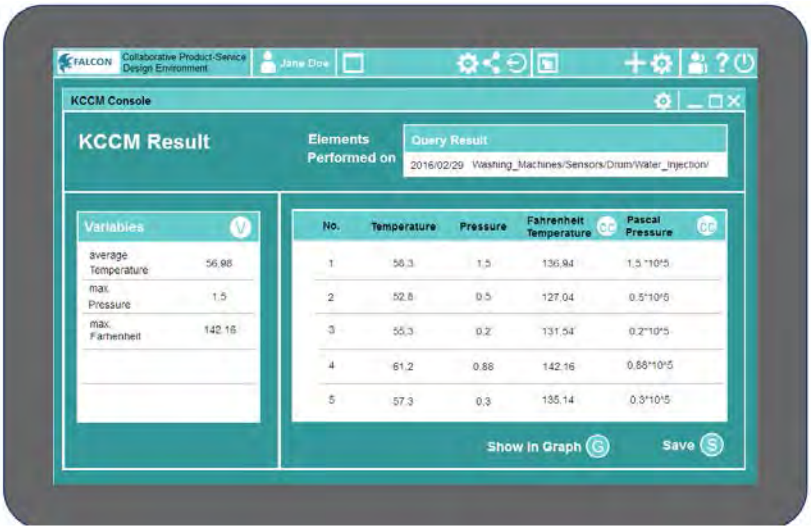
| Input | Math Element | Type | Output | Progress |
|---|---|---|---|---|
| Temperature | arithmetic mean | V | average Temperature | ✓ |
| Pressure | max. value | V | max. Pressure | ✓ |
| Temperature | conversion | CC | Farhenheit Temperature | ... |
| Farhenheit Temperature | max. value | V | max. Farhenheit | ... |

Figure 11 KCCM Status Console widget of FALCON VOP

The status table (left side) visualizes the different types of calculations that can be performed, namely calculated columns (displayed as type CC) and variables (displayed as type V). In addition, used inputs and outputs are outlined as variable names or field-names respectively.

In the given sample the config-file is processed first on the instances of the property "Temperature" of the "water_Injection" concept. The array with values is processed by the "arithmetic mean" method providing an output of type variable. The output is named "average temperature" and the checkmark indicates it is already processed.

In a next step the instances of the property "pressure" of the concept "water_Injection" are analyzed with the method max_value to identify the value for the max. pressure variable which represents the maximum pressure of the column.



KCCM Console

KCCM Result

Elements Performed on

Query Result

2016/02/29 Washing_Machines/Sensors/Drum/Water_Injection/

| Variables | V |
|---|---|
| average Temperature | 56.98 |
| max. Pressure | 1.5 |
| max. Farhenheit | 142.16 |

| No. | Temperature | Pressure | Fahrenheit Temperature CC | Pascal Pressure CC |
|---|---|---|---|---|
| 1 | 58.3 | 1.5 | 136.94 | 1.5 *10^5 |
| 2 | 52.8 | 0.5 | 127.04 | 0.5*10^5 |
| 3 | 55.3 | 0.2 | 131.54 | 0.2*10^5 |
| 4 | 61.2 | 0.88 | 142.16 | 0.88*10^5 |
| 5 | 57.3 | 0.3 | 135.14 | 0.3*10^5 |

Figure 12 KCCM Result widget of FALCON VOP

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Patrick Klein <klp@bba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Marco Franke <fma@biba.uni-bremen.de>

Copyright ©FALCON Consortium 2015-2017

### 2.3.3.8 FALCON Data Export Module

**Data Export Module**
(TBD)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module allows the exportation of data visualized the Data Analysis module in different formats: CSV, JSON, XML

**Interfaces:**

*Methods:*

1. *getCSVData*
    a. **Input**: parameters
    b. **Output**: CSV object containing the set of selected data
2. getJSONData
    a. **Input**: parameters
    b. **Output**: JSON object containing the set of selected data
3. getXMLData
    a. **Input**: parameters
    b. **Output**: XML object containing the set of selected data

**GUI Mock-ups:**

This module has no graphical user interface

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Weian Xu <weian.xu@holonix.it> (HOLONIX): Nexus(-), GIT(X), Jenkins(-)

Copyright ©FALCON Consortium 2015-2017

### 2.3.3.9 FALCON Idea Manager

**Idea Manager**
(TBD)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The Idea Manager application aims at supporting the creation and management of new Product Service ideas within a company or within a more generic ecosystem. It can be used to support an Open Innovation approach, where new ideas can come from different actors (both within and outside the enterprise), can be promoted and commented by peers. These new ideas are then selected by the company, after multiple criteria analysis to become new “concepts” that enter the complex process (beyond the scope of the Idea Manager but supported by the other FALCON or external tools) to transform them into (physical) prototypes, ready for a validation by real end users (consumers). In addition to the user interface, the Idea Manager system provides a small set of webpages that can be easily included in the Collaboration Environment as widgets. These pages are designed to offer dedicated interfaces to the Idea Manager for optimal access to specific information from within the Collaboration Environment, such as viewing the ideas that are being discussed during a collaboration meeting (e.g. an Idea Day, where people from different locations discuss and promote together the most interesting new ideas), voting on a concept or obtaining detail.

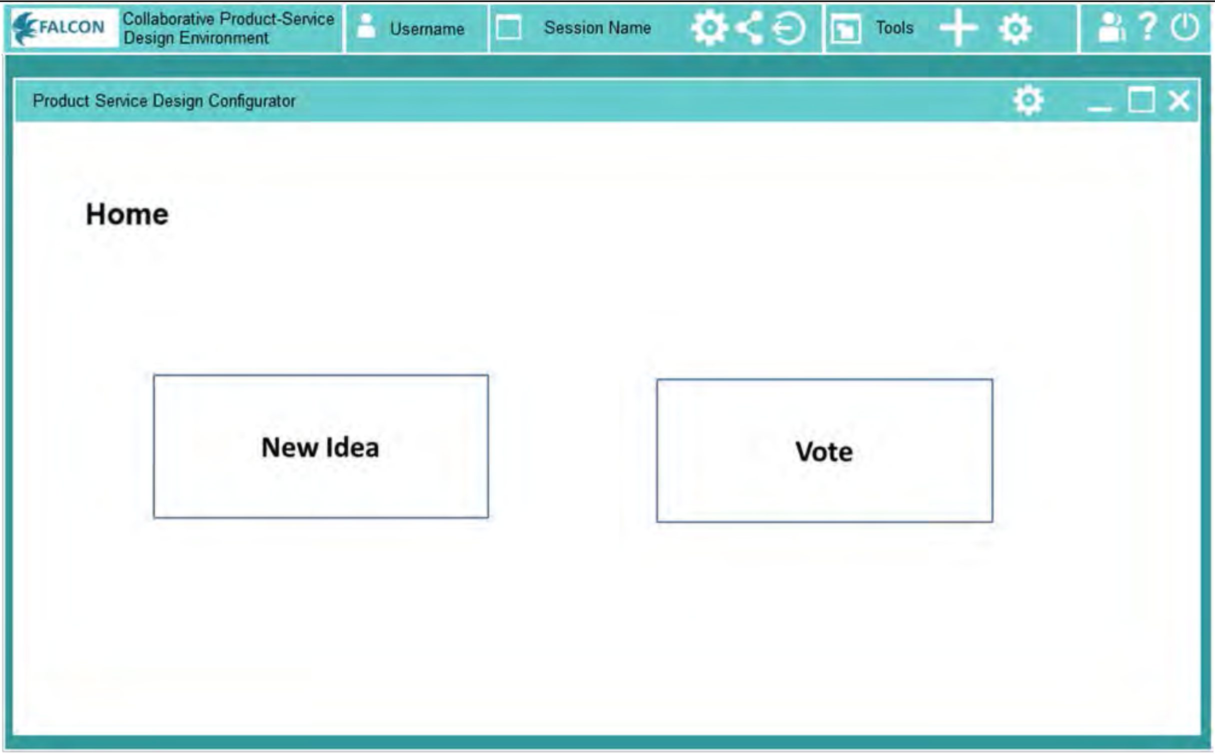
**Interfaces:**

*Methods:*

1. getIdeaData
    a. **Input**: ID request indicating the idea/ideas that external systems want to accede
    b. **Output**: JSON indicating the idea information
2. getWidgetURL
    a. **Input**: ID of the widget
    b. **Output:** *JSON indicating the URL of the idea widget*

**GUI Mock-ups:**

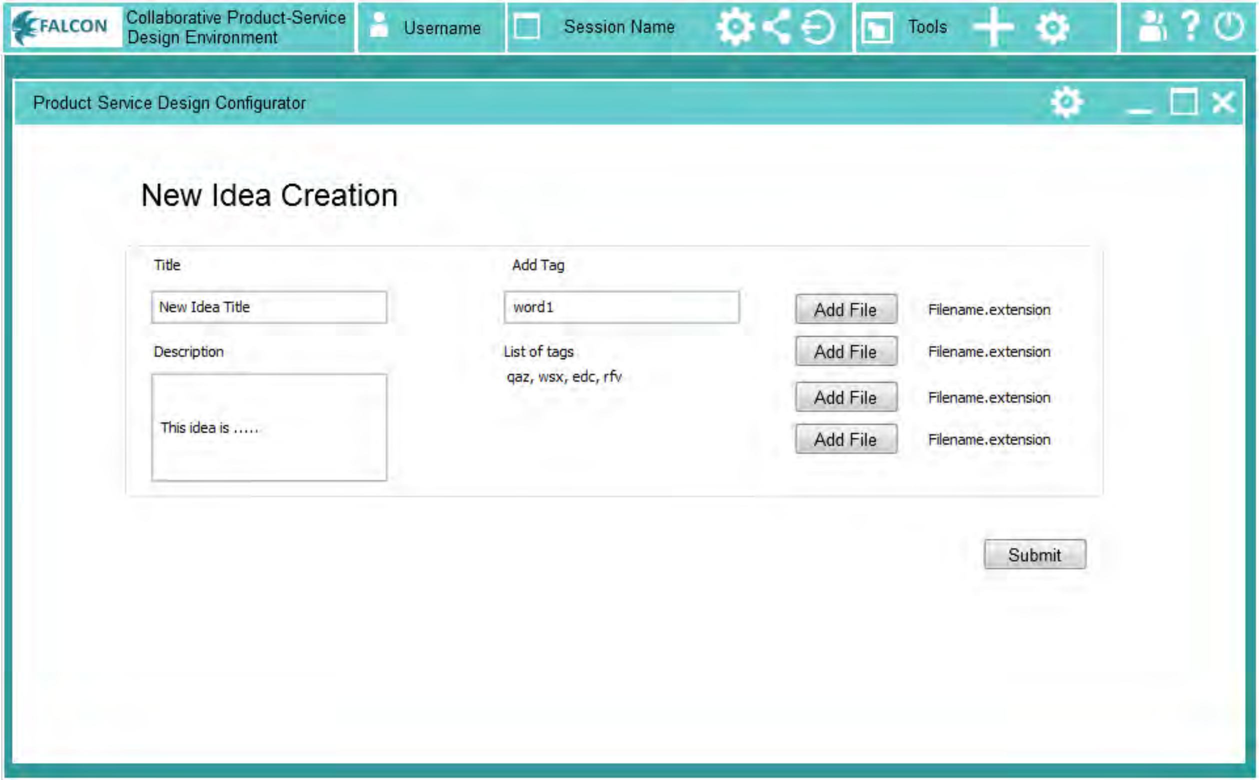
The user can decide what to do with the tool selection between: crate new ideas, vote or comment already presented ideas.


FALCON
Collaborative Product-Service Design Environment
Username
Session Name
Tools
Product Service Design Configurator
New Idea
Vote

Home

**Figure 13 Home widget of FALCON VOP's Product Service Design Configurator**

Clicking on submit the user can create new ideas through the provision of a title, a description, the usage of some attachments as well as some tags (ontology concepts) to facilitate the idea search.


FALCON
Collaborative Product-Service Design Environment
Username
Session Name
Tools
Product Service Design Configurator
Add File
Add File
Add File
Add File
Submit

New Idea Creation

Title: New Idea Title

Description: This idea is .....

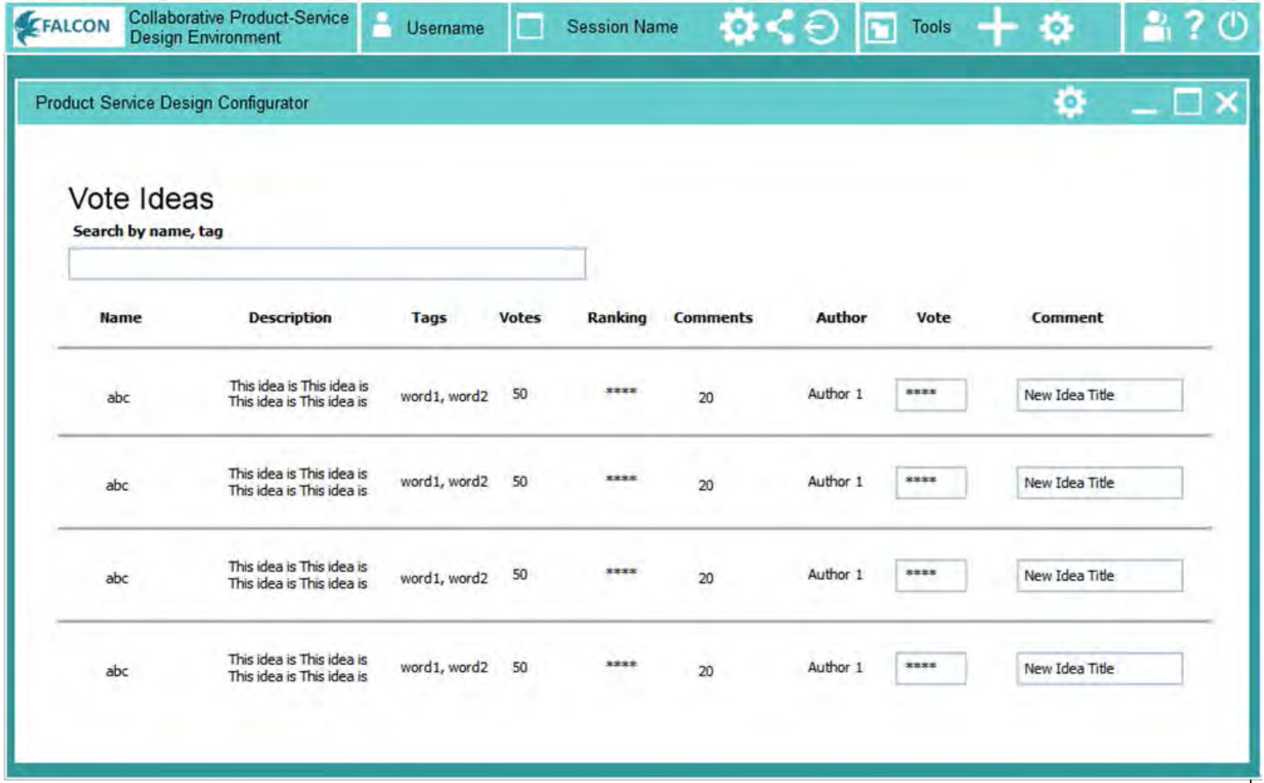
Add Tag: word1

List of tags: qaz, wsx, edc, rfv

Filename.extension
Filename.extension
Filename.extension
Filename.extension

**Figure 14 New Idea Creation widget of FALCON VO's Product Service Design Configurator**

Clicking on vote or comments the user can rate or comments the already existing public ideas.


FALCON
Collaborative Product-Service Design Environment
Username
Session Name
Tools
Product Service Design Configurator

Vote Ideas

Search by name, tag

| Name | Description | Tags | Votes | Ranking | Comments | Author | Vote | Comment |
|---|---|---|---|---|---|---|---|---|
| abc | This idea is This idea is This idea is This idea is | word1, word2 | 50 | **** | 20 | Author 1 | **** | New Idea Title |
| abc | This idea is This idea is This idea is This idea is | word1, word2 | 50 | **** | 20 | Author 1 | **** | New Idea Title |
| abc | This idea is This idea is This idea is This idea is | word1, word2 | 50 | **** | 20 | Author 1 | **** | New Idea Title |
| abc | This idea is This idea is This idea is This idea is | word1, word2 | 50 | **** | 20 | Author 1 | **** | New Idea Title |

**Figure 15 Vote Ideas widget of FALCON VOP's Product Service Design Configurator**

**<u>Responsible Partner(s)/Developer(s):</u>**

*Developer:*

- Sandro Taje < *sandro.taje@holonix.it* > (HOLONIX): Nexus(-), GIT(X), Jenkins(-)

### 2.3.3.10 FALCON Simulation and Forecasting Manager

**FALCON Simulation and Forecasting Manager**
(TBD)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The simulation and forecasting manager facilitates the FALCON end user in selecting and deploying the most suitable of available software tools for predicting (i.e., simulating or forecasting) MOL-related processes, based on given needs for life-cycle decision support, and on a given set of available input data from PEID (and/or possibly from social media).
It allows selecting/defining the envisioned decision support (e.g., "obtaining a forecast of supplies consumption" or "perform what-if studies to reduce effect $e$ by fine-tuning design parameters $x$, $y$, $z$"), and querying/filtering available input data, and it supports preparation of the selected data to a format that can be read by the prediction software. At least part of the considered software will be off-the-shelf, externally provided software (especially for simulation, see especially for simulation, see 2.2.4.3), but some of it may be available from within FALCON (especially for forecasting).

The Simulation and Forecasting Manager is provided as a proof-of-concept to show that one or more combinations of a need for decision support, available input data and available approaches can provide predictive information or knowledge that fulfils the specified need, and is thus valuable to the end user. Developing the Simulation and Forecasting Manager as readily deployable software is not foreseen within the project, but critical algorithms will be implemented in a form that allows testing.

**Interfaces:**

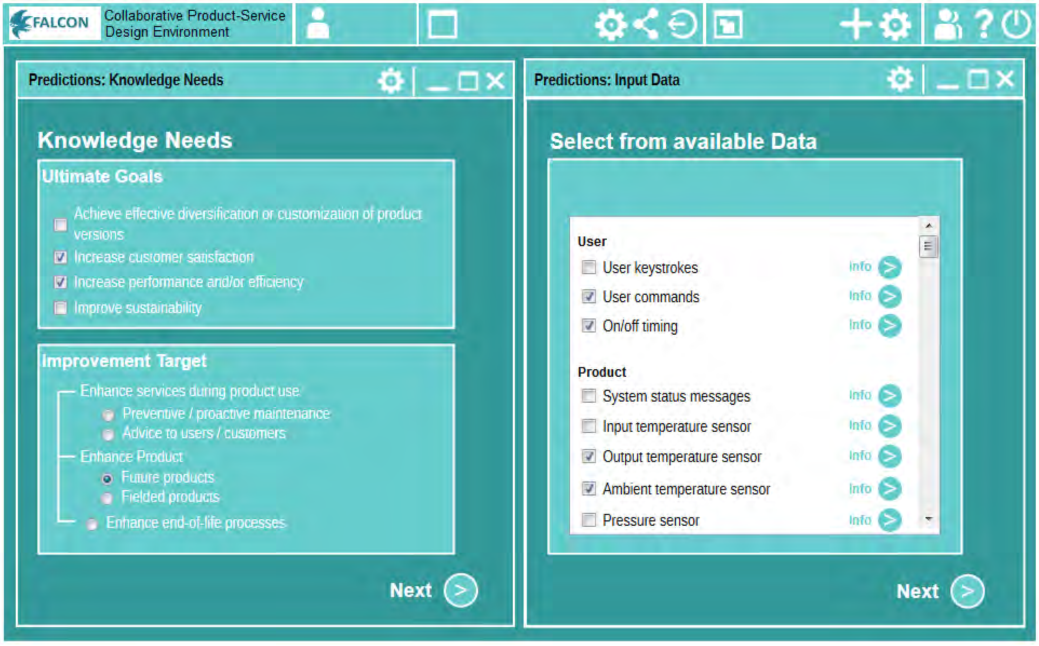
*Methods:*

1. *MatchNeedsToResources*
   a. **Input**: User selection of characteristics that define the envisaged decision support; User selection of input data from SPARQL Endpoint; user input of filtering criteria for data; available software/tools software for prediction
   b. **Output**: Suggestion of suitable software/tools for prediction
2. *PreparePredictionInputs*
   a. **Input**: User selection from selected software/tools for prediction

**Output**: Input file suitable for selected software/tools (e.g. time-stamped csv file)

**GUI Mock-ups:**

Copyright ©FALCON Consortium 2015-2017



Predictions: Knowledge Needs

Knowledge Needs

Ultimate Goals

- ☐ Achieve effective diversification or customization of product versions
- ☑ Increase customer satisfaction
- ☑ Increase performance and/or efficiency
- ☐ Improve sustainability

Improvement Target

- Enhance services during product use
  - ○ Preventive / proactive maintenance
  - ○ Advice to users / customers
- Enhance Product
  - ● Future products
  - ○ Fielded products
- ○ Enhance end-of-life processes

Predictions: Input Data

Select from available Data

User

- ☐ User keystrokes
- ☑ User commands
- ☑ On/off timing

Product

- ☐ System status messages
- ☐ Input temperature sensor
- ☑ Output temperature sensor
- ☑ Ambient temperature sensor
- ☐ Pressure sensor

**Figure 16 Configuration widget for Simulation and Forecasting Manager of FALVON VOP**

The GUI is expected to allow selecting/defining the envisaged decision support, and querying/filtering available input data, and to support preparation of the selected data to a format that can be read by the prediction software.

**<u>Responsible Partner(s)/Developer(s):</u>**

*Developer:*

- Wilfred van der Vegte <w.f.vandervegte@tudelft.nl> (TU Delft): Nexus(-), GIT(-), Jenkins(-)
- Fatima-Zahra Abou Eddahab <f.aboueddahab@tudelft.nl>(TU Delft): Nexus(-), GIT(-), Jenkins(-)

#### 2.3.3.11 FALCON PUI Alert Module

**PUI Alert Module**
(TBD)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

This module is used to check the status of a particular product presenting detected alarms.

**Interfaces:**

*Methods:*

1. *getProductAlarms*
    a. **Input**: product ID as parameter
    b. **Output**: Object indicating the list of alarms for the selected product
2. *getAllAlarms*
    a. **Input**: -
    b. **Output**: Object indicating the whole list of alarms

**GUI Mock-ups:**

Copyright ©FALCON Consortium 2015-2017

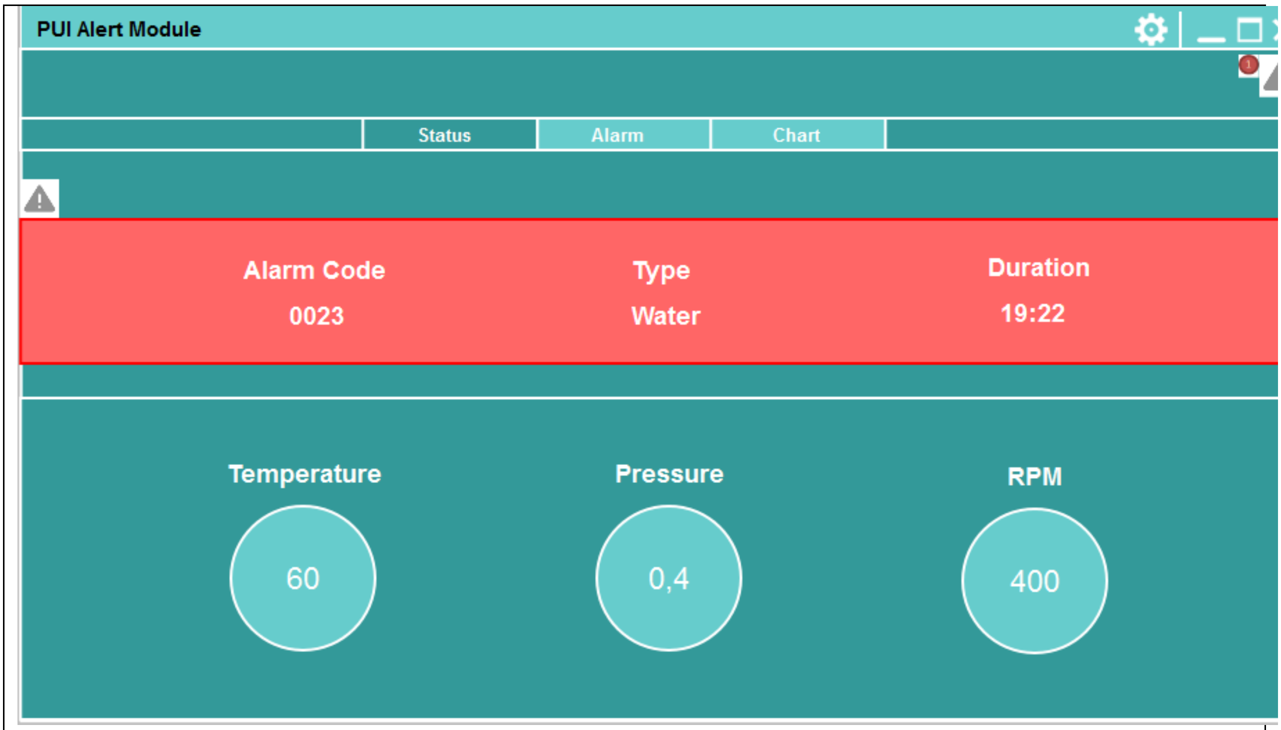


**Figure 17 Example of the FALCON VOP's PUI Alert widget**

The mock-up presents the conditions of working products. It shows some indicators regarding the products, in this case a washing machine, with some indicators like temperature, pressure and drum RPM. Moreover, in the top is possible to visualise real time warnings provided from the monitored product.

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Sandro Taje <sandro.taje@holonix.it> (HOLONIX): Nexus(-), GIT(X), Jenkins(-)

#### 2.3.3.12 FALCON PUI Query Builder

**PUI Query Builder**
(falcon-service-pui-query-builder)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The PUI Query Builder is a web application, which enables the creation of SPARQL queries. The objective is to create a graphical user interface, which enables a tool-supported creation of SPARQL queries. The support shall enable that users can create SPARQL queries without needing to have an understanding of SPARQL. For that purpose, the object oriented perspective of PUI is used to enable the selection of concepts and properties and a corresponding creation of SPARQL queries in the background

**Interfaces:**

*Methods:*

*Create Query*

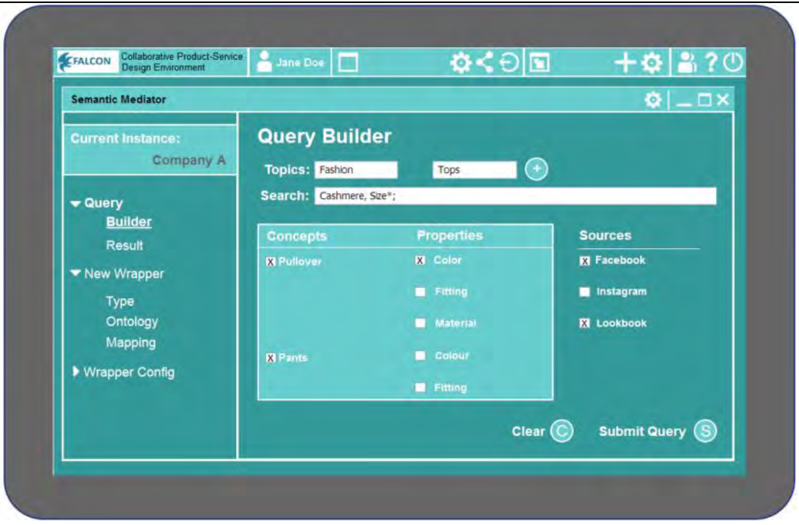
*c.* **Input**: The list of available wrapper configurations
*d.* **Output**: SPARQL query as JSON String

*SubmitQuery*

*a.* **Input**: Selected wrapper configurations and SPARQL query
*b.* **Output**: void

**GUI Mock-ups:**

The following mock up shows how a user can create SPARQL queries without the knowledge of the syntax and semantic of SPARQL.



Current Instance: Company A

Query Builder

Topics: Fashion Tops

Search: Cashmere, Size*;

Concepts: ☒ Pullover ☒ Pants

Properties: ☒ Color □ Fitting □ Material □ Colour □ Fitting

Sources: ☒ Facebook □ Instagram ☒ Lookbook

**Figure 18 Query Builder widget of FALCON VOP**

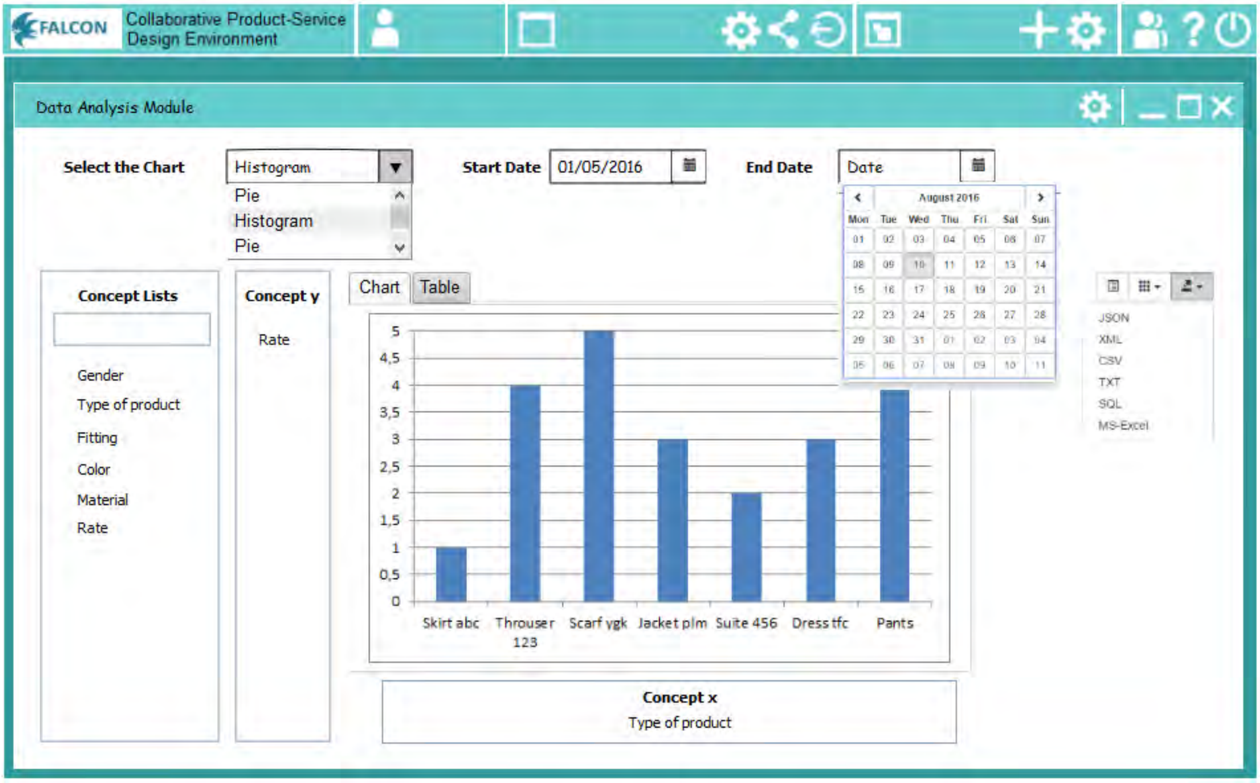
**<u>Responsible Partner(s)/Developer(s):</u>**

*Developer:*

- Marco Franke <fma@biba.uni-bremen.de> (BIBA): Nexus(X), GIT(X), Jenkins(X)
- Quan Deng <dqu@biba.uni-bremen.de> (BIBA) : Nexus(X), GIT(X), Jenkins(X)

#### 2.3.3.13 FALCON Data Visualisation Module

| |
|---|
| **Data Visualization and Exportation module**<br>(TBD) |
| **Module type:**<br>☐ VOP Core REST service<br>☒ Application/UI level tool<br>☐ PUI Wrapper<br>☐ Other |
| **Description of Main Responsibilities of this Module:**<br>The Data Analysis and Exportation module will allow users to select the kind of concepts he/she would like to visualize historical or punctual information through four kind of charts: Histograms, Linear Charts, Pie Charts as well as forecast. In case of simple data the module will directly contact the Triple Store Webservices in order to accede at the updated list of concepts and related data. In case of mathematical calculations (e.g. median, average, forecast analysis) it will contact the KCCM Module. This module will simplify the data analyzing through a visive approach. Finally, data shown could be exported for further analysis with other systems. |
| **Interfaces:**<br>*Methods:*<br>1. GetVisualizedData<br>a. **Input**: -<br>b. **Output**: text file indicating the list of visualized data |
| **GUI Mock-ups:**<br>Within the module the user can select the type of chart to see from the list, the initial and final date for the analysis as well as the concepts he/she would like to visualize. Moreover, the user can visualize the chart or visualize the data in table view and export the visualized data for further analysis. |



FALCON Collaborative Product-Service Design Environment

Data Analysis Module

Select the Chart: Histogram

Start Date: 01/05/2016

End Date: Date

Concept Lists: Gender, Type of product, Fitting, Color, Material, Rate

Concept y: Rate

Concept x: Type of product

**Figure 19 Data Analysis widget of FALCON VOP**

## Responsible Partner(s)/Developer(s):

*Developer:*

- Weian Xu <weian.xu@holonix.it> (HOLONIX): Nexus(-), GIT(X), Jenkins(-)

### 2.3.4 3rd Party Software

3rd Party Software provides a set of modules to enable the interaction with 3rd party software. In this section, the modules in the group 3rd Party Software are presented and explained in detail.

#### 2.3.4.1 3rd Party LCA Tools

**3rd Party LCA Tools**
(falcon-ui-lca-module)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

It assesses environmental impacts associated with the entire Product Life Cycle from cradle to grave. It comes to play the roleof an environmental impacts estimation tool for the business partners depending on the business scenarios and can compare these results. It extracts the relevant data such as exhaustion of resources from the repository, and automatically tune into the parameters considering the business scenario. It provides the basic schema to represent Product Life Cycle, processes, resources and so on.

It aims at implementation of analysis, evaluation and optimization of environmental impacts for smart innovators in the field of sustainability suppliers. Further on, depending on the definition of the study scope, it considers not only direct activities but also indirect activities such as purchased goods and services, and so on.

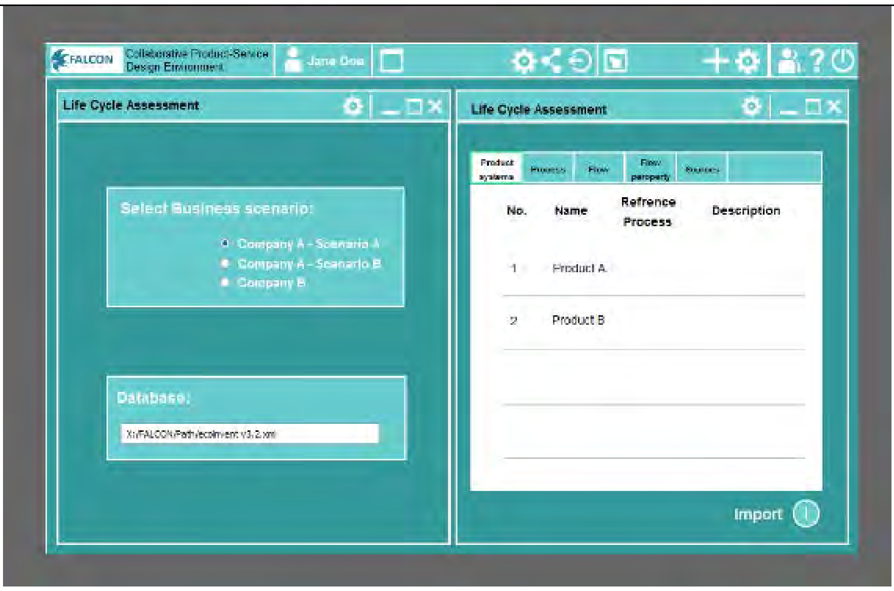
**Interfaces:**

*Methods:*

1. *Query*
   a. **Input**: SPARQL query to get environmental relevant data

**Output**: Float representing the String representing the analysis result

**GUI Mock-ups:**



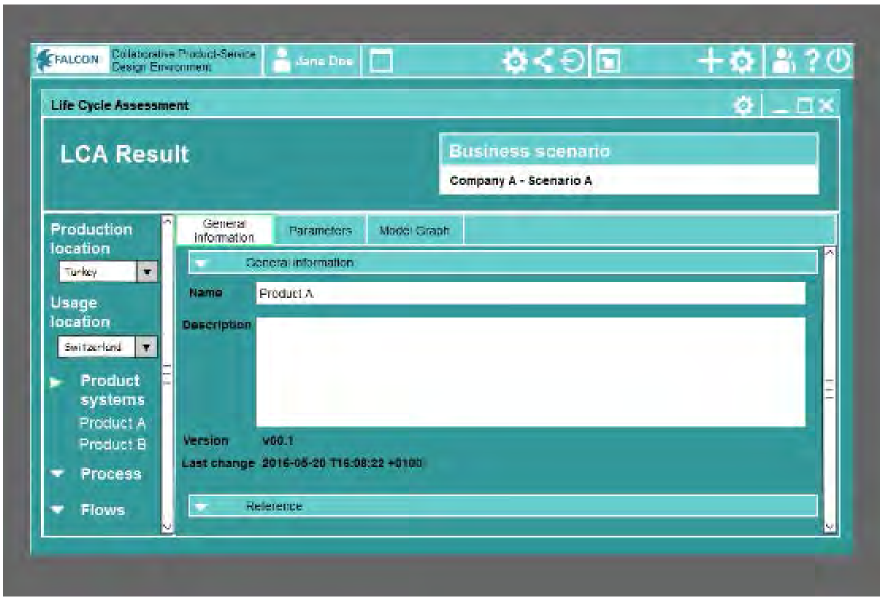
Select Business scenario:

- ● Company A - Scenario A
- ○ Company A - Scenario B
- ○ Company B

Database: X:/FALCON/Path/ecoinvent v3.2.xml

| No. | Name | Refrence Process | Description |
|---|---|---|---|
| 1 | Product A | | |
| 2 | Product B | | |

**Figure 20 Configuration widget of Life cycle Assessment of FALCON VOP**



Business scenario: Company A - Scenario A

Production location: Turkey

Usage location: Switzerland

Name: Product A

Description:

Version: v00.1

Last change: 2016-05-20 T16:08:22 +0100

**Figure 21 Result widget of Life cycle Assessment of FALCON VOP**

To define the study scope, UI provides the screen to select a business scenario with the production location and usage location. In addition, it supports to tune specific environment parameters for accurate results.

**Responsible Partner(s)/Developer(s):**

*Developer:*

- Sangje Cho <sangje.cho@epfl.ch> (EPFL): Nexus(-), GIT(-), Jenkins(-)

### 2.3.4.2 3rd Party CAx Tools

**CAx Tools**
N/A

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

The abbreviation CAx is covering the different support systems in design, production and other phases of product lifecycle management (PLM): such as **C**omputer **A**ided design (CAD); **C**omputer **A**ided engineering (CAE) or **C**omputer **A**ided manufacturing (CAM), and other. Due to the varying purpose a high diversity of systems and vendors are established in the market. Even though those systems are usually based on their proprietary and encapsulated internal structure and providing their own subset of interfaces, three different types for system-to-system communication and information exchange have been identified:

Import & Export Functions:

As outlined in the screenshots below the import and export functionality may differ from system to system, but usually parameters can be imported & exported as well as lists such as a bill of material. Of course the geometrical models can be exported to neutral standards (such as STEP/IGES) either. But usually the inheriting knowledge gets lost with the export.

Script & Macro-Languages:

Many systems provide some sort of macro programing language support (such as visual basic). The main purpose of the scripts macros is to enable some sort of automation within the application (the approach can be compared to macro's in MS office).

Application programming interfaces:

This access is often limited to "expert users", since extended licences are usually required, as well as specific knowledge on the internal structure of the system. The latter is demanding for documentation which is usually not accessable for free in those systems.

**Interfaces:**

*Methods:*

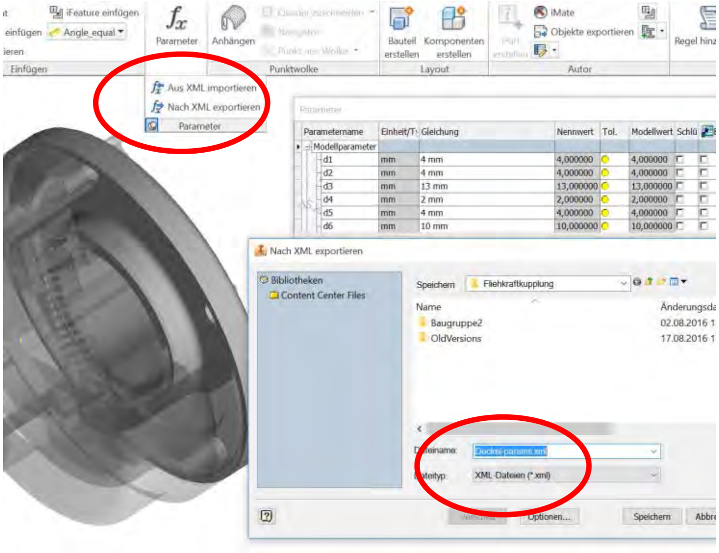
N/A

**GUI Screenshots:**

In the following, a screenshot of commercial CAD Systems is provided to visualize typical access to CAx systems (instead of mock-ups). The screenshot outlines parameter-

Copyright ©FALCON Consortium 2015-2017

import capabilities of the commercial CAD software INVENTOR (from Autodesk). The software allows a parameter export into an XML file and vice versa an parameter import from an XML file, if the file is structured accordingly. It has to be noted that “parameter import & export” can be interpreted more precisely as a “value import & export” for predefined parameters. Such XML files can be controlled with the mapping tool of the FALCON VOP.



Parameter

| Parametername | Einheit/T | Gleichung | Nennwert | Tol. | Modellwert | Schlü |
|---|---|---|---|---|---|---|
| Modellparameter | | | | | | |
| d1 | mm | 4 mm | 4,000000 | ● | 4,000000 | □ |
| d2 | mm | 4 mm | 4,000000 | ● | 4,000000 | □ |
| d3 | mm | 13 mm | 13,000000 | ● | 13,000000 | □ |
| d4 | mm | 2 mm | 2,000000 | ● | 2,000000 | □ |
| d5 | mm | 4 mm | 4,000000 | ● | 4,000000 | □ |
| d6 | mm | 10 mm | 10,000000 | ● | 10,000000 | □ |

Nach XML exportieren

Bibliotheken
Content Center Files

Speichern: Fliehkraftkupplung

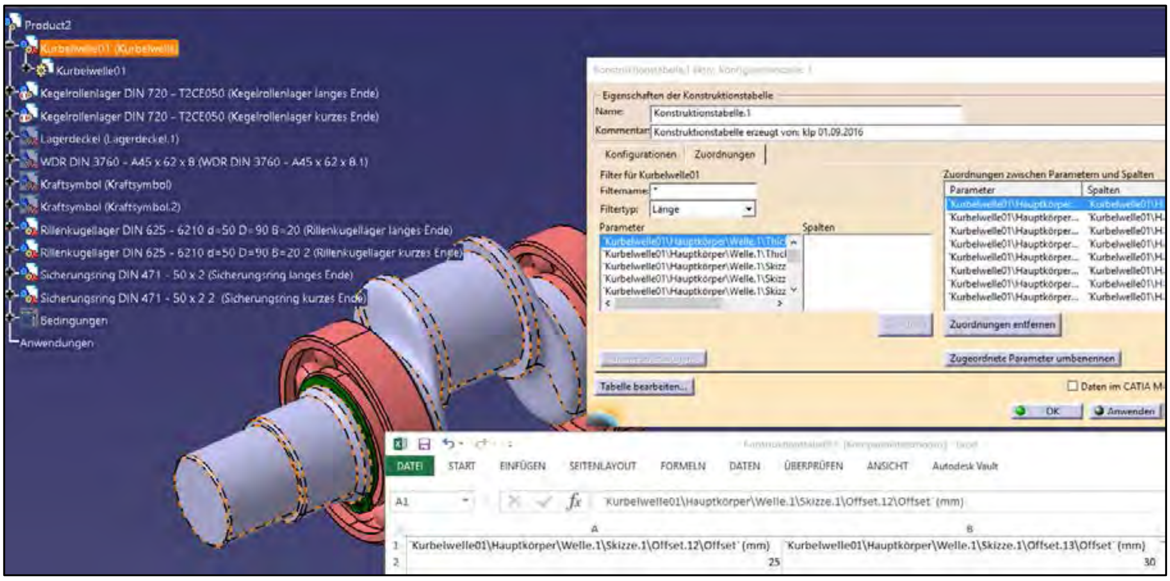
| Name | Änderungsda |
|---|---|
| Baugruppe2 | 02.08.2016 1 |
| OldVersions | 17.08.2016 1 |

Dateiname: Deckel-params.xml

Dateityp: XML-Dateien (*.xml)

**Figure 22 Example of CAD relevant parameters**

As outlined above the way to access parameters in CAx systems differs from system to system. For comparison, the following screenshot from CAD system CATIA illustrates the access to parameters in a similar but different way. The software can synchronize with XLS or coma separated (csv) files. Parameters are transferred to Field-names (column-names) of tables, which offers the possibility to handle an array of values for a parameter.


Product2
Kurbelwelle01 (Kurbelwelle)
Kurbelwelle01
Kegelrollenlager DIN 720 - T2CE050 (Kegelrollenlager langes Ende)
Kegelrollenlager DIN 720 - T2CE050 (Kegelrollenlager kurzes Ende)
Lagerdeckel (Lagerdeckel.1)
WDR DIN 3760 - A45 x 62 x 8 (WDR DIN 3760 - A45 x 62 x 8.1)
Kraftsymbol (Kraftsymbol)
Kraftsymbol (Kraftsymbol.2)
Rillenkugellager DIN 625 - 6210 d=50 D=90 B=20 (Rillenkugellager langes Ende)
Rillenkugellager DIN 625 - 6210 d=50 D=90 B=20 2 (Rillenkugellager kurzes Ende)
Sicherungsring DIN 471 - 50 x 2 (Sicherungsring langes Ende)
Sicherungsring DIN 471 - 50 x 2 2 (Sicherungsring kurzes Ende)
Bedingungen
Anwendungen

Eigenschaften der Konstruktionstabelle

Name: Konstruktionstabelle.1

Kommentar: Konstruktionstabelle erzeugt von: klp 01.09.2016

Filter für Kurbelwelle01

Filtername: *

Filtertyp: Länge

Parameter
`Kurbelwelle01\Hauptkörper\Welle.1\Thic
`Kurbelwelle01\Hauptkörper\Welle.1\Thicl
`Kurbelwelle01\Hauptkörper\Welle.1\Skizz
`Kurbelwelle01\Hauptkörper\Welle.1\Skizz
`Kurbelwelle01\Hauptkörper\Welle.1\Skizz

Spalten

Zuordnungen zwischen Parametern und Spalten

| Parameter | Spalten |
|---|---|
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |
| `Kurbelwelle01\Hauptkörper... | `Kurbelwelle01\H |

□ Daten im CATIA M

A1 `Kurbelwelle01\Hauptkörper\Welle.1\Skizze.1\Offset.12\Offset` (mm)

| | A | B |
|---|---|---|
| 1 | `Kurbelwelle01\Hauptkörper\Welle.1\Skizze.1\Offset.12\Offset` (mm) | `Kurbelwelle01\Hauptkörper\Welle.1\Skizze.1\Offset.13\Offset` (mm) |
| 2 | 25 | 30 |

**Figure 23 Parameter export capabilities in CATIA**

## Responsible Partner(s)/Developer(s):

Copyright ©FALCON Consortium 2015-2017

*Developer:*

- Patrick Klein <klp@bba.uni-bremen.de> (BIBA): Nexus(r-), GIT(-), Jenkins(-)

### 2.3.4.3 3rd Party Simulation Tools

**3rd Party Simulation Tools**
(TBD)

**Module type:**

☐ VOP Core REST service

☒ Application/UI level tool

☐ PUI Wrapper

☐ Other

**Description of Main Responsibilities of this Module:**

These tools are software packages already available on the market, or from open-source repositories. For a selected set of one or more simulation tools, the 3rd Party Simulation Tools will produce suitable input data based on which the end user will be able to run simulations using the 3rd Party Simulation Tools, which will support them in their decision making – e.g., (re)design decisions, (re)programming decisions, maintenance decisions.

**Interfaces:**

Interfaces depend on particular software package.

**Inputs**: data file in appropriate format, produced by 3rd Party Simulation Tools

**Outputs**: Graphics, animations and/or data files displaying predictive knowledge to FALCON end user.

**GUI Mock-ups:**

N/A (external)

**GUIs depend on particular software package**

N/A (external)

**Responsible Partner(s)/Developer(s):**

*Developer:*

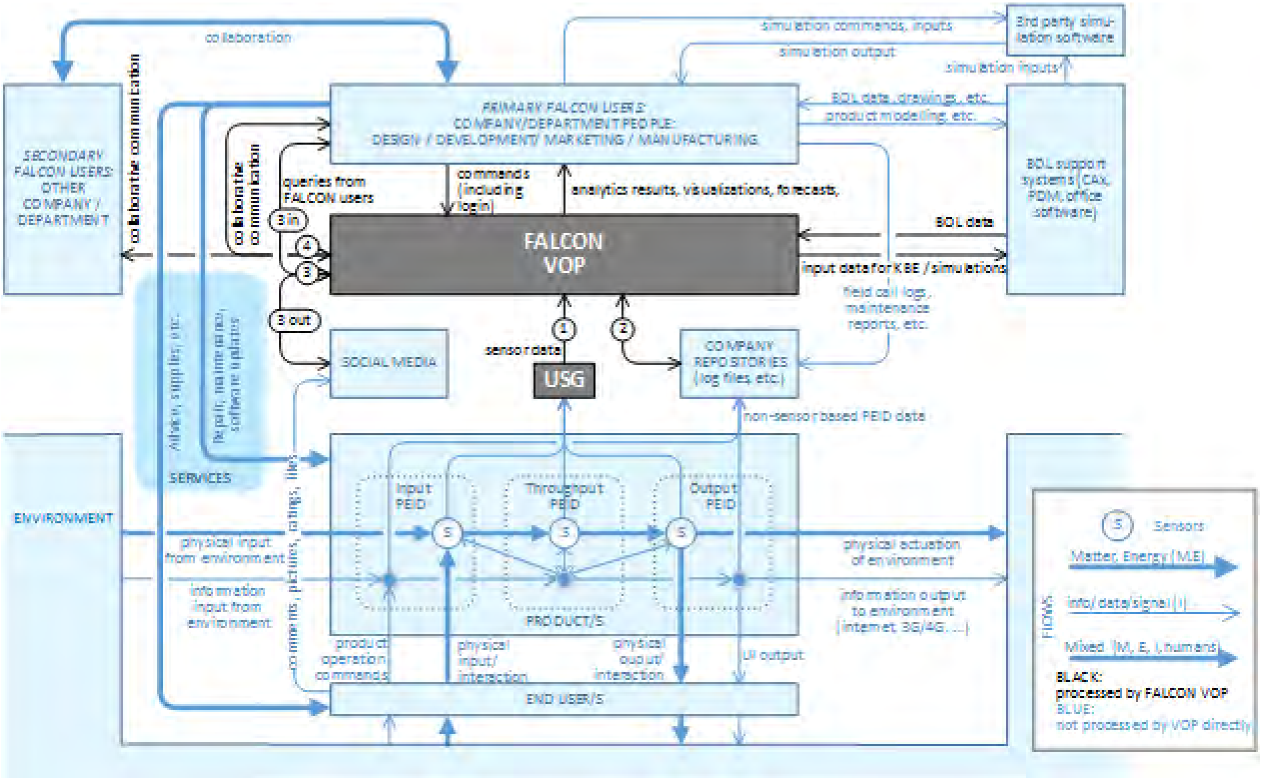
Already developed externally

### 2.3.4.4 Other Tools

The current evaluation of the requirements has identified that no other external tools (apart from the above mentioned) need to be integrated in the FALCON VOP. The ongoing development and evaluation activities could, however, result in findings which indicate the necessity to integrate additional tools which are not simulation, CAD or LCA tools. For this purpose, the category will be applied as a placeholder.

## 2.4 Interfaces with third Party Software and APIs

The module description forms included in Section 2 supplies a first definition of the interfaces that the VOP modules will implement internally. These interfaces will be refined and fully implemented in the context of work package WP1, in particular in the context of task T1.4. One other important part of the VOP architecture definition concerns integration with external and third party software systems, again of competence of task T1.4. The general overview over the available information and material flows between FALCON VOP and external world are shown in Figure 24.



collaboration
simulation commands, inputs
3rd party simulation software
simulation output
simulation inputs
SECONDARY FALCON USERS: OTHER COMPANY / DEPARTMENT
collaborative communication
PRIMARY FALCON USERS: COMPANY/DEPARTMENT PEOPLE: DESIGN / DEVELOPMENT/ MARKETING / MANUFACTURING
BOL data, drawings, etc.
product modelling, etc.
BOL support systems (CAx, PDM, office software)
collaborative communication
queries from FALCON users
commands (including login)
analytics results, visualizations, forecasts,
BOL data
3 in
4
3
FALCON VOP
input data for KBE / simulations
field call logs, maintenance reports, etc.
3 out
1
2
SOCIAL MEDIA
sensor data
USG
COMPANY REPOSITORIES (log files, etc.)
Advice, supplies, etc.
Repair, maintenance, software updates
non-sensor based PEID data
SERVICES
comments, pictures, ratings, likes
ENVIRONMENT
Input PEID
Throughput PEID
Output PEID
S
physical input from environment
information input from environment
physical actuation of environment
information output to environment (internet, 3G/4G, ...)
PRODUCT/S
product operation commands
physical input/ interaction
physical output/ interaction
UI output
END USER/S
S Sensors
FLOWS
Matter, Energy (M,E)
info/ data/signal (I)
Mixed (M, E, I, humans)
BLACK: processed by FALCON VOP
BLUE: not processed by VOP directly

**Figure 24 Flows between FALCON VOP and external**

In particular, the following ways of integrating with external systems are of relevance:

Sensors and social network data represent an important flow of information coming from the external world is of course represented by sensors and social network data, which represent the main input flows to the FALCON system. Integration with the sources of such information is of course implemented by the functionality of the relevant wrappers described in section 2.3.2 (and subsections).

A second aspect of the integration with external and third party software concerns how data is exported from the FALCON VOP to external applications for their consumption. Specifically, the data to external simulation and forecasting software exploited in work package WP4 has to be provided. This kind of integration often requires some heavy development work addressing specific legacy systems with their proprietary APIs and interfaces, leading to a risk of vendor lock-in. A better approach is to identify some set of standards or widely used data formats for data exchange and allow therefore, the transfer of information via csv or other formats. As mentioned, the details will be analyzed and defined during task T1.4

Another mechanism we considered in order to allow external systems to integrate with the functionalities provided by the FALCON VOP consists in the publishing of an open Application Programming Interface, exporting VOP functionality through a set of public well-documented functions. This is what we have defined as the FALCON OpenAPI. The best candidate for this is the definition of a REST interface providing a coherent access to FALCON 's data to external applications. This way, the possibility of integration is not limited to those specific third party applications that might have been taken into consideration during the FALCON project lifetime for pilot development and demonstration purposes, but can be extended in future to other software by the development of specific "oubound" wrappers exploiting the OpenAPI in their implementation.

A last but not less important aspect of integration with external system concerns the integration of the FALCON authorization and access control framework with authentication and authorization data that companies might already use in their legacy systems. Such an integration will be considered in the context of the specific task T3.5 of Workpackage WP3, which will start in M22.

# 3 Technical Quality Standards

This section presents the approach adopted in FALCON to ensure that the architecture and the resulting software modules, including all its dependencies, fulfill technical quality standards regarding performance, reliability and efficiency. Starting with the introduction of the overall verification and validation definition, the proposed development process as well as the applied verification and validation (V&V) are presented. Subsequently, the criterion of performance, reliability and efficiency and its application in the FALCON project are described in detail.

## 3.1 Overall approach

The technical quality will be ensured through verification and validation (V&V). The proposed approach applies the standards ISO 14756, ISO/IEC 9126[1] and recommendations of International Software Testing Qualification Board to ensure the technical quality of the FALCON VOP. In general, the international literature is replete with various validation and verification definitions. FALCON adopts the ANSI/IEEE Std 1012-2012 (IEEE 2012) definition of V&V, as presented in the following table:

**Table 2 Verification & Validation**

| | Verification | Validation |
|---|---|---|
| **Definition** | The process of providing objective evidence that the software and its associated products conform to requirements (e.g., for correctness, completeness, consistency, accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance); satisfy standards, practices, and conventions during life cycle processes; and successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities (e.g., building the software correctly). | The process of providing evidence that the software and its associated products satisfy system requirements allocated to software at the end of each life cycle activity, solve the right problem (e.g., correctly model physical laws, implement business rules, use the proper system assumptions), and satisfy intended use and user needs. |
| **Question** | Are we building the product right? | Are we building the right product? |
| **Objective** | To ensure that the product is being built according to the requirements and design specifications. | To ensure that the product actually meets the user's needs, the specifications were correct in the first place and the product fulfils its intended use when placed in its intended environment. |

To meet dynamic change requests, the parallel development processes in FALCON will take advantage of an agile approach to V&V. In the following, the proposed development process as well as the corresponding application of an appropriate V&V approach are presented.

[1] http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749

### 3.1.1 Development Process Model

The proposed development process model is aligned to the V-model and applies aspects of the agile development approach. In the following, the addressed V-model, the agile approach as well as the V&V procedures within the development processes are described in detail.

#### 3.1.1.1 V-Model

The V-model (see Figure 25) is an extension of the waterfall model; in fact, its left branch practically is the waterfall model, while its right branch includes approaches/techniques for the verification and validation of the software under development. Unit Testing, Integration Testing, System Testing, Operational Testing, Acceptance Testing constitute members of an indicative, yet not exhaustive, list of such verification and validation techniques. This detailed verification and validation phase are going to ensure the performance, the reliability, the efficiency and the applicability of the solution to the given FALCON vision, which is mandatory for a successful research project.

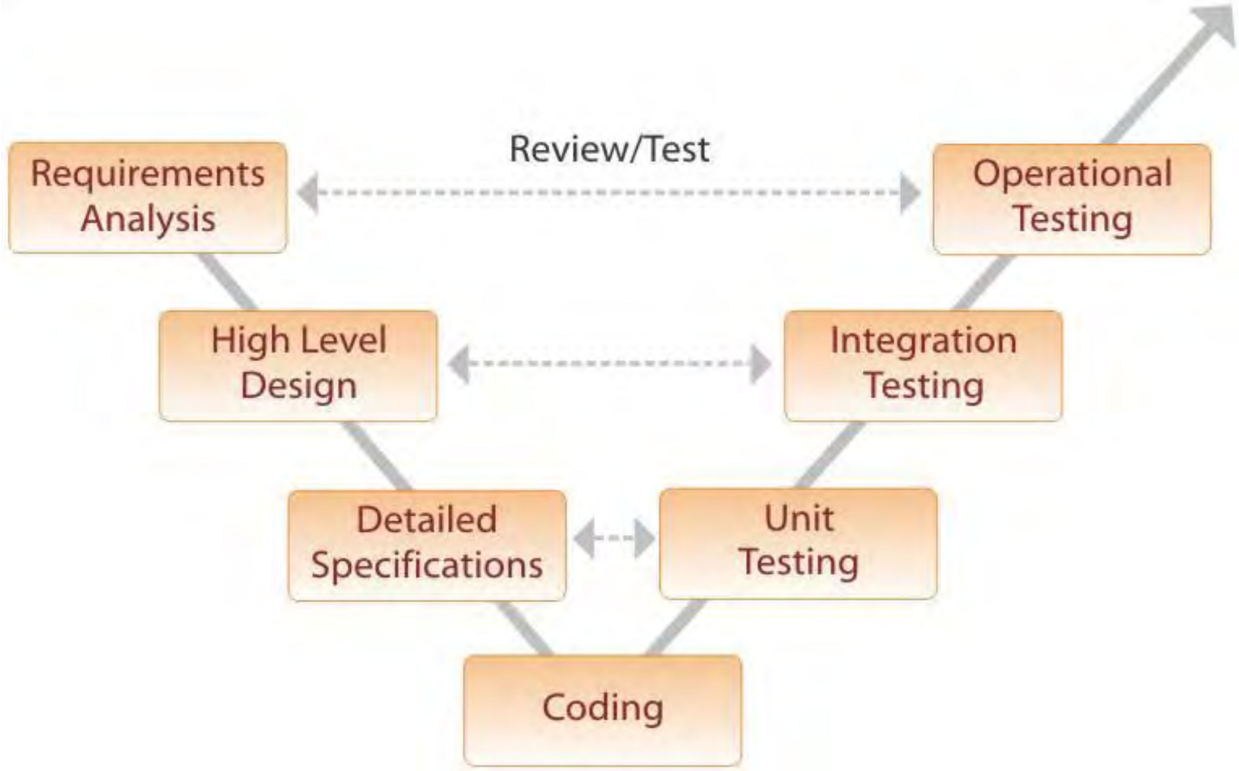


**Figure 25 Traditional V-model (SRM Technologies 2013)**

#### 3.1.1.2 Agile Software Development Model

On the other hand, agile software development (Larman 2004) has been gaining more and more momentum over the last few years. In contrast to the waterfall model, agile software development is based on an iterative and incremental workflow, aiming at rapid, collaborative and flexible end-product development. The main idea behind agile software development is seperate the main project into a larger number of relatively small increments; each one developed in short, efficient and effective iterations.

Unlike the V-model, in agile software development, there is no specific procedure (or set of procedures) dedicated to verification and validation. Each individual iteration involves working in all functions: planning, requirements analysis, design, coding, unit testing, acceptance testing etc. At the end of each iteration, a working product (regardless of its size and importance) is demonstrated to stakeholders who provide the necessary feedback. Although there are voices suggesting that agile is inappropriate for products of sequential nature, agile software development is an efficient and effective approach, accompanied with adaptability and predictability (Boehm and Turner 2004).

To enable a continuous, stakeholder-driven validation, general principles were carried out and concentrated, in particular, on the adoption of a common component model and communication model for the development of FALCON VOP modules. Technical requirements involving partners' expertise with specific technologies and/or integration with pre-existing software systems developed or used by partners have been examined. The examination of technical requirements will be updated each time the requirements of the business scenarios changes. Java was chosen as preferred platform and programming language for the development of the FALCON software modules and the Spring framework was selected for the implementation of the FALCON Virtual Open Platform architecture. Several alternative options, in particular a REST micro-services architecture and a central application based on the Spring framework, were considered, and consensus was finally reached on the micro-services approach.

The chosen technical approach enables the testability of one micro-service, a set of micro-services or the overall FALCON platform. Therefore, the full spectrum of test levels (unit tests through to system tests) are applicable and will ensure the quality of the VOP.

#### 3.1.1.3 Agile Methodologies in FALCON

In order to enable an efficient testing approach, the agile development approach suggests the application of an agile methodology for software design, development and integration activities. All partners have agreed to setup and exploit a stack of infrastructure tools allowing a process based on Continuous Integration (CI) and Delivery principles. This set of tools includes:

- A GitLab server for the management of versioned source code for the VOP modules.
- A Sonatype Nexus repository for the management and provision of compiled modules
- A Jenkins CI server to continuously monitor changes in the modules' source code in GitLab and automatically rebuild modified modules and notify the developer in case something goes wrong.
- A project management tool such as Redmine or Open Project to keep track of open issues and assign tasks to project members.

Besides the abovementioned tools, Linux container solutions like Docker and related technologies such as Docker Compose were considered for deployment and will be included in the CI deployment workflow. The provided virtualization will enable the delivery and maintenance of different development states to the business cases and enables a continuous evaluation through the business partners and feedback.

The quality of the software engineering work will be ensured through the FALCON development process, which is shown in Figure 26. This process enables the immediate detection and quick reaction of failures on all testing levels when a developer submit a new version of his source code.

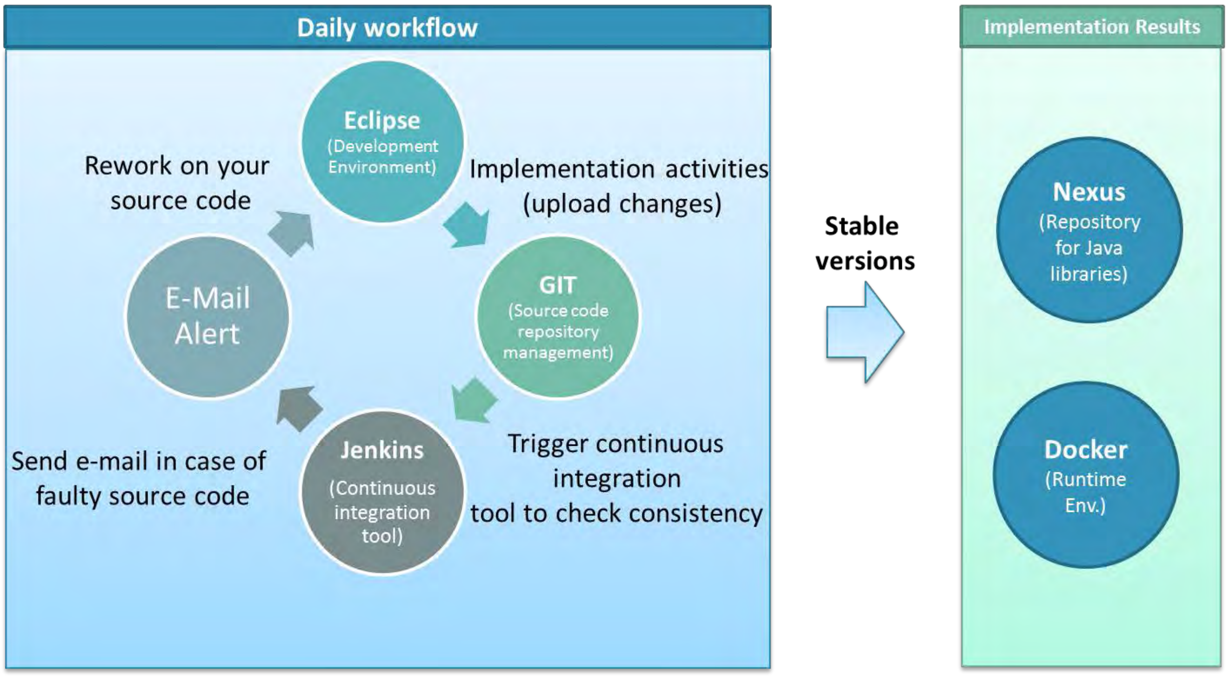

Daily workflow
Eclipse
(Development Environment)
Implementation activities
(upload changes)
GIT
(Source code repository management)
Trigger continuous integration
tool to check consistency
Jenkins
(Continuous integration tool)
Send e-mail in case of faulty source code
E-Mail Alert
Rework on your source code
Stable versions
Implementation Results
Nexus
(Repository for Java libraries)
Docker
(Runtime Env.)

**Figure 26 FALCON development process**

The continuous monitoring of the FALCON software prototype quality is ensured on basis of test cases which are executed by the continuous integration tool each time a source code change was detected. In the following, the scheduled application of the test driven approach is presented in detail to ensure the quality with respect to the performance (robustness and speed) and reliability (functionality and reliability). Then the estimation process for the overall software quality by stakeholders is presented in 3.4.

### 3.1.2 Test Case Definition, Specification and Implementation

The abovementioned V-model as well as the FALCON development process is based on test cases to ensure the software quality in the development process. The following table summarizes the ongoing testing capabilities.

**Table 3 FALCON Test Strategy**

| Test level | Test Method | Coverage | Test Bed | Responsibility |
|---|---|---|---|---|
| Unit testing | Black Box | MC[2] : 100% | Development environment: Jenkins | Micro-service owner |
| | White Box | BC[3]: 70% | | |
| Integration testing | Black Box | APC direct[4]: 100% | Jenkins | Micro-service owner-who invokes another service |
| | | APC Indirect: 50% | | |
| System testing | Black Box | Requirement Coverage: 100% | Docker instance per business case | Business case owner |

The responsibility for the specification, implementation and integration of the relevant set of test cases is dependent on the testing level. The responsible person can choose the methods for determining the test space

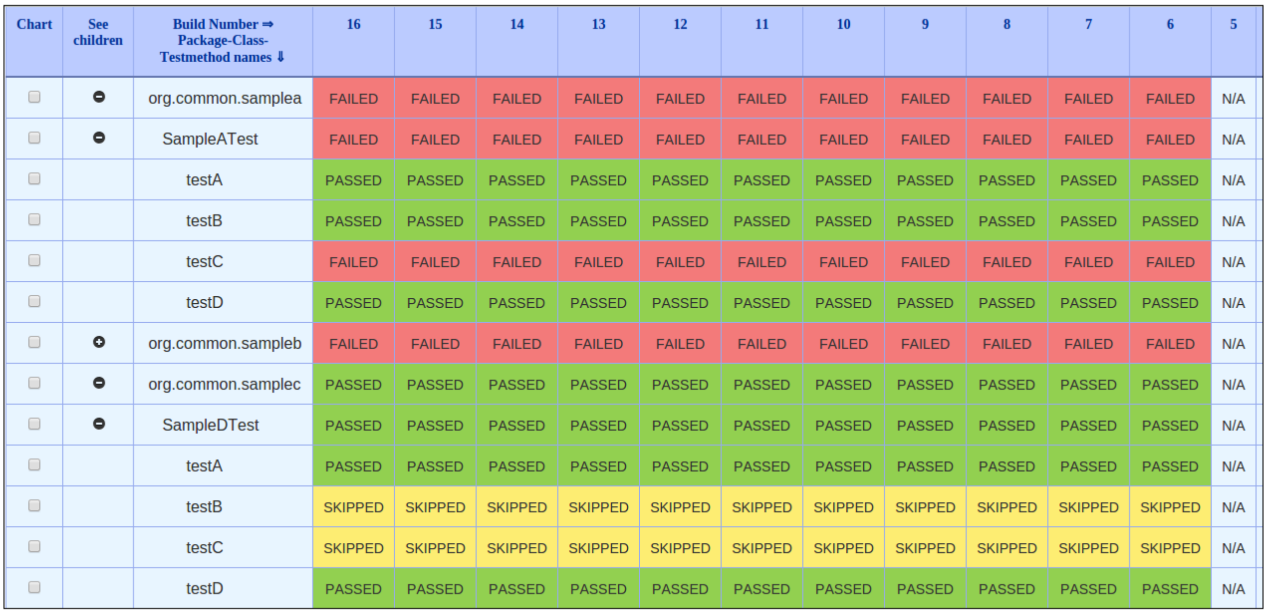
[2] MC: Method Coverage

[3] BC: Branch coverage

[4] APC: Abstract API coverage

and SUT behaviour. Common methods like equivalence partitioning, boundary value analysis, decision table testing is recommended. The FALCON developers create no test plan how it is defined in IEEE 829-2008 but rather the developers update the amount of specified and implemented test cases each time they update the source code on Git. The intended effect to the VOP will be classified into a testing level and determines the test methods and the coverage to be applied.

Jenkins will create the test report automatically. For that purpose, a "Test Results Analyzer Plugin" will be installed and used to create test reports. The generated reports will summarize the outcome of each test case according to the build number. An example is given in Figure 27.



| Chart | See children | Build Number ⇒ Package-Class-Testmethod names ⇓ | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| □ | ⊖ | org.common.samplea | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | N/A |
| □ | ⊖ | SampleATest | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | N/A |
| □ | | testA | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | N/A |
| □ | | testB | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | N/A |
| □ | | testC | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | N/A |
| □ | | testD | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | N/A |
| □ | ⊕ | org.common.sampleb | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | FAILED | N/A |
| □ | ⊖ | org.common.samplec | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | N/A |
| □ | ⊖ | SampleDTest | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | N/A |
| □ | | testA | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | N/A |
| □ | | testB | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | N/A |
| □ | | testC | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | SKIPPED | N/A |
| □ | | testD | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | PASSED | N/A |

**Figure 27 Test Results of Analyzer Plugin**

### 3.1.3 Proceeding, Validation of Performance, Reliability and Efficiency

Following the Agile methodology principles, the FALCON development process will pass different development cycles. After each development cycle, a verification and validation activity will be executed. The proposed validation and verification methodology is shown in Figure 28.

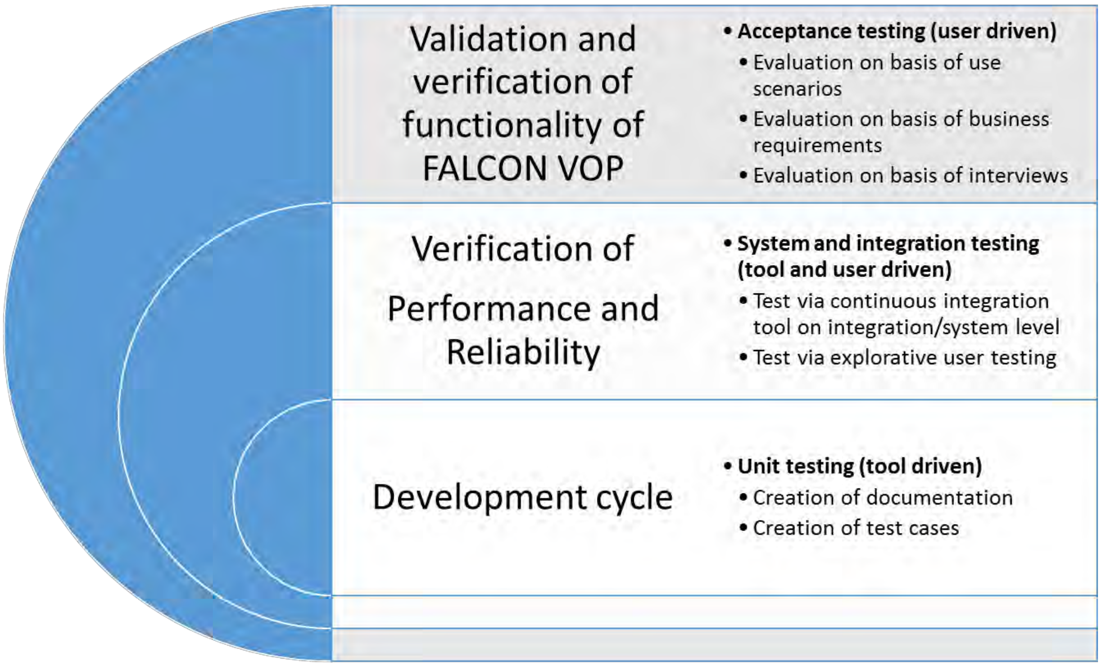

Validation and verification of functionality of FALCON VOP
• Acceptance testing (user driven)
• Evaluation on basis of use scenarios
• Evaluation on basis of business requirements
• Evaluation on basis of interviews
Verification of Performance and Reliability
• System and integration testing (tool and user driven)
• Test via continuous integration tool on integration/system level
• Test via explorative user testing
Development cycle
• Unit testing (tool driven)
• Creation of documentation
• Creation of test cases

**Figure 28 Validation and verification approach**

During the cycle execution, the verification of performance and reliability will be ensured on two levels. Firstly, it will be ensured on unit test level throughout the whole development cycle. Secondly, it will be ensured on integration test and system test level after a development cycle has ended. The duration of a development cycle is determined dynamically with respect to its complexity. This proceeding differs to a fix period of a sprint in the agile development process.

It is worth noting that this set of tests will be updated by the partners responsible of each module during every cycle, integrating the new objectives and taking into account possible changes. Possible failures in the automatic tests will be addressed by developers responsible for the involved module(s), within the current development cycle or in the following ones.

When a cycle is completed successfully until the system test level, the test beds built upon business stories will be updated with the new software, and the evaluation of Efficiency will take place. The Efficiency (user driven verification and validation) includes:

- the execution of business test cases
- the review by key users
- the creation of evaluation reports
- the improvement of requirements

Hereby, the review of ‘key users’ will produce evaluation reports; after evaluation reports, they will also be interviewed in order identify improvements and changed requirements for the following cycle.

It is worth noting that, for the validation of each cycle and for each business scenario, it will be necessary to describe the subset of the test cases and the parts of scenario currently covered by the software development. The responsibility for the validation and verification of the usability is both of the business case partner and the supporting technical partner. The user centric review process envisages a couple of documents to enable an objective review and evaluation process. The proposed documentation material, which could be applied, is shown in the following listing.

a. Template for evaluation reports

b. template for reviews

c. template for interviews

d. concrete interviews

The templates and a base version of the interviews shall be provided within the activities of evaluation tasks (T5.4, T6.4, T7.4, T8.4).

The concrete interviews to be used for a validation cycle shall be created from the base version, taking into account the received validation reports.

Moreover, for each business case, the identification of the following staff is required:

- Key users
- Personnel dedicated to review the key users when using the scenario
- Personnel dedicated to interview the key users
- Personnel dedicated to business test case execution

This personnel will be indicated by the business case partners and their supporting technical partners in the context of evaluation tasks (T5.4, T6.4, T7.4, T8.4)

## 3.2 Performance

The performance of a software can be determined and ensured through stress testing, which focus on software testing activities regarding software robustness. The application of the V&V standard in addition to the tool chain of the FALCON development process enables stress testing on all testing levels. The bottleneck of performance can be detected from micro-service level via integration level between more micro-services and or on the overall platform level. Naturally, small changes within a micro-service could have significant impact on the overall performance. To detect changes on micro-service level regularly (during and after a development cycle), which would downgrade the performance, stress testing is a mandatory part of the daily work of the continuous integration tool Jenkins.

In doing so, the stress testing covers load as well as stress tests. A stress test tries to break the system under test by overwhelming its resources or by taking resources away from it. Examples of possible stress test with respect to the FALCON micro-service architecture could be:

- Increase the baseline number for concurrent users/HTTP connections
- Take essential resources like the Triple Store offline and restart it
- Close randomly ports and open it to simulate random firewall settings by the stakeholders.

A load test is conducted in a controlled environment, which should be similar to the IT infrastructure of the later stakeholder. Then, load tests are executed which move the amount of data from low loads too high. The high amount of data should be so high, that it will not occur in a real environment.

Both kinds of tests are standardized through the ISO 14756[5], which describe how to execute these tests through setup the user's profiles, determine the momentum for low and high data loads and finally execute the measurements.

[5] http://www.iso.org/iso/catalogue_detail.htm?csnumber=25492

## 3.3 Reliability

The reliability of a software is determined and ensured through reliability testing which focuses on the fulfillment of the functional requirements. In so doing, the software's ability to function correctly according to given environmental conditions and for a particular amount of time should be ensured. The application of the V&V standard in addition to the tool chain of the FALCON development process enables the execution of tests on all testing levels to check the correct function with respect to varying input parameter values and environmental aspects. In theory, input data is gathered from various stages of development, such as the design and operating stages. The tests are limited due to restrictions such as cost and time restrictions. For that reasons the chosen test space is determined through specification-based or black-box techniques. The International Software Testing Qualification Board [6]teach the following methods to identify the relevant test space and support the necessary amount of test cases: equivalence partitioning, boundary value analysis, decision table testing, state transition testing and the use case testing. The listed methods will be also applied in the FALCON project to achieve the necessary test coverage of such kind of modular solution.

The specified and created test cases will be added physically to the micro-service source code. This linkage enables an automatic tool chain to check the reliability. Each time a micro-service was changed and the source code are uploaded, the continuous integration tool executes all relevant test cases to ensure the ongoing reliability of the overall solution. In the worst case, developers will be informed which modules violates the reliability in which test cases. This approach enables a fast reaction to faulty changes and therefore, it will result in mostly stable software snapshots for the stakeholders.

## 3.4 Efficiency

Here, the term efficiency addresses not the performance aspect but rather the overall quality aspect of the software. The ISO/IEC 9126 defines the quality aspects and enables the estimation of the overall software quality. The estimation is based on criteria, which are categorized in groups. In the following, the groups are listed which are considered in the FALCON project.

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

The groups functionality/reliability are also considered in 3.3 and the groups efficiency is also considered 3.2 on bases of test cases in detail. The consideration of these groups through automatic testing approaches establish the quality ensurance during the development phase.

The quality of the FALCON VOP will be evaluated on basis of prototypes in different stage of expansion. The evaluation process will not only be applied on basis of test cases but rather on the strong involvement of the stakeholders and key users. In consequence, the listed criteria groups will be evaluated through interactive reviews and subsequent interviews. The objective of a review is to monitor the key users while they are using the software in defined scenarios. The scenarios will be defined on basis of the developed business stories (D5.1 – D8.1) and will cover the requirements and complexity of the proposed real world application

[6] http://www.german-testing-board.info/fileadmin/gtb_repository/downloads/pdf/lehrplan/ISTQB_FL_Syll_2011.pdf

scenario. After each review, the key users create an estimation report on basis of the criteria groups. Subsequently, an interview is held to identify improvements and changed requirements for the next iteration.

# 4 Summary and Outlook

## 4.1 Summary

The requirements identified in D1.1 as well as the requirements which are the outcome of the Business Cases (WP5-WP8) have been used to define the FALCON VOP architecture, which is shown in section 2.2. For each contained FALCON module, a corresponding module description was included in this deliverable. For each module descriptions, the main functionality and GUI mock-ups have been documented. The link of the conceptual module description to the current software engineering activities has been realized through a micro-service name and a corresponding interface description. Both kinds of information are part of a module description.

Apart from the architecture and list of module descriptions, the deliverable has presented the technical quality standards to be applied. The quality standards focus on performance, reliability and efficiency. To ensure the three quality criteria, FALCON applies a software engineering approach, which combines the benefits of the V-model, and the agile approach. The validation and verification will be based on a test process as well as on user driven reviews.

The test process ensures the quality of the software prototypes through test cases covering the following test levels: unit testing, integration testing and system testing. All test cases will be part of the software modules and will be created for each test level. The inclusion of test cases into the FALCON VOP modules will enable an automatic test execution and quality control. For this purpose, the FALCON development process identified a set of development tools to provide the continuous and automated testing approach. The testing details like the coverage criteria or the responsibilities have been described in 3.1.2 in more details.

In contrast, the reviews are based on scenarios. The identified key users evaluate the FALCON VOP on basis of scenarios. The resulting evaluation results will be collected via interviews and subsequently will be analysed. The interviews will follow a strict structure to be quick evaluable and therefore, direct useful for the developers. The evaluation and interview process has been described in 3.1.3.in more detail

## 4.2 Outlook

The development of the listed software modules/micro-services has been started in the technical work packages (WP1 -4). The results are going to be reported in the corresponding deliverables. The proposed architecture and the corresponding set of modules are not fixed. The ongoing evaluation, which is based on prototypes in different progress states, could result in additional or changing functional requirements. In such cases, the architecture and affected module description will be updated and uploaded to Owncloud to have the newest version available.

# 5 References

Boehm , B., and R. Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Vols. ISBN 0-321-18612-5 Appendix A, pages 165–194. Boston: MA: Addison-Wesley, 2004.

Larman, Craig . Agile and Iterative Development: A Manager's Guide. 2004.

SRM Technologies. Embedded 360. 2013. http://www.embedded360.com/execution_approach/traditional_model.htm.