Smart Anything Everywhere Initiative

Area 3: Advanced micro-electronics components and Smart System Integration
Project: H2020–No 761809

Digital Innovation Hubs boosting European Microelectronics Industry

Manufacturing Experiment
Step-by-Step Integration Guide

Author(s):	
Status - Version:	V01
Date:	13th of March 2018
Last update:	13th of March 2018
Distribution - Confidentiality:	Public
Code:	DIATOMIC_Manufacturing_Experiment_Step_by_Step_Integration_FINAL.docx

Abstract:
This manual provides useful insight on how to interact with Virtual Fort Knox Clients, APIs and User Interfaces and provides a basic idea of the interaction of the different components of the manufacturing experiment (subject to change).

Disclaimer

The DIATOMIC Consortium is the following:

Participant number	Participant organisation name	Short name	Country
01	INTRASOFT International S.A.	INTRA	BE
02	F6S NETWORK LIMITED	F6S	UK
03	BioSense	BIOS	SRB
04	Synelixis Solutions	SYN	EL
05	Instituto Pedro Nunes	IPN	PT
06	Fraunhofer IPA	IPA	DE
07	InoSens	INO	SRB
08	Libelium Comunicaciones Distribuidas SL	LIB	ES
09	FastTrack	FASTT	PT

Document Revision History

Date	Issue	Author/Editor/Contributor	Summary of main changes
13.3.2018	1.0	Hatiboglu, Schneider (IPA)	Generation of main document (1.2 will be subject to change)
15,3,2018	1,1	Rubén Hermoso (IPA)	Final Revision

Table of Contents

List of Abbreviations

VFK	Virtual Fort Knox (cloud-platform)
CPS	Cyber-physical system
SDK	Software development kit
API	Application programming interface
MSB	Manufacturing Service Bus (middleware supplied by VFK)
IT	Information technologies
REST	Representation State Transfer
IPA	Institute for Manufacturing Engineering and Automation
GUI	Graphical user interface
JSON	Java Script Object Notation
AGV	Automated guided vehicle

1 Virtual Fort Knox Integration Guide

The following explanation serve as an introduction to the utilization of the *Virtual Fort Knox Platform* in the diatomic manufacturing experiment, in particular the provided middleware Manufacturing Service Bus. It provides an extensive explanation of the REST-API. Additionally, client libraries for the connection via WebSocket for most common programming languages that exist. It is recommended to use these client libraries while developing software or smart objects in context of Virtual Fort Knox. Unofficial alpha version of clients for OPC-UA and MQTT are available but currently unsupported.

All client libraries are currently not publicly available. If this status changes, this document will be updated accordingly. Winners of the open calls will be provided access to the client repository, the associated FAQ as well as access to the Virtual Fort Knox infrastructure at the beginning of their pull experiments to start implementation.

1 Before starting

Understanding the provided cloud-platform for manufacturing is required to develop or adapt smart objects and applications. This section provides a generic introduction to the functionality of the cloud platform from an application developer's perspective. Information specific to the manufacturing experiment will be provided in later half of the document.

1.1 Introduction to the Virtual Fort Knox Cloud Platform for Manufacturing

Virtual Fort Knox (VFK) is a federative platform for the manufacturing industry developed by the Fraunhofer Institute for Manufacturing Engineering and Automation (Fraunhofer IPA). It will offer manufacturing companies an IT strategy that is cost-efficient, agile and scalable. Companies will be provided with efficient access to Industry 4.0 software solutions which are independent of manufacturers, in order to make advances in the digitalisation and optimisation of their production processes. Figure 1 depicts the concept of VFK. It is based on a cell structure and follows the "security-by-design" principle. Each VFK cell is a securely encapsulated environment for service users and service providers.

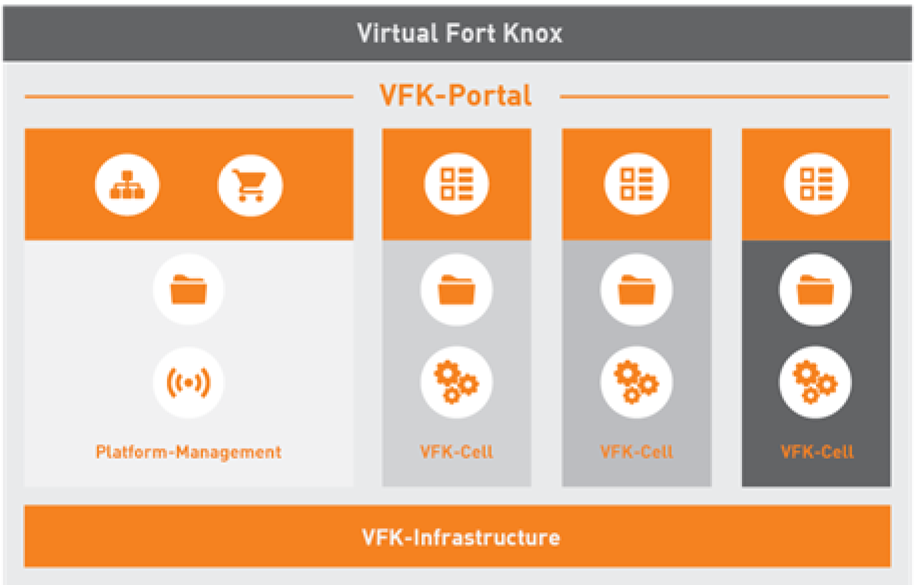

Figure 1: Virtual Fort Knox Concept

Figure 2 illustrates the VFK architecture. The physical devices are on the show floor level and are called Smart Objects (e.g. equipment or cyber-physical systems (CPS)). Due to the large number of communication protocols, a middleware is used for the communication. This middleware is called Manufacturing Services Bus (MSB) and is described in chapter 1.3. To communicate with IT services running in the cloud these services are also connected to the MSB. Following a service-oriented approach, services can be aggregated to new services that provide new functionalities. Economically relevant will be the opportunity for Independent Service Vendors (ISVs) to offer their services in the VFK marketplace where the end users will be able to purchase them. E.g. an equipment manufacturer can offer some special services for its equipment and the customers can purchase the services which they need. From the technical side VFK offers a software development kit (SDK) which is available in most common programming languages. Applications can be hosted in the cloud infrastructure in form of virtual machines and docker containers. Additionally, the platform provides a flexible middleware as abstraction layer between components which allows changes to the flow of information at run-time.

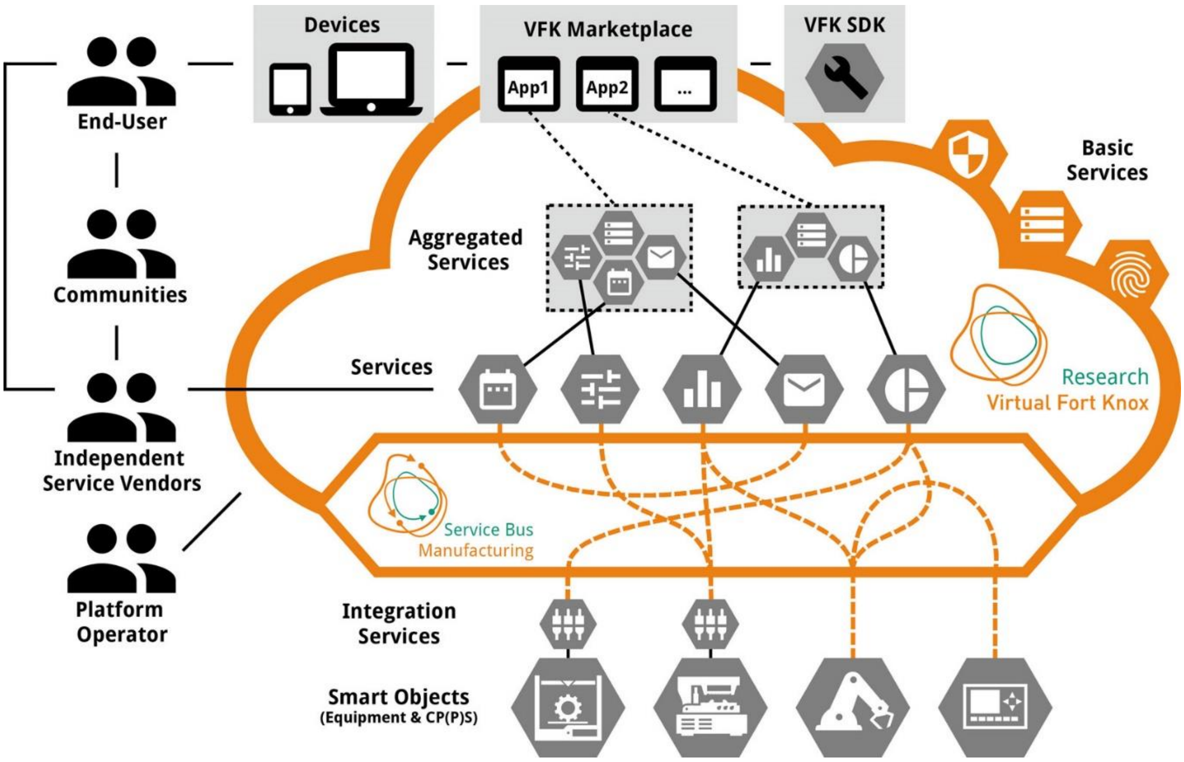

Figure 2: Virtual Fort Knox Architecture

1.2 Cell Concept of Virtual Fort Knox

Each organization, using VFK, operates within an encapsulated environment, referred to as a *cell*. These cells can be publicly hosted or run on a local machine in the network infrastructure of the organization, as shown in Figure 3. Data cannot be transferred between cells, unless applications or smart objects are specifically set up to do so (e.g. a bridge interface). Generally, it is advised to use the middleware accompanying VFK to set up communication within each cell. Each organization with its own cell may consist of multiple users. Users can deploy virtual machines or will be able to download preset software from the shop. Such preset software will be available in the centralized shop, which operates similar to other app-shops. By default, components which are deployed by a user are only visible, and therefore useable, to him. However, they can be made visible to other users within the organization as well.

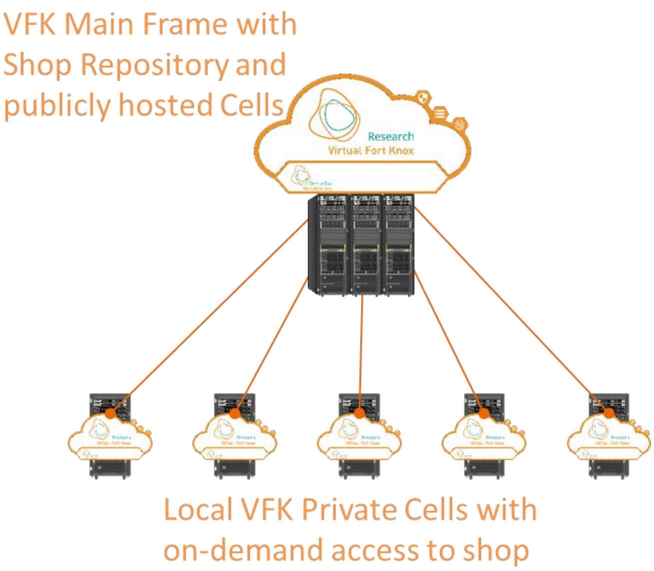

Figure 3: VFK cell concept with locally hosted cells and publicly hosted cells in the main infrastructure

1.3 Middleware - Manufacturing Service Bus

The Manufacturing Service Bus (MSB) enables a fast and low-effort integration of smart objects or IT-Services, because it provides the integration between various communication protocols such as RESTful Web Service or WebSocket API and various communication standards, for instance OPC UA. For these purposes the MSB provides common interfaces which allow the communication between smart objects and IT-services. The communication process is shown in Figure 4. Data are transferred in an encrypted channel. All send data are transformed to a common data format which ensures that all communication participants can communicate with each other. Received data are added to a queue to allow communication between communication partners with different communication cycles. The routing of the data is done using so-called integration flows, which allow the users to flexibly define where data is forwarded to. Integration flows can be defined without programming skills in the web-based user interface of the MSB. Alternatively, a RESTful API is also available for automation purposes.

Figure 4: Communication Process of the Manufacturing Service Bus

1.3.1 Communication pattern

The communication follows the pattern depicted in Figure 5. At the start of the communication, the client registers itself with the appropriate interface (depending on the used communication protocol). When registering, the client sends its self-description, so that the MSB knows who is registering and what capabilities are available and which data can be expected. After registration is done the client can send data by throwing an event that contains the data. To send data to the client the MSB calls the appropriate function on the client with the data as function parameters.

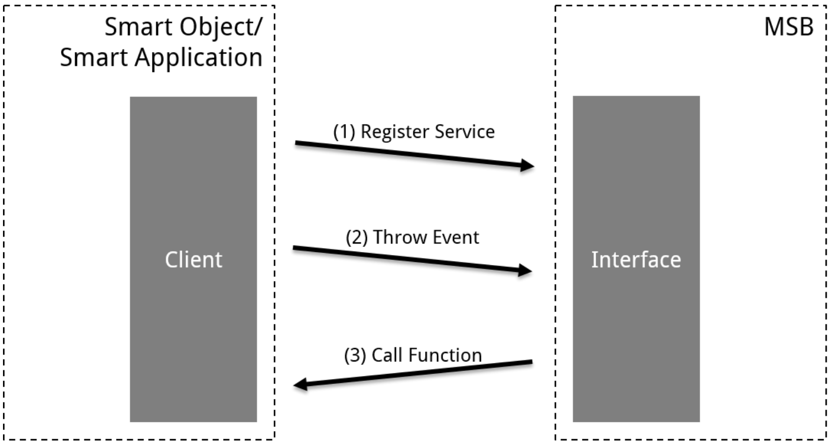

Figure 5: Communication Pattern of the Manufacturing Services Bus

1.3.2 Self-description of services

Each service has a self-description describing its characteristics. The structure of the self-description is shown in Figure 6. A service is classified as an Application or as a Smart Object and can be identified by its unique UUID. Data that is send by a service is described as events. Data that is sent to the service can be received as function. Functions can be used to trigger capabilities of the service by internally mapping the incoming function to a callback function in the service-specific code. Such a callback function can trigger return events as well.

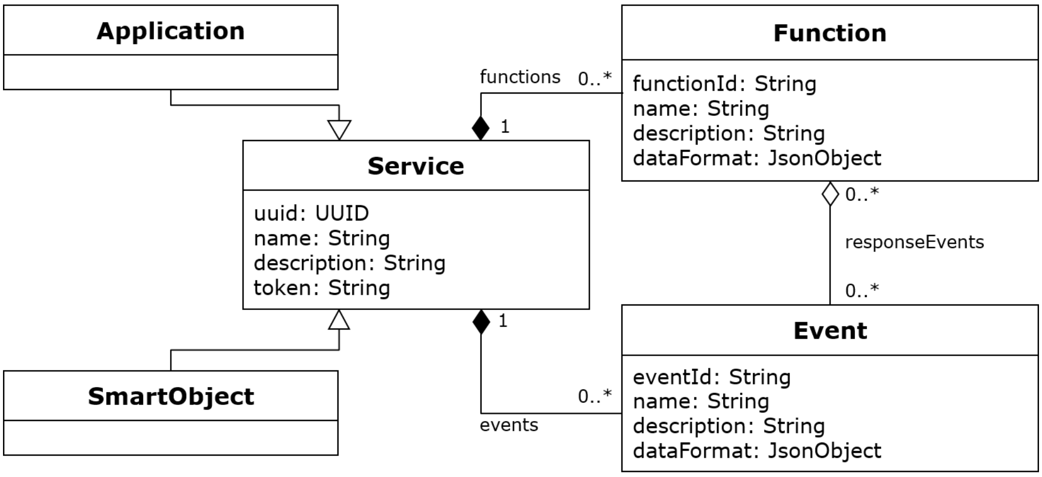

Figure 6: Self-description of services

Once a component (smart object or application) is registered at the MSB, the MSB can be configured to transfer information to and from the component automatically by manual configuration via a graphical user interface (GUI). A simple example for the communication pattern is shown in Figure 24. To achieve the shown information exchange, three main configuration steps have to be completed: selection of the two components, selection of the corresponding event and function and finally mapping of output data of the smart object to the input parameter of the function of the application. The data emitted by the smart object is attached to the event as a JSON string. The data is then mapped to the corresponding input parameters of the function of the smart application and wrapped in a JSON string again.

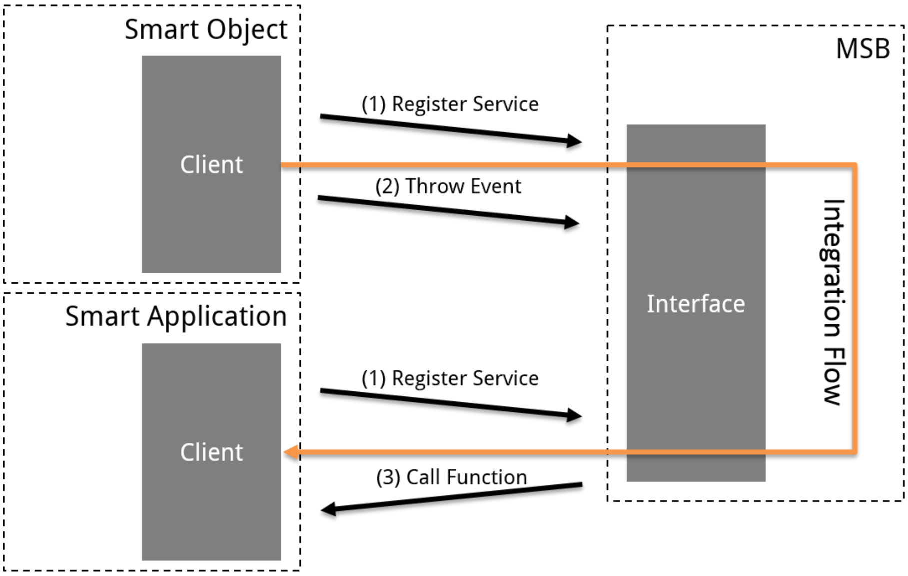

Figure 7: Exemplary pattern for data transfer of smart object to an application

For WebSocket communication ready-to-use client libraries in several programming languages are available, which developers of smart objects and applications can use to connect their own product to the MSB.

1.3.3 Meta data format

The data format of events and functions is shown in Table 2. It is based on the OpenAPI Specification 2.0 (fka Swagger Specification) derived from the JSON Schema for programming language independent definitions of data format. The complete OpenAPI specification that is used for the Swagger-UI as well as for the applications JSON definition can be found under https://github.com/OAI/OpenAPI-Specification.

Error! Reference source not found. shows an example for the specification of a complex object.

Table 1: Meta data format of the Manufacturing Service Bus

Common Name	Type	Format	Comments
Integer	integer	int32	signed 32 bits
Long	integer	int64	signed 64 bits
Float	number	Float	
Double	number	double	
String, Short	string		
Byte	string	Byte	
Boolean	boolean		
Date	string	date-time	As defined by date-time - RFC3339

Common Name	Type	Items	Comments
Array, List, Set	array	<items>	

Common Name	Type	Properties	Comments
Model	object	<properties>	

Common Name	$ref		Comments
Reference	#/definitions/<Model>		

Listing 1: Sample of a complex object specification

```
{  "dataObject":{
     "$ref":"#/definitions/alarm" },
    "alarm":{
      "type":"object","properties":{
        "reason":{"type":"string"},
        "errorCode":{"type":"integer","format":"int32"},
        "machine":{"$ref":"#/definitions/machine"} } },
    "machine":{
        "type":"object","properties":{
          "id":{"type":"integer","format":"int64"},
          "name":{"type":"string"}
        } } }
```

1.3.4 RESTful API

1.3.4.1 Registration

The MSB supports REST to communicate with applications and smart objects. While the WebSocket interfaces allow to infer the connection state of connected smart objects and applications, the REST interface does not allow this. The reason for this is the stateless nature of REST interfaces.

The MSB's REST interface can be reached at port 8083, regardless of the cell the MSB resides in. The OpenAPI specification can be found at the same port under the path /swagger-ui.html. That means, that for MSB reachable under the URL msb.vfk.de (not a real valid URL), the REST API would be reachable under the URL msb.vfk.de:8083, with the API documentation available under msb.vfk.de:8083/swagger-ui.html.

The Swagger-UI is a documentation tool for APIs that provide an OpenAPI specification. The MSB provides such a specification. While the OpenAPI specification should provide enough information on the interface for basic use cases, it does not provide enough information for complex applications. This documentation serves to supplement the Swagger-UI specification.

As shown in Figure 8, the self-description described in Figure 6 has been extended to support the integration of the endpoints of a RESTful application. A RESTful application can be registered to the MSB in two different ways. One possibility is to use the MSB GUI and the other one the REST API of the MSB.

1.3.4.1.1 Registration with REST API

The REST API has two endpoints: One for the registration of an application (Figure 9) and the other one for the registration of SmartObjects (Figure 10). The self-description of the application that should be registered must be described as a JSON object. Listing 2 shows such a JSON description of a simple application and is meant to serve as an example. The fields contained in the JSON definition are described in Table 2.

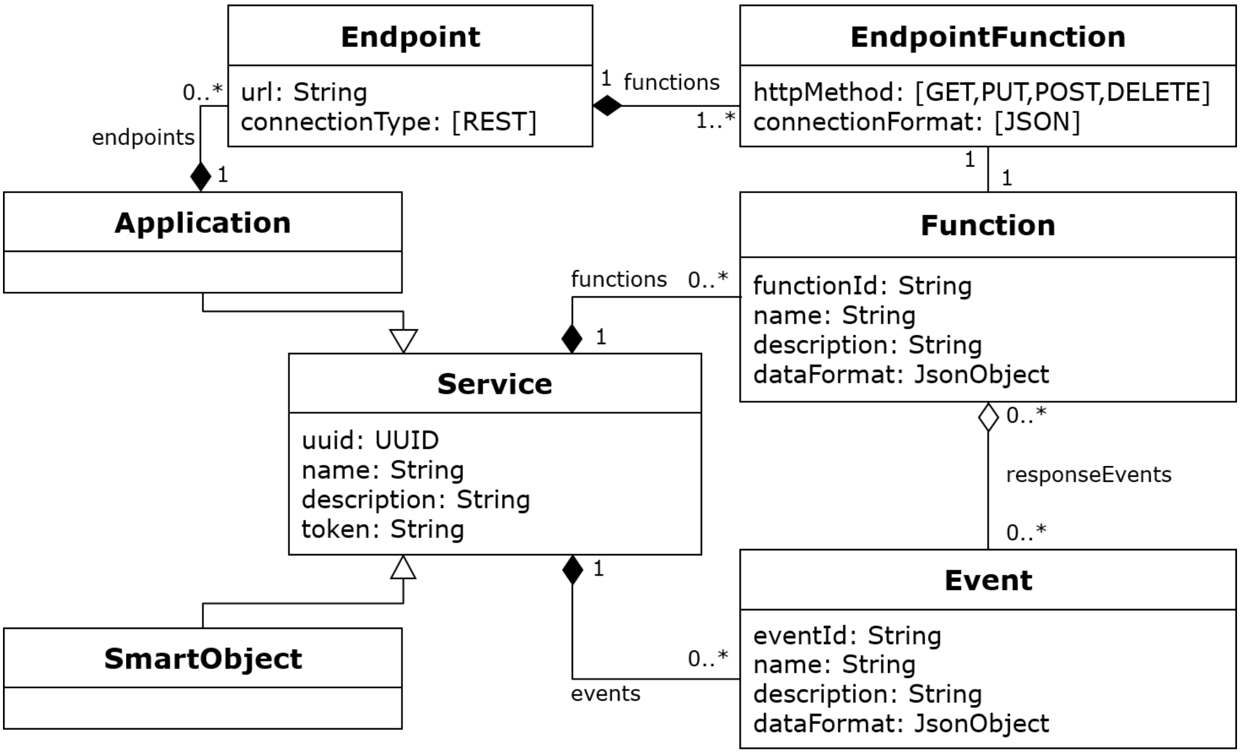

Figure 8: Extended self-description for RESTful API

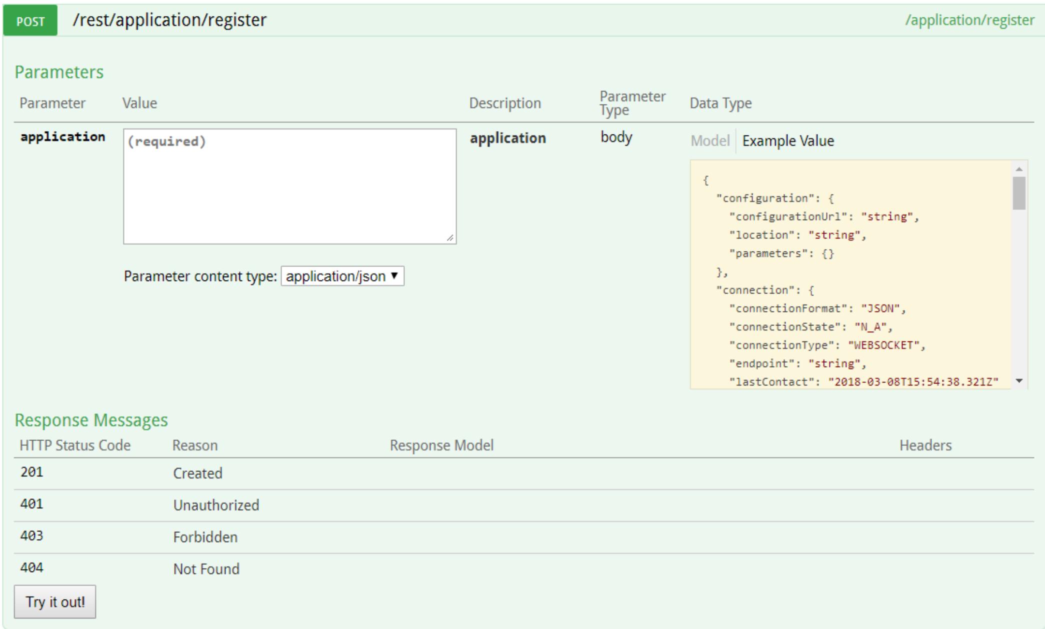

Figure 9: REST endpoint to register application

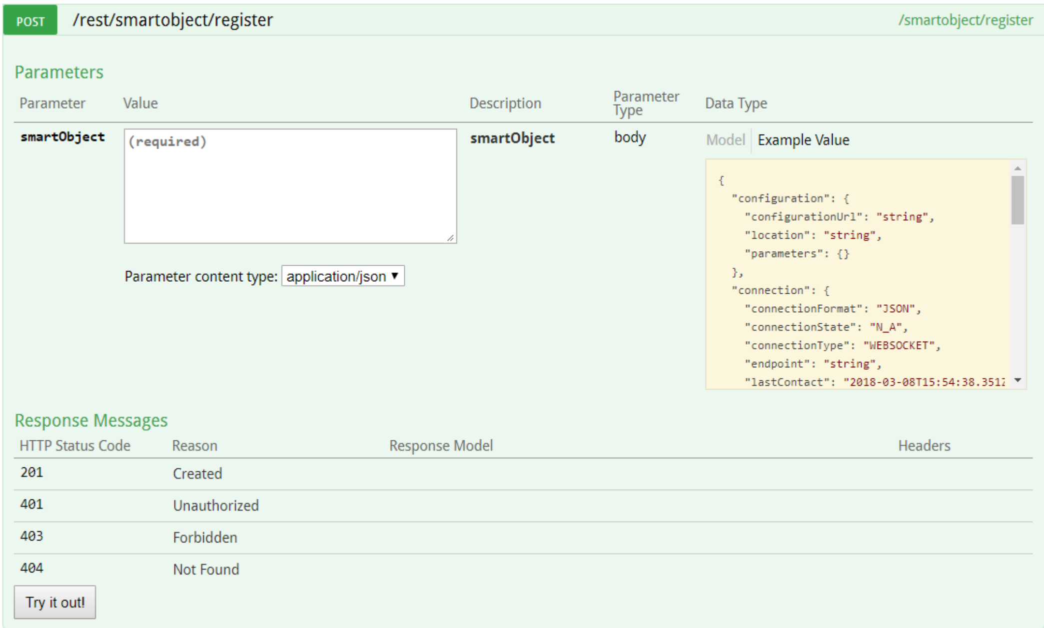

Figure 10: REST endpoint to register SmartObject

Listing 2: Sample self-description of REST application as JSON object


```
{ "@class": "Application",
  "uuid": "71f747a8-b12e-476a-bdb7-85c68c59c282",
  "name": "System Information",
  "description": "Provides information about a remote system",
  "token":"auniquestring",
  "events": [ {
   "@id": 1,
      "dataFormat": {
       "dataObject": {
         "type": "object",
          "properties": {
            "system": {"type": "string"},
             "name": {"type": "string"} } } },
```

```
        "description": "Displays live information about a remote system.",
        "eventId": "live-information",
        "name": "Live System Information" }],
  "functions": [ {
   "@id": 1,
        "functionId": "/system-information",
        "name": "System Information",
        "description": "Provides System information",
          "dataFormat": {},
             "responseEvents": [1] }],
         "endpoints": [ {
             "url": "http://www.example.com",
             "connectionType": "REST",
             "functions": [ {
                 "httpMethod": "POST",
                 "connectionFormat": "JSON",
                 "function": 1 } ] } ] }
```


Table 2: Description of fields contained in JSON object of a self-description

@class	Can either be Application or SmartObject, depending on which type of object is described.
uuid	A unique identifier for the application. A valid identifier can be generated under https://www.uuidgenerator.net/version4.
name	The name of the application that will be displayed on the MSB GUI.
description	A textual description of the application
token	A token that will be entered in the MSB GUI to complete the registration of the application
events	A JSON description of the events that the application provides to the MSB. The defined events have to provide in incremental numerical id, as well as a unique textual event id. The numerical id is used to refer to the event as a response event in a function definition, while the textual id is used to route the events information within integration flows. The dataFormat follows the OpenAPI specification 2.0. The outer object of the data format must be called dataObject and must be of type object. Everything within the dataObject is optional and can be defined by the developer.
functions	A JSON description of the functions that the application provides to the MSB. The defined functions have to provide in incremental numerical id, as well as a unique textual function id. The numerical id is used to refer to the event as a response event in a function definition, while the textual id is used to route the function information within integration flows. Additionaly, the functionId is the path that is attached to an endpoint. If a function can be reached under the url www.example.com/someFunction, the functionId must be /someFunction. The numerical ID is used under endpoints to connect endpoint definitions with functions. The dataFormat follows the OpenAPI specification 2.0 and is completely developer defined.
endpoints	A description of the endpoints under which the application can be accessed by the MSB. A URL and a connection type have to be provided. The functions section further describes how the functions can be accessed by the MSB. The function attribute in the specification refers to the @id attribute of a function defined in the outer application description scope.

1.3.4.2 Alternative Registration via MSB GUI

A REST Application can be also added using the MSB GUI. Therefore, you must select the APPLICATIONS tab and press the "+" Button in the left corner. In the pop-up you must press "Create App" and the "Manual app creation wizard" as shown in Figure 11 will appear. In Step 1 an UUID is automatically generated and you can enter other basic information like the name and the description of the application.

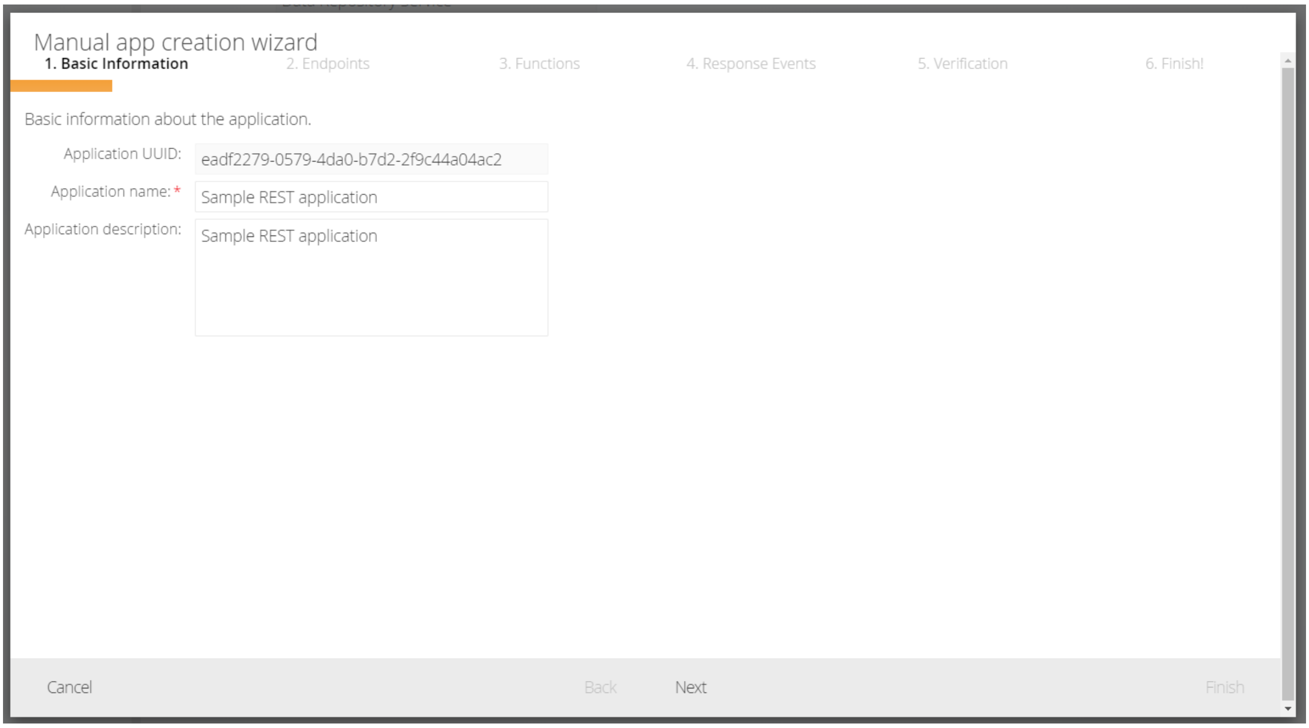

Figure 11: Manual app creation wizard – Step 1: Basic Information

In Step 2 (see Figure 12) the URL of the REST endpoints must be defined.

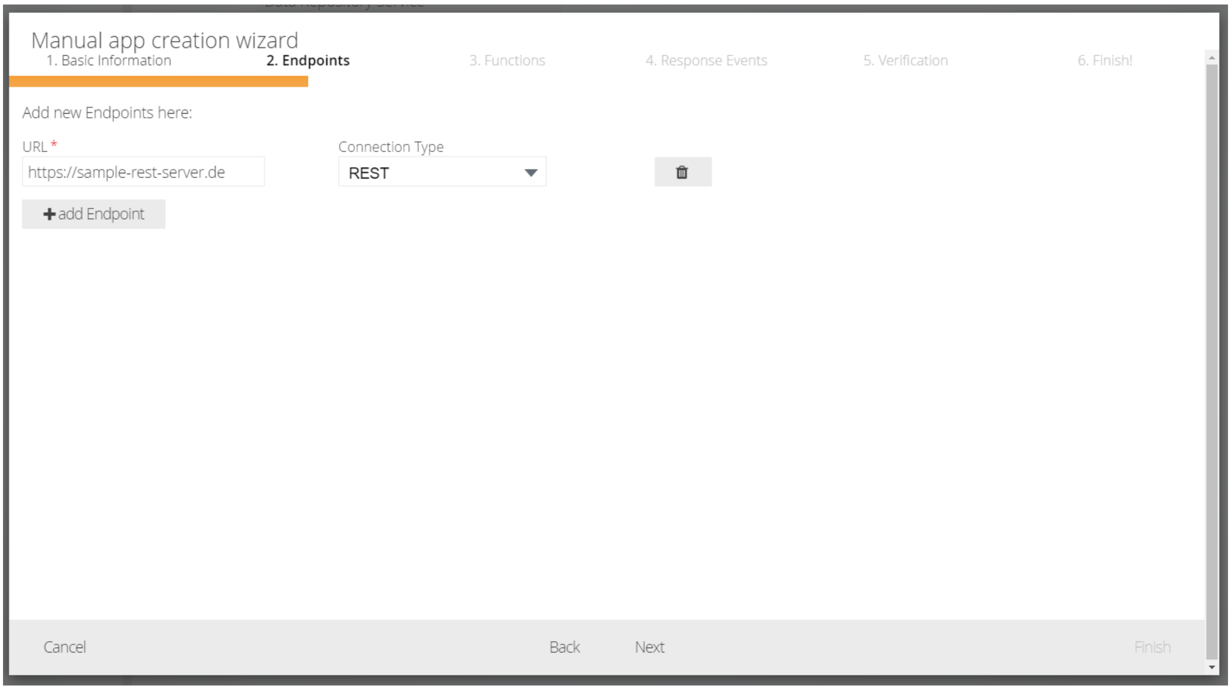

Figure 12: Manual app creation wizard – Step 2: Endpoints

In Step 3 (see Figure 13) for each defined REST endpoint functions can be defined. The path can contain parameters in the form of {parameter1}. This data format of the parameters must be defined in the "Request Schema" as described in section 10. If the function is called via an integration flow the parameter can be mapped from the triggering event. The parameter will be replaced with the value of provided by the event and the REST call will be executed.

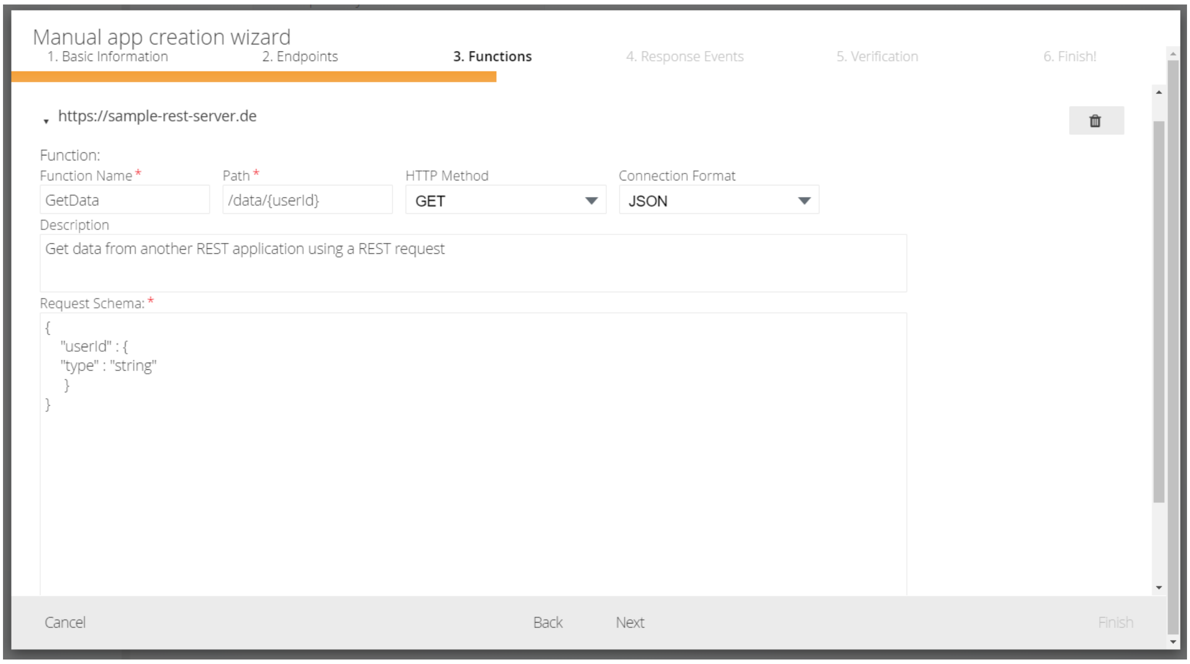

Figure 13: Manual app creation wizard – Step 3: Functions

The response of the executed REST call will be sent as a response event. Response events for the functions can be defined in Step 4. In the "Response Event Schema" you must describe the data format of the information that will be responded by the REST application (see section 1.3.3).

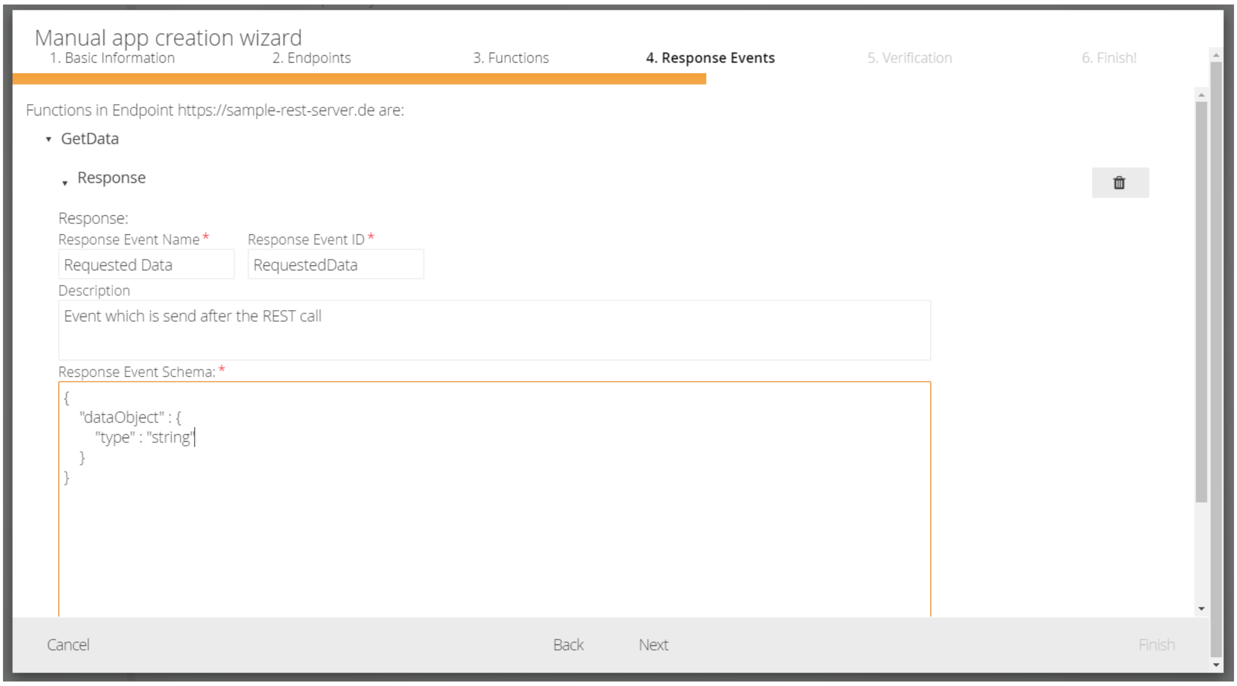

Figure 14: Manual app creation wizard – Step 4: Response Events

The last two steps allow verification of the input and completion of the wizard. After that the application will automatically appear in the applications list.

1.3.4.3Send data

Once your application has been registered and verified to the MSB it is ready to receive information via a function call and to send information to the MSB via an event. An event is send as JSON object with the fields described in Table 1.

Table 3: Description of fields contained in JSON object of an event

uuid	UUID of the Service that sends the event.
eventId	Id of the event as definied in the self-description of the Service.
priority	Priority with which the event is to be processed by the MSB
dataObject	JSON object that contains the data of the event.

The event is then sent to the REST endpoint (/rest/data) shown in Figure 15.

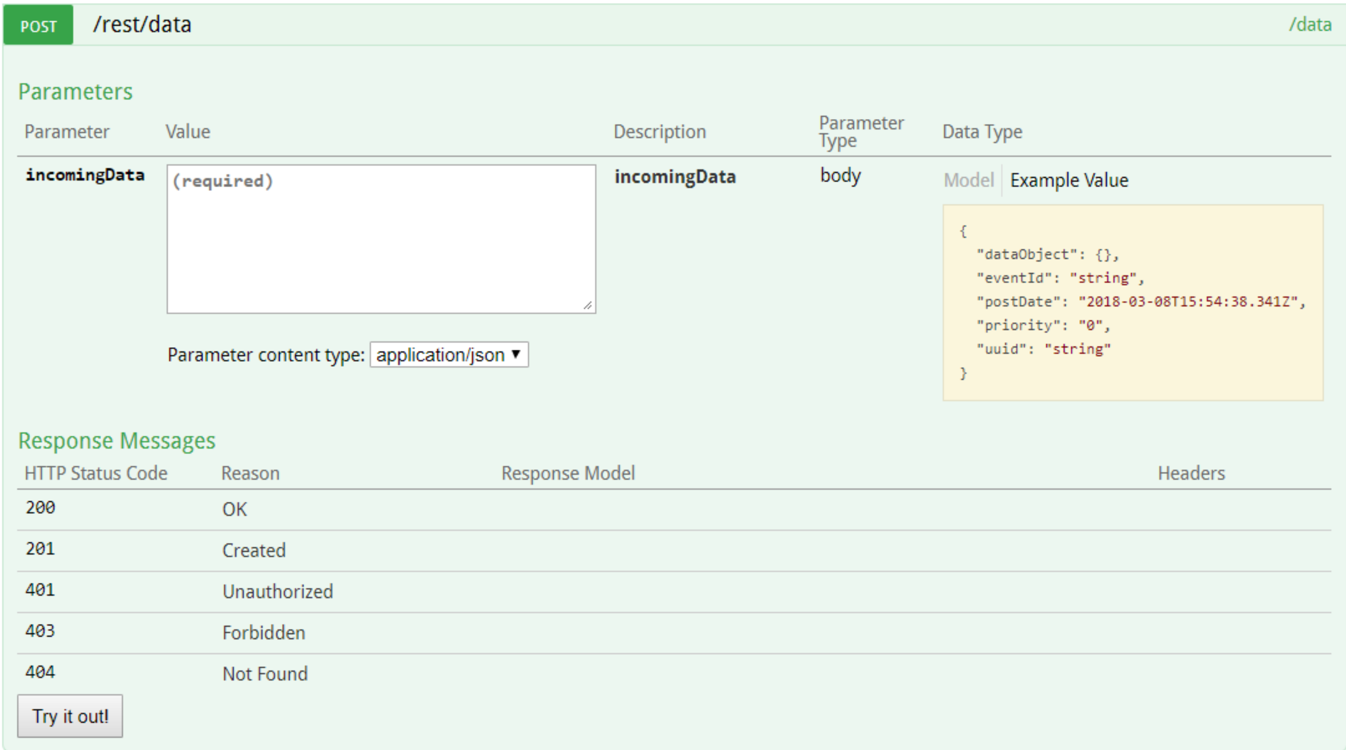

Figure 15: REST endpoint to send data

2 Step-by-Step Interaction within the Manufacturing Experiment

An full integration guide to the complete setup of the manufacturing experiment would be too extensive to describe in this section. Instead an overview over all provided components will be given and the general steps to set up new integration will be provided. Be advised that this information is subject to change. In general all components of the experiment can be replaced or extended by similar components with improved functionality. Such components might be developed in the open call's, associated with this experiment.

Subject of the base version of the manufacturing experiment, without extension through SME activities in the open calls, is the setup of automated 3D-printing equipment. Figure 16 shows all components involved. Two applications (software with no physical counterpart) are hosted in the cloud environment of VFK. They are shown in the upper section of the figure. All other participants are smart objects which physical functionality. In the figure, they are shown below the middleware (orange bar). The smart objects are equipped with a software component which provides the connectivity to the cloud (shell). In some cases this software component also contains the objects logic control. All components communicate with the middleware via WebSocket over an access point which supports ethernet for stationary smart objects and WLAN connectivity to the cloud-based middleware.

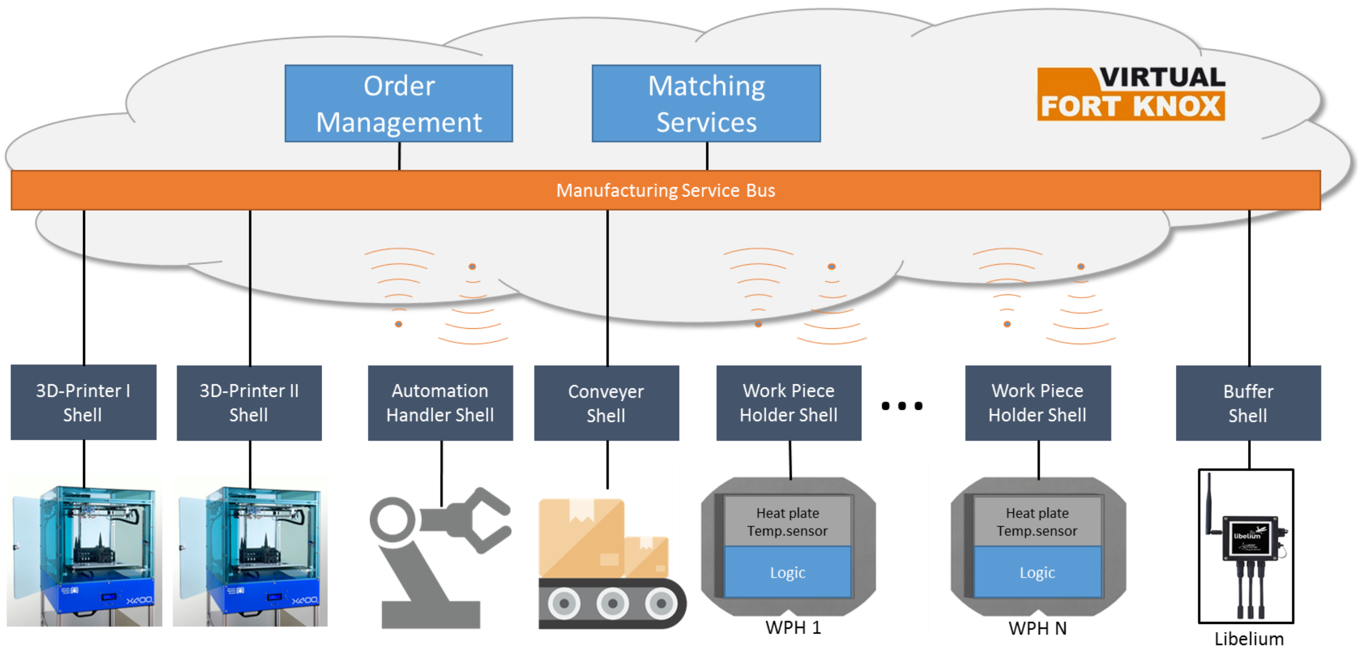

Figure 16: Overall Setup of the Manufacturing Experiment, including all basic Components

The basic purpose of the participants is:

Order Management: Storage of incoming order and initiation of manufacturing process based in first-in-first-out principle.

Matching Service: Supplies feedback about which process resource is suited to process an order (currently limited to 3D-printing) and serves as entry point for new orders.

3D-Printer I / II: Process modules of the experiment. The printers differ in printing capabilities (e.g. two-material printing vs. single-material printing). Their shell provides slicer functionality for processing STL-files to executable machine code, as well as status update of the printing resource for external components.

Work Piece Holders: Stored in the buffer (see below). Basic functionality is transport platform of the product, as well as heating of the process area when externally powered in a printer (functionality improves process quality). Their shell allows feedback and setting of current surface temperature (external power source required for heating).

Automation Handler: Used for on-demand transport of work piece holders between buffer, printers and conveyor belt in fenced area. Functionality provided through standard functions with input parameters (e.g. buffer slot to pick from).

Conveyer Belt: Used to deliver work piece holders from inside the equipment to pick-up position for human operators or (automated guided vehicles) AGVs. Transport process is internally controlled, based on surface temperature of the work piece holder, so ensure safety of human operators (danger of burns). Its shell provides feedback if new work piece holder can be transferred to pick-up point.

Buffer: Storage rack for work piece holders equipped with sensor. Smart shell based on Libelium's platform supplies feedback about filled buffer slots as information for the automation handler.

2.1 Login to Manufacturing Service Bus (MSB)

Winners of the open calls will be provided with access information to the Cloud Platform and the middleware. This information includes:

- User Name
- Password
- Access URL for MSB

Access the URL of the MSB, extended as follows: https://YourMsbUrlGoesHere/msb/gui. You will be greeted by the screen shown in . Please select your preferred language and insert your password and user name.

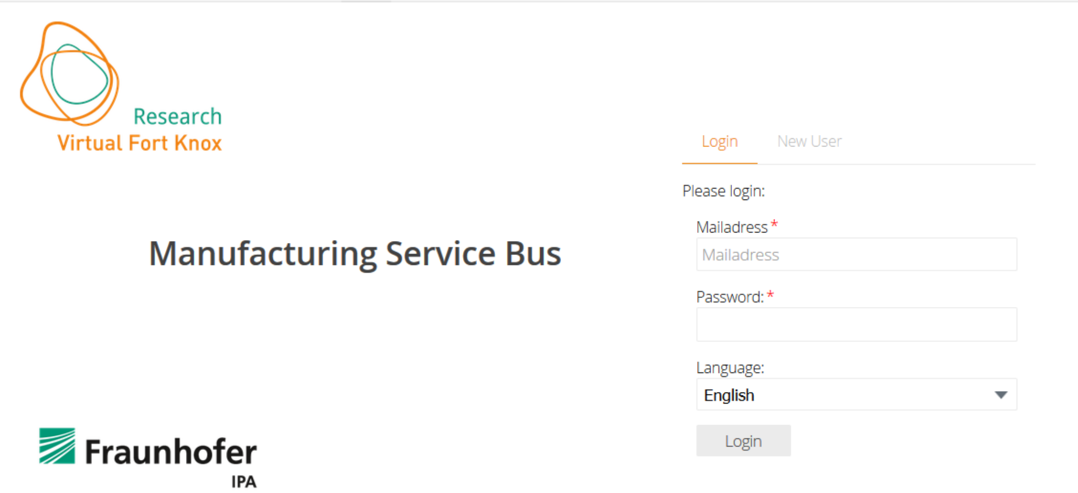

Figure 17: Login Screen of Manufacturing Service Bus

2.2 Registration and Verification of Component

Upon first boot up of a service, the service connects to the MSB and provides a self-description (more on how to achieve this and details on the self-description in section 1.3.2). At this point the component is not yet visible to the user, who needs to activate it first. To do so the user has to navigate to the *SMART OBJECTS* or *APPLICATION* tab, depending on the self-description of the component and click the button for new components ("+"), as can be seen in Figure 18. The classification into smart objects or application is based on the self-description, set by the developer. Its purpose is to allow easier distinction for human users and has no further implication beyond that. As a rule of thumb, smart object should contain at least one sensor or actuator.

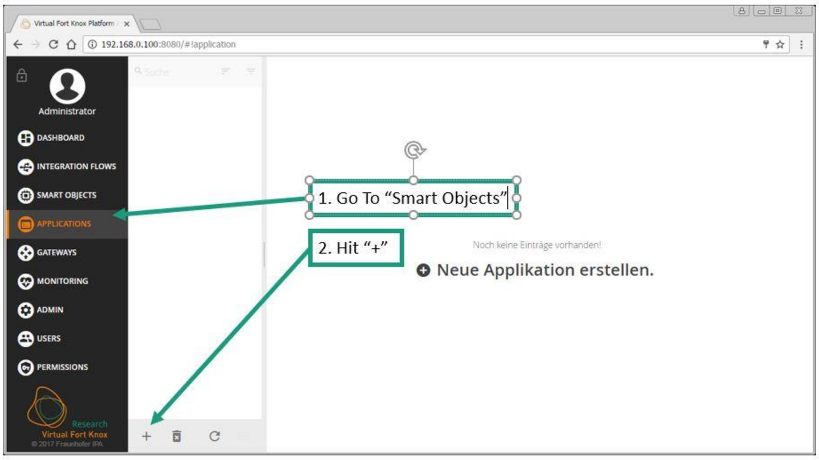

Figure 18: Activate new Component in GUI of MSB

The user is then asked to verify the component by inserting the security token which is part of the self-description and should be provided by the developer. At this stage the user can also decide if the component should only be visible to him or the entire organization (see Figure 19 below). Visibility decides if other user can see the component when they connect to the MSB with their user credentials. If they do not see the component, they cannot set up new information flows including the component. However, they still might be affected, as information flows set up by one user by trigger actions in organization wide visible components. An example might be a component which takes a considerable amount of time to process data and blocks any other request in this state. One user may find the component permanently blocked when setting up his information flow without any capability to identify who is blocking the component, since he cannot see the integration flow of another component which is invisible to him.

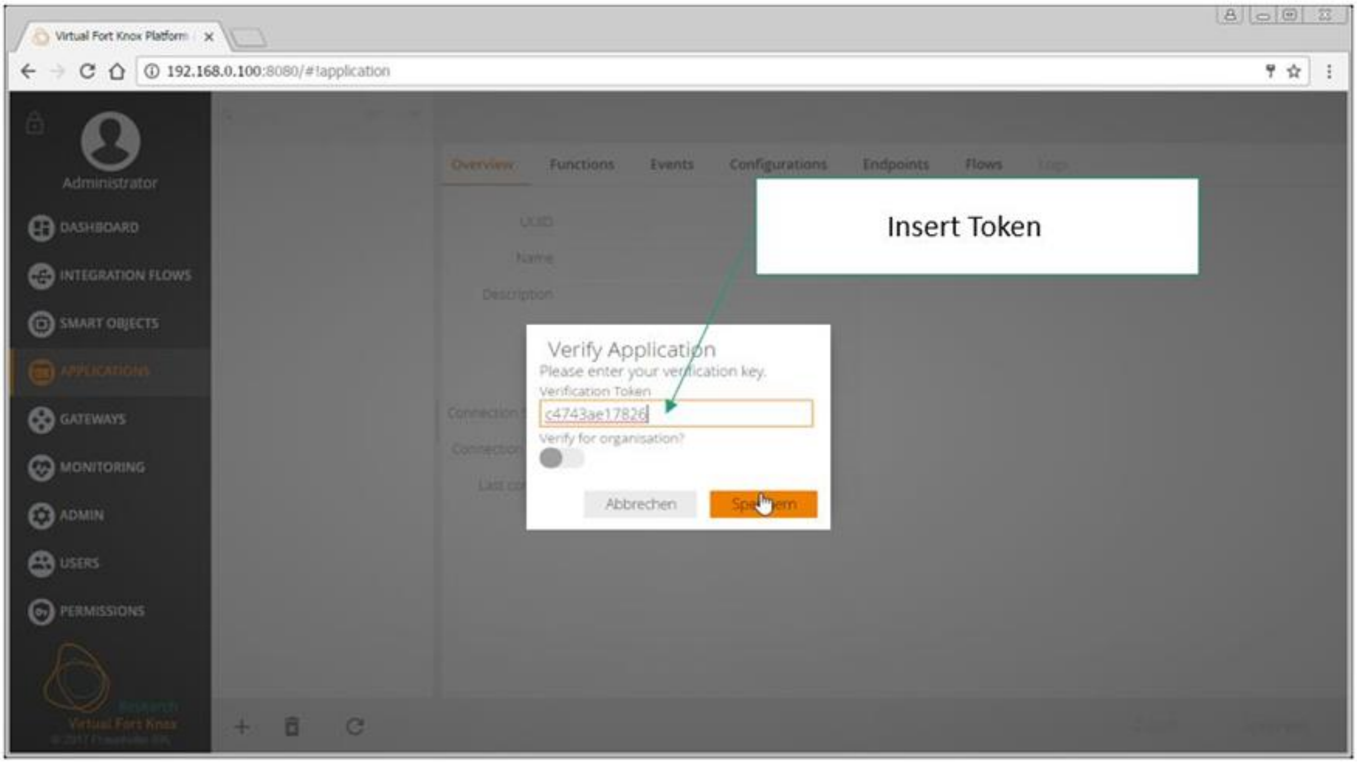

Figure 19: Security Token and Visibility Setting (MSB GUI)

2.3 Reviewing if registered Components

After completion of the registration process, the new component will be listed in the appropriate tab (*SMART OBJECTS* or *APPLICATIONS)* with its self-description. This description includes general information like name and prose description of its general purpose, as well as specific information regarding the outgoing events the component can throw and functions which can be linked to incoming events, as seen in Figure 20. The component is now ready for the modelling of information flows. If the user wishes to delete the component, he can do so at any time by selecting the component in the respective tab, clicking the small garbage bin icon in the bottom left and confirming his decision.

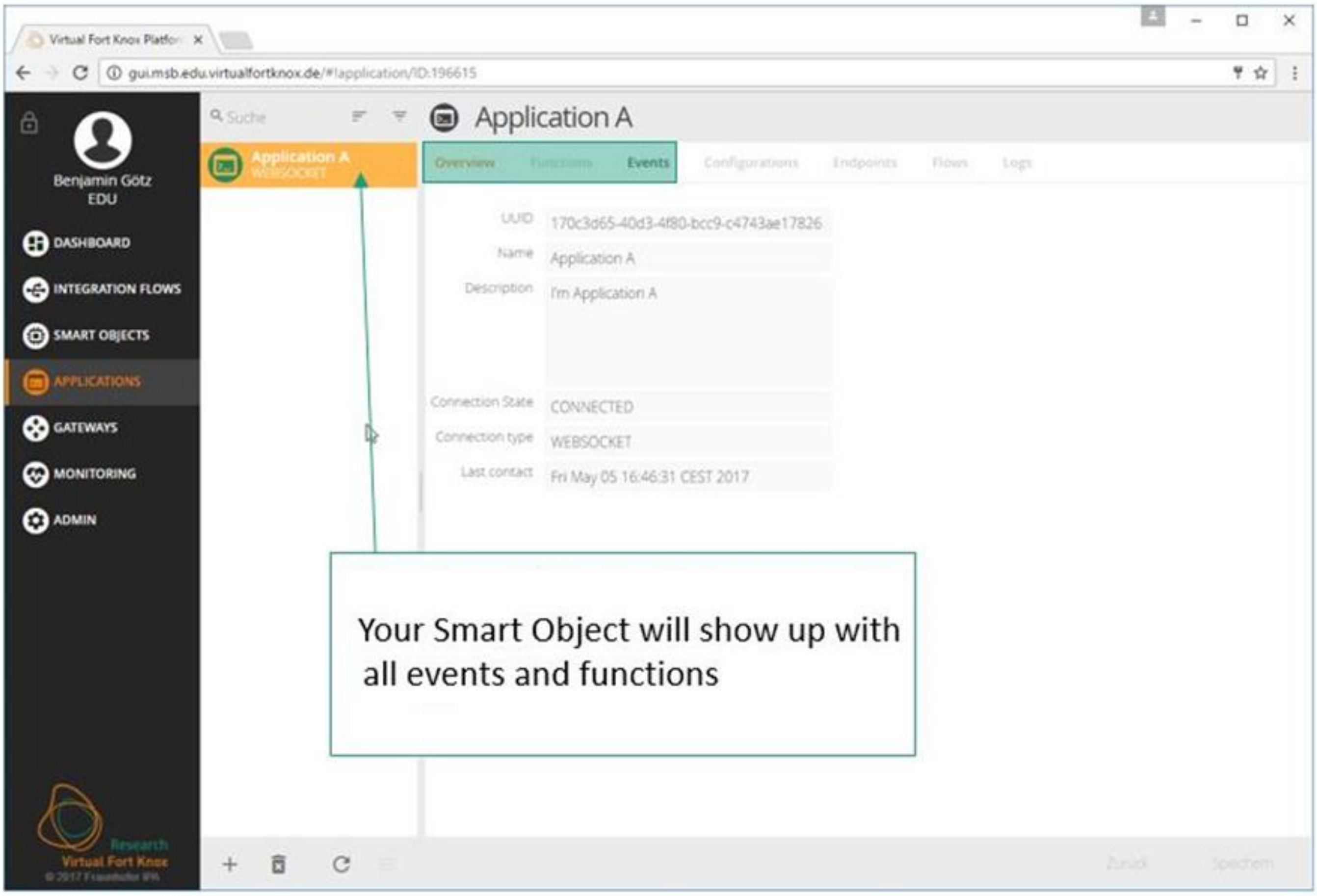

Figure 20: Detailed Information about Component (MSB GUI)

2.4 Setting of Configuration for Components via MSB

Optionally, the *configurations* tab, shown in Figure 21, allows the user to configure internal values of a component remotely via the MSB. This feature is optional and has to be set up by the developer in the program code of the component. If no remote configuration is allowed, the tab name is greyed out and inactive. New parameters can be set by change the value on the right and pushing the orange save button on bottom right. If the new value does not correspond to the required format indicated in the middle row, the changes will not take effect. If a component is modified which is currently offline, the changes will not take effect.

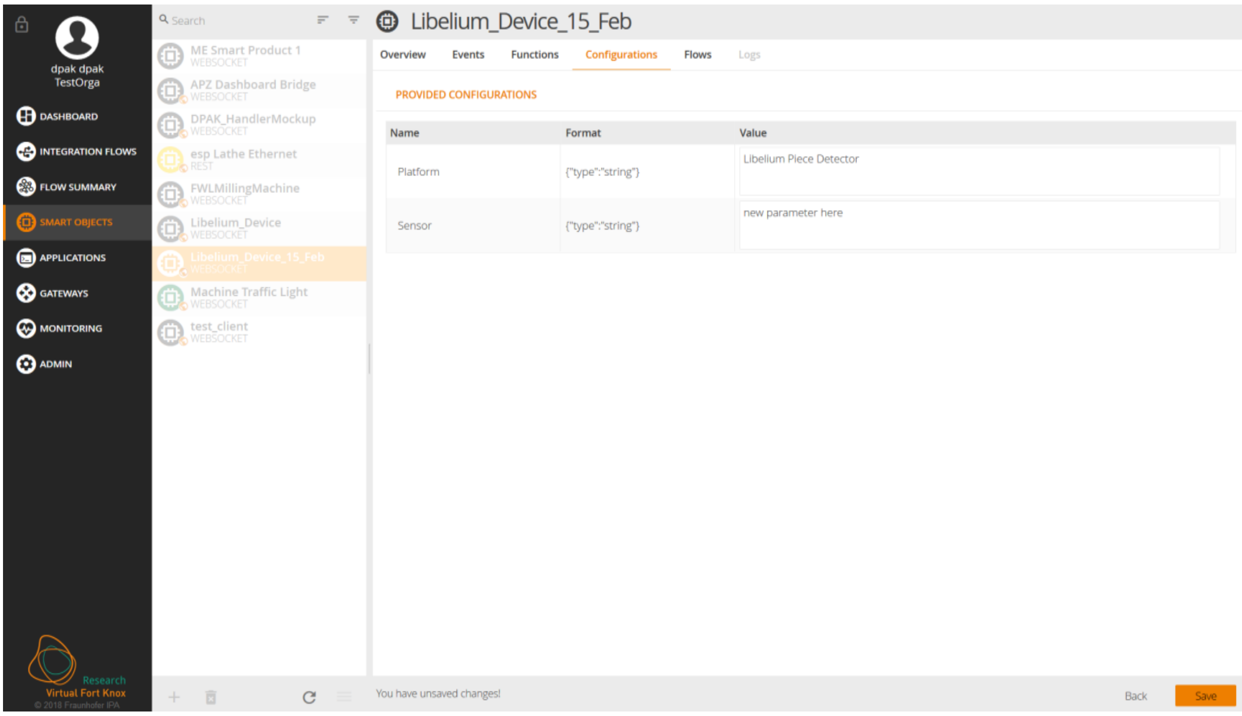

Figure 21: Detailed View on Component - Configurations Tab

The *Flows* tab show all integration flows which the component is currently a member of (see Figure 22). When activating a new component, this tab is empty. Clicking on the arrow on the right lets the user jump directly to the selected flow.

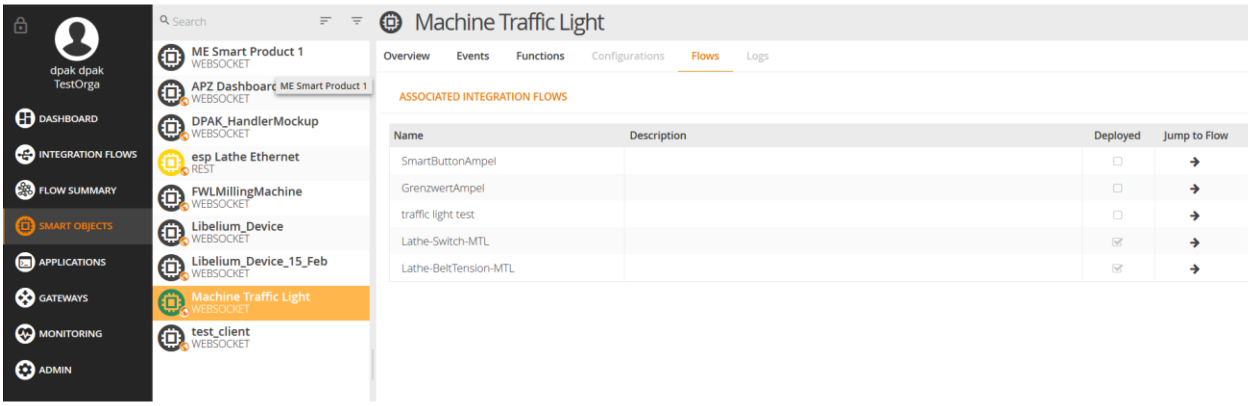

Figure 22: Detailed View on Component - List of all associated Flows (MSB GUI)

2.5 Data Routing with Integration Flows

2.5.1 General information about integration flows

On the one hand, the middleware approach replaces the otherwise required point-to-point connections between the components and reduce the maintenance effort for the IT-personnel. On the other hand, it takes over the function of the event listener for all components and allows the configuration of this function at run time. This simplifies the adaption process in case changes to existing solutions are required (e.g. replacement of an old component). It also simplifies the implementation of new solutions, which rely on data or processing capabilities of existing components and can be configured at run-time without shutdown of the entire system.

To understand how to design components for the use with the MSB, it is useful to first understand, how the user configures his solution based on the available components. Figure 23 shows a simple example for such a solution from view of a user who configures two components to communicate. Once the integration process is complete, the left component can send information to the second one for further processing. The user who implements the solution used a building block concept in a graphical user interface to setup this connection. To achieve the desired information flow, three conditions need to be met, which are represented in Figure 24:

- The components are registered with the MSB (self-description is provided) and activated.
- The information flow is modelled in the MSB by the user.
- The information flow is triggered by the first component in the chain at run-time.

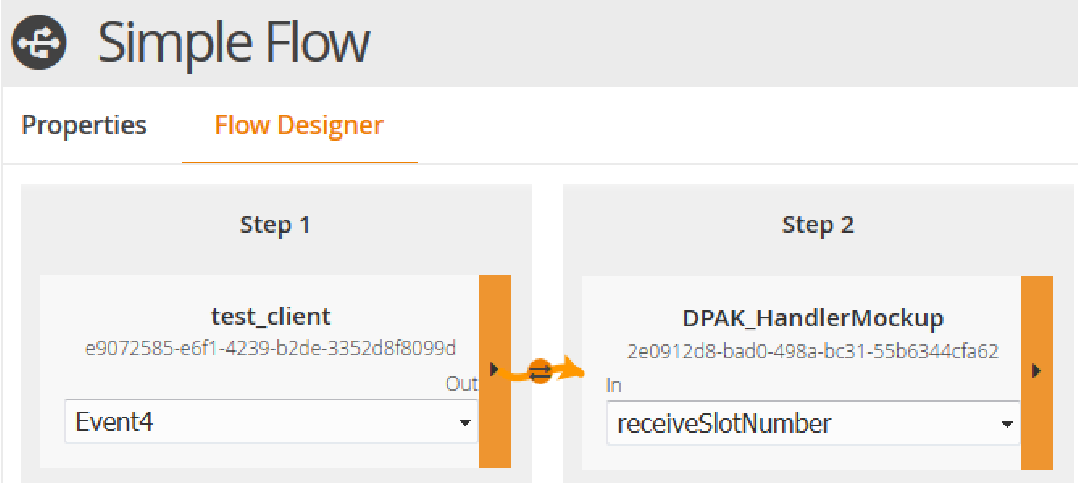

Figure 23: Simple Information Flow modelled in the MSB GUI (only part of GUI is shown)

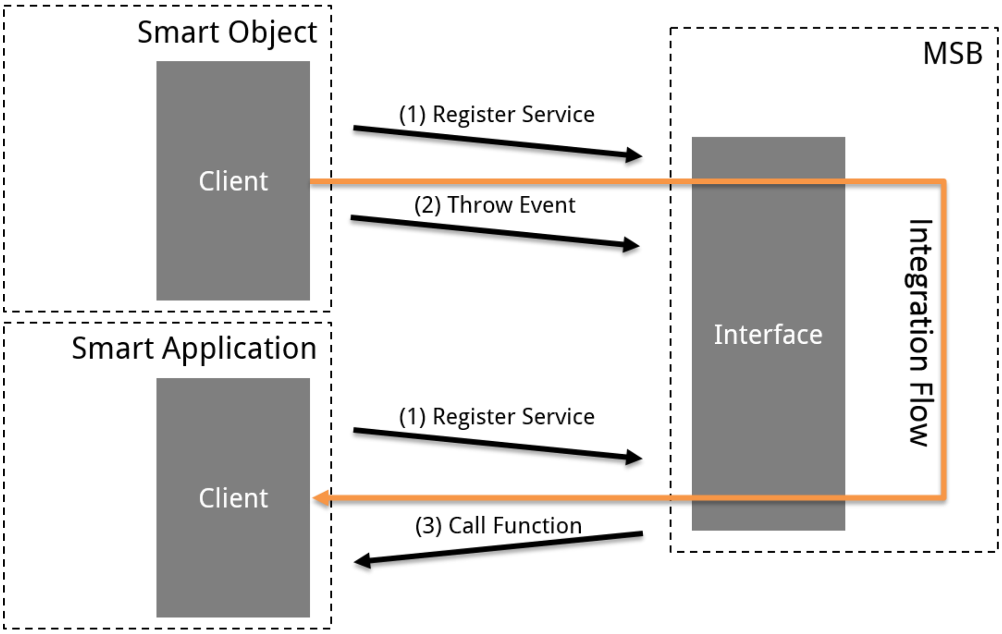

Figure 24: Steps to compete the Exchange of Information

It is important to understand that each component solely communicates with the MSB and does not per se know preceding or subsequent component in the chain of information. This has implications on the design as intrinsic knowhow between components cannot be expected and should therefore be avoided and all contextually required information needs to be available in the self-description or needs to be exchanged in the events.

2.5.2 Modelling of an Information / Integration Flow

The MSB is technically able to map events from a component onto functions of the same component. However, this capability should not be used in general to keep the load on the MSB low. Besides, the latency of the MSB typically exceeds component-internal communication by a large margin due to the underlying IP-based communication.

2.5.3 Initial Creation of an Integration Flow

To build a meaningful information flow, at least two separate components are required. In context of the MSB a model for an information flow are called *integration flow*. Figure 25 shows the first step in creation of such a flow in the *INTEGRATION FLOWS* tab which is initiated similarly to the activation of a component. Once the blank flow is created, it needs to be named, while a description by the user is optional. The modelling can then be initiated by clicking the *Flow Designer* tab (3. in Figure 25).

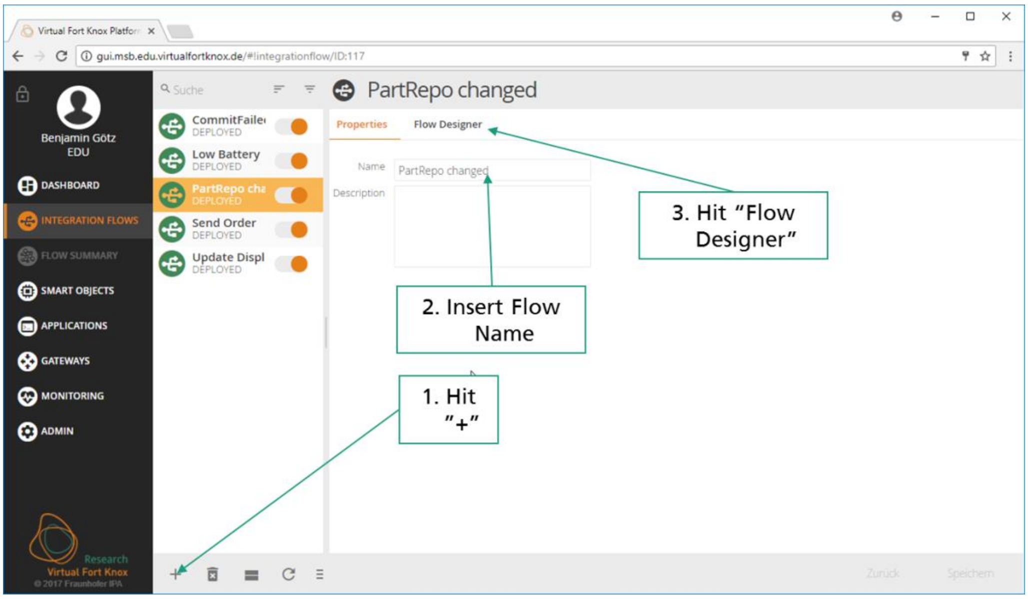

Figure 25: First Step in Creating a new Integration Flow (MSB GUI)

2.5.4 Selection of Components for an Integration Flow

Within the flow designer view, all available components are show on the right side. If the desired component is not shown, the list can be extended to show all smart objects or all applications by clicking on the respective fields. If the component cannot be found, it has not been activated and the steps of section 2.2 need to be competed again. The components, required for the integration flow need to be dragged and dropped onto the main area, as indicated in Figure 26.

After dragging all components into the main area of the GUI, the user should check if all components are positioned in the correct order according to the desired information flow from left (first component in chain) to right (last component in the chain). This is not necessary but advised, as it improves readability.

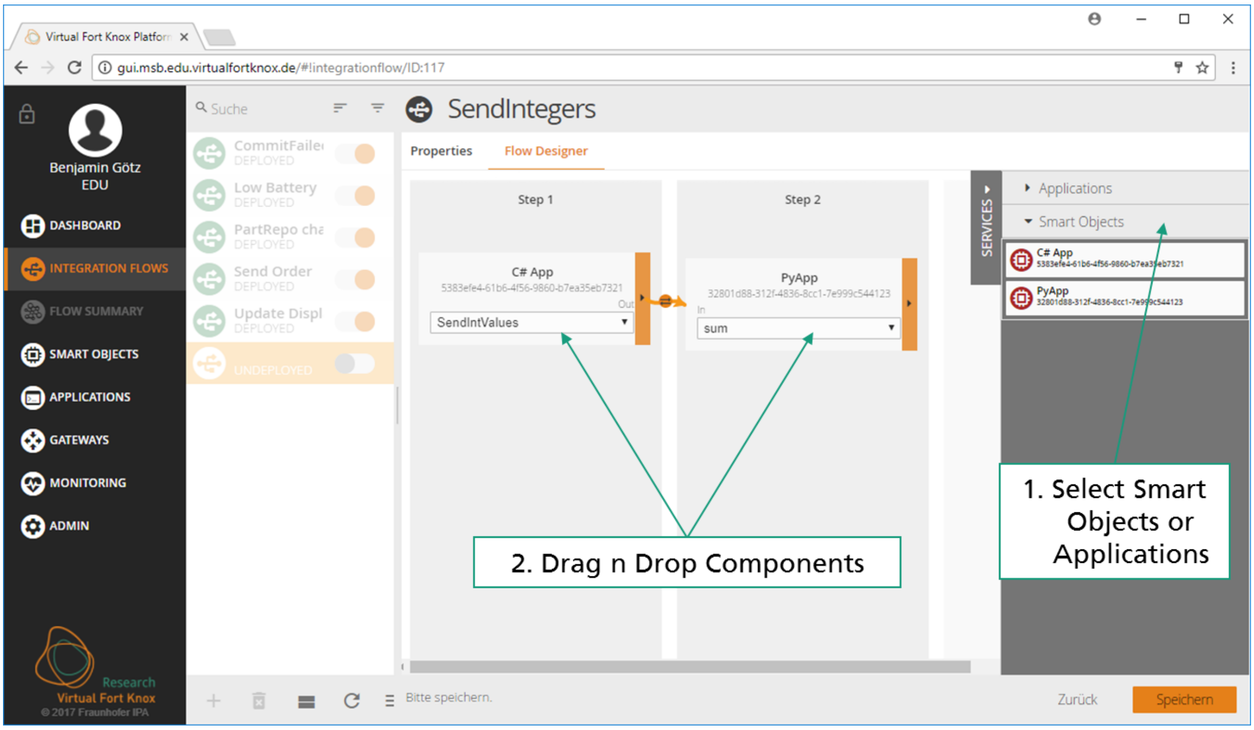

Figure 26: Drag and Drop of Components to initiate the Creation of a new Integration Flow

2.5.5 Selection of required Events and Functions for each Component

Before connecting the components, the user needs to select the appropriate event and function for components he wishes to connect next. This is done by clicking on the drop-down menu for the component and then clicking the desired event or function as indicated in Figure 27. By default, the first event from the list in the self-description is selected for all components. If a component does not supply events, the first function from the list in the self-description is set as default. Selecting a function manually may result in a second drop down menu to appear next to the previous one as can be seen in Figure 28.

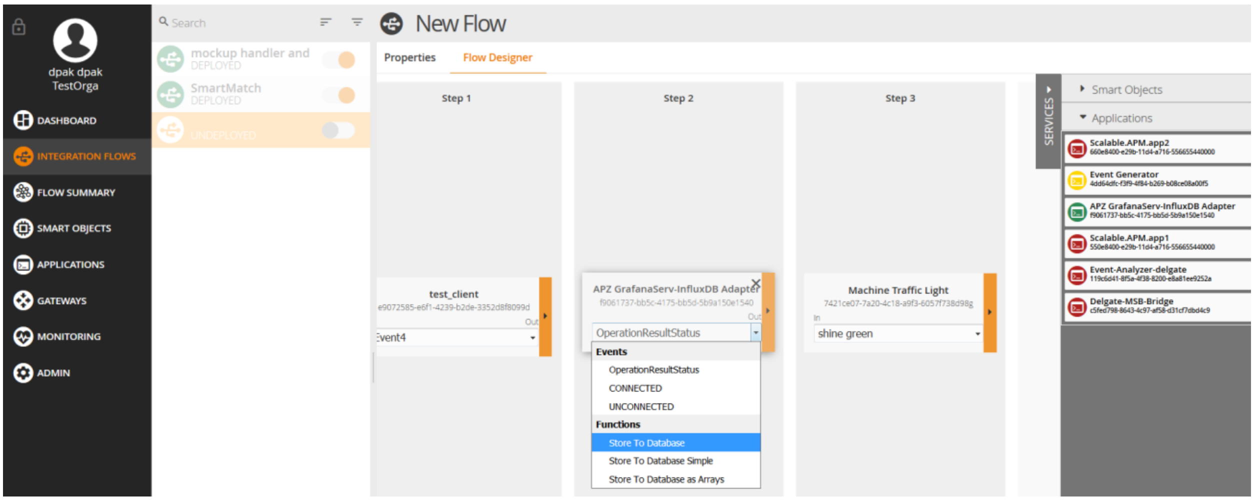

Figure 27: Selection of Event or Function from Drop Down Menu

This implies that the selected function may trigger one of the listed events. The user has to select the desired event from this new list, unless the component is the last one in the chain, where the output event is irrelevant. Due to this behaviour of the GUI it is strongly advised to begin the selection of events and functions at the last component in the desired integration flow.

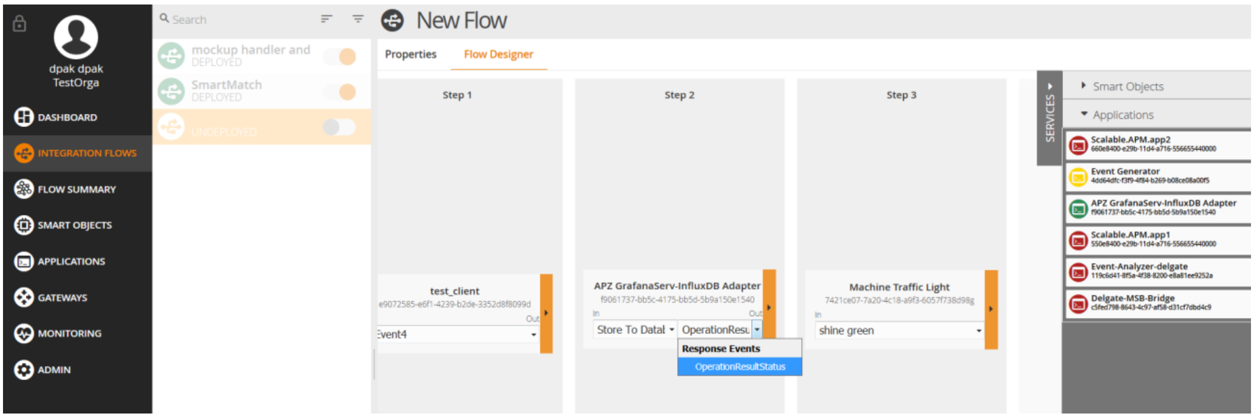

Figure 28: Selection of Return Event based on preselected Function

2.5.6 Linking of Events to Functions

After the selection process the links between the components have to be set up. Links are always initiated from an event towards a function. The user achieves this by clicking on the orange area of a component with the event and dragging the mouse to the component with the function which he wishes to link to. A successful link is indicated by an orange arrow between the two components, where the arrow is directed towards the component which's function should be executed. An example for successful links can be seen in Figure 29.

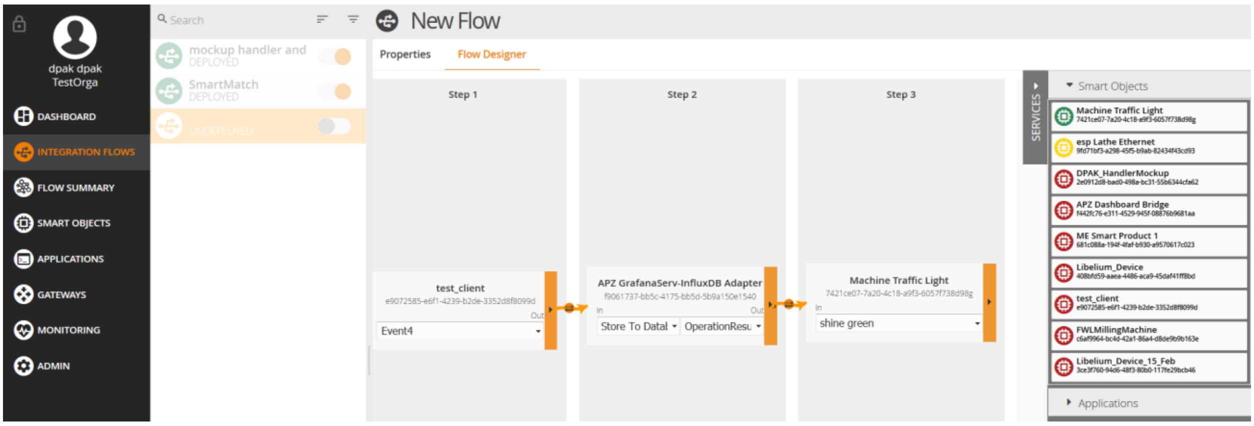

Figure 29: Successfully linked Components

2.5.7 Mapping of Event Data to Function Parameter Inputs

If the selected functions require no input parameters, specific mapping of parameters can be forgone and the flow is ready to be saved. However, in most cases a mapping of data from the event to the input parameters of the corresponding function is required. The user does this by clicking the small orange dot in the middle of each arrow which will result in a similar GUI to Figure 30. On the left all available data from the event is displayed and on the right all input parameters are shown. For every parameter the particular type is displayed as well.

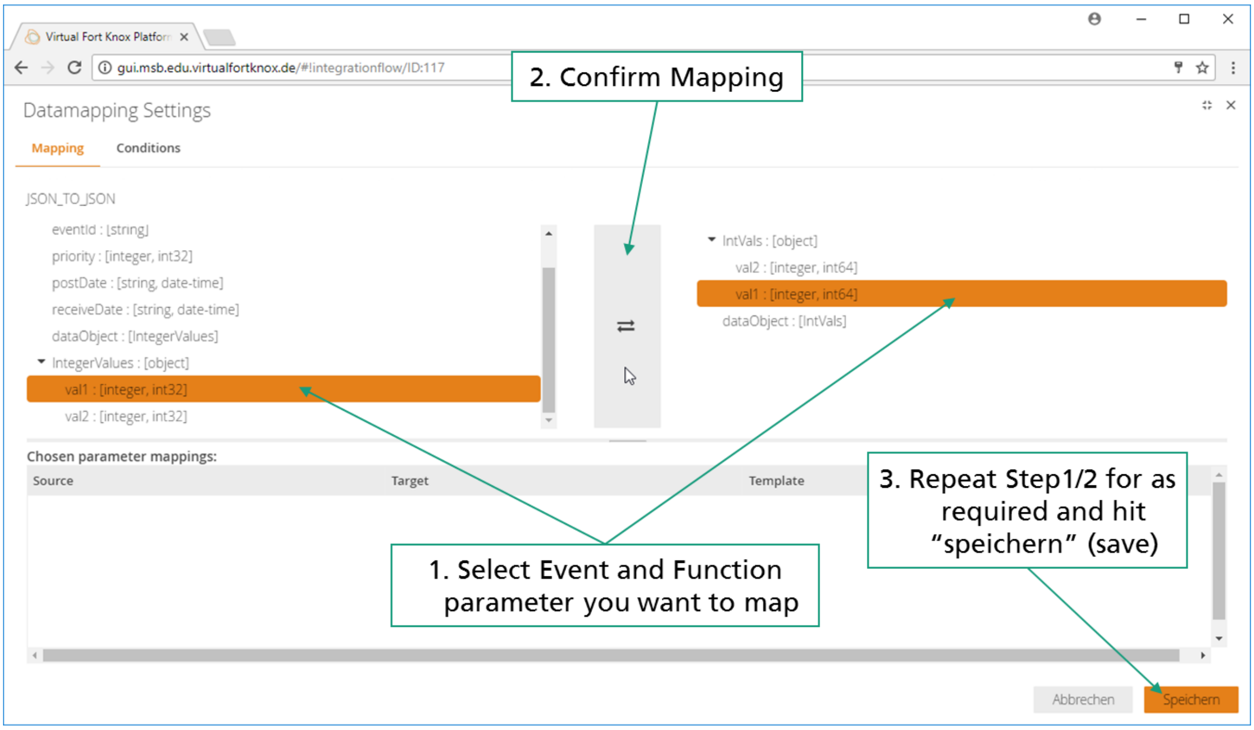

Figure 30: Detail View for Mapping Event Data to Input Parameters for a Function

The MSB is capable of simple type casts which require no further specification such as integer to float, integer to string. Most input and all output parameters are specific to the component developers. However, the parameters mentioned in Table 4 are always available for mapping on the side of the event. The mapping is conducted by selecting the input and output parameters which are supposed to be mapped, indicated by an orange background and clicking the button with the two arrows in the middle. Once the mapping is completed it will show up in the bottom half of the screen, were it can also be deleted by clicking the icon with the garbage bin. Once the user has mapped all function input parameters to the respective event output parameters, he can click the orange save button in the bottom right to save his changes. In general double mappings should be avoided, as they can lead to confusion. If a double mapping occurs, the latest mapping, indicated by being lower in the list, takes precedence. The mapping process has to be repeated for all links (all orange arrows in Figure 29), where mapping is required

Table 4: List of standard event properties

uuid	UUID of the component which sends the event.
eventId	ID given to the event by the developer
priority	Priority set by component developer for transfer by the MSB which might be relevant in case of high load. Possible values are: 0-low / 1(default)-medium / 2-high
postDate	Time when event was thrown by the component.
recieveDate	Time when the event was published to the MSB. The distinction is relevant when a component is set up in such a way that it can function autonomously without MSB connection (e.g. in remote regions without WIFI connection. Optionally events can be buffered in this case and published to the MSB once reconnected.

2.5.8 Setting Conditions for Data Transfer

In some cases, the information of an event should only be forwarded to a function when specific conditions are met, which were not foreseen by the developers who designed the components. In this case the *Conditions* tab in the mapping view can be used to set conditions. Conditions can only be set for the data associated with the current event. Other conditions, for example including information from the previous event, are not configurable. In some cases, smart design of the integration flow using the branching (section 2.5.9) and merging (section 2.5.10) can be used to achieve the desired results in combination with conditions. A new condition is set by clicking the relevant input parameter form the associated event and selecting the parameter on the left side and clicking the large button in the middle. This yields a view with a drop down menu like Figure 31 from which the desired comparator can be chosen. Once the user does so, he can set the compare value in a newly appeared field.

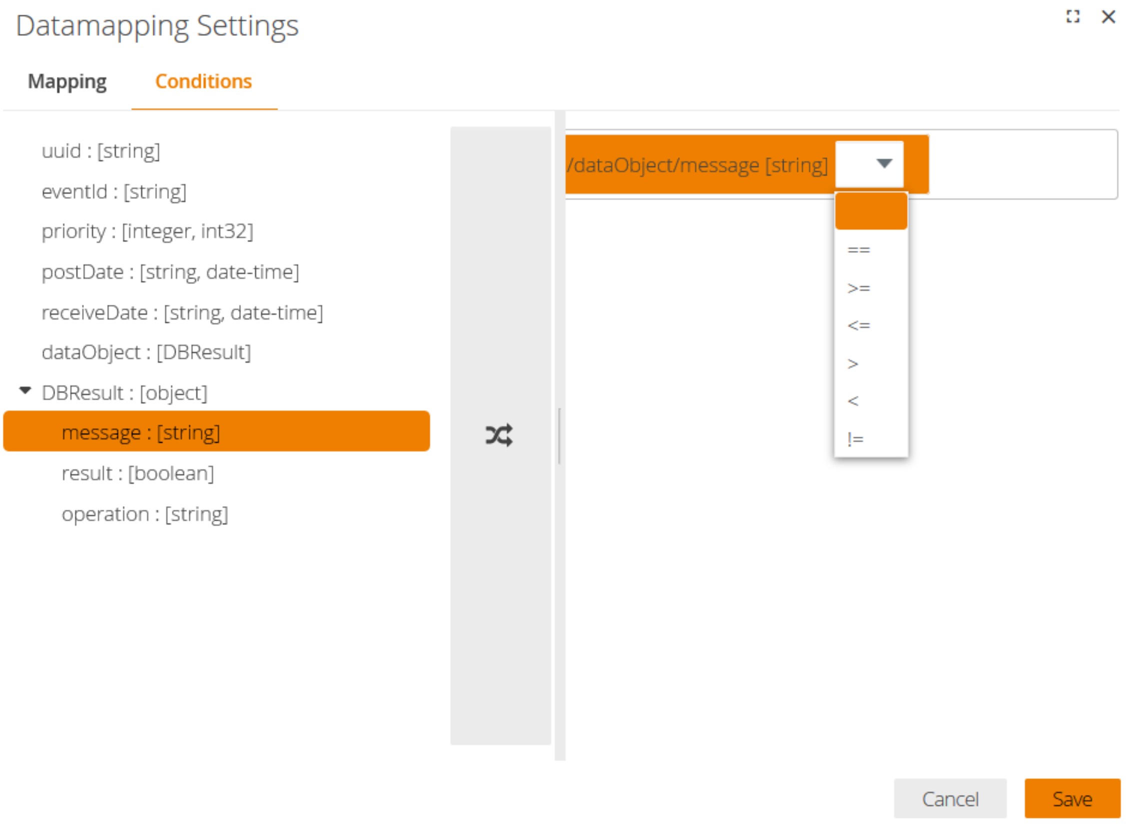

Figure 31: View to set Condition which incoming Events are forwarded to the next Component in the Flow.

2.5.9 Branching of Integration Flows

The length of integration flows, as in the number of event to function links, is largely unlimited. The MSB allows more complex designs as well, beside strictly linear integration flows. It is possible to map one event to several functions of one or more components, effectively creating branches in the flow, as shown in Figure 32. In case a subset of these functions is from the same component, the component needs to be dragged and dropped once for each individual mapping. If the branches do not merge again, the user should decide if two separate integration flows (with two separate names) would improve the overall overview.

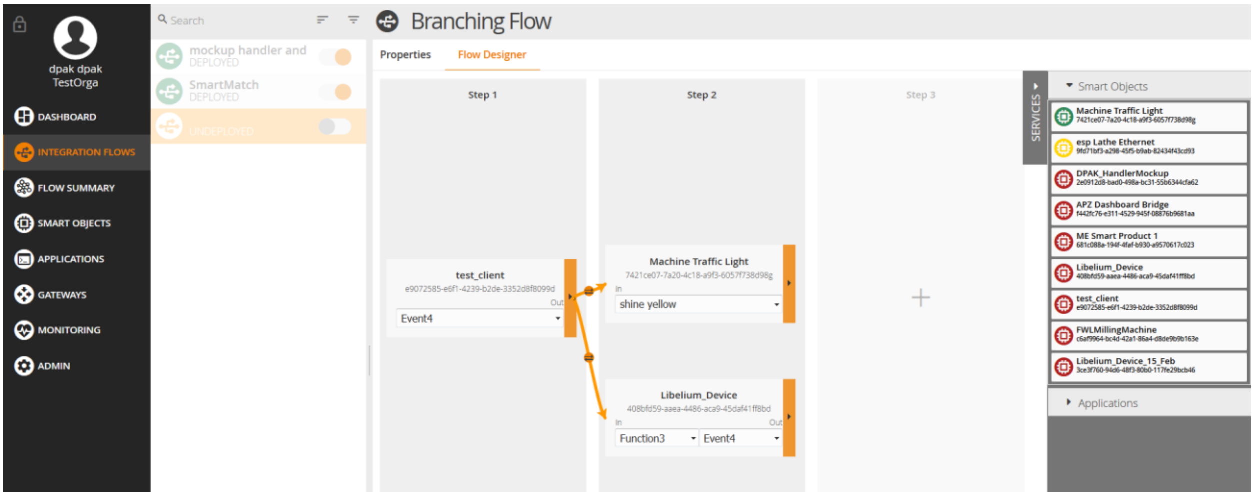

Figure 32: Branching Integration Flow where one Event is forwarded to Two Components

2.5.10 Merging of Branches in Integration Flows

The merging branches works in a similar fashion as linking and mapping (see Figure 33 for the final view after linking). Once the data of the first event is mapped to the input parameters and saved by clicking the button, the mapping of the data of the second input event can be conducted accordingly. The mappings of the other event will appear in the list of existing mappings (lower half of Figure 30) and vice versa.

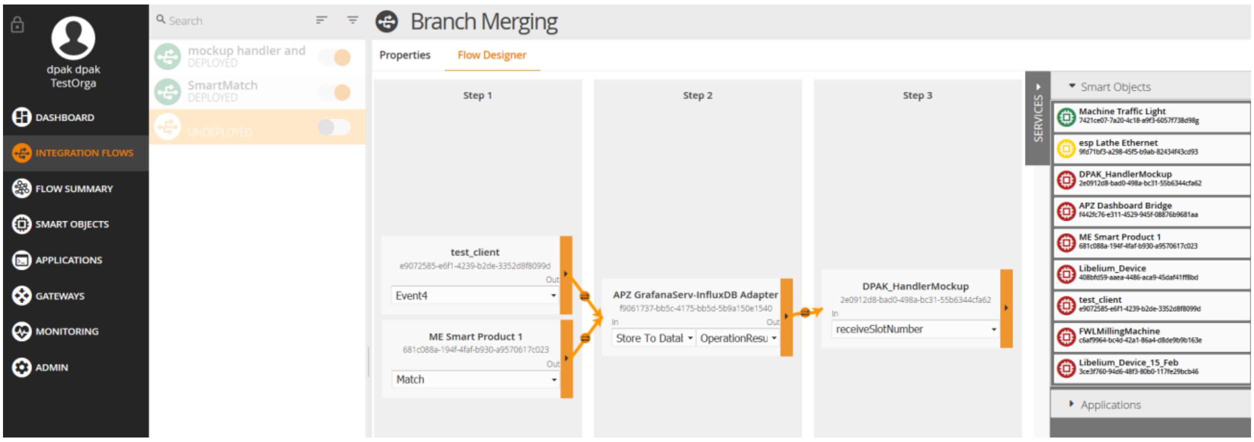

Figure 33: Merging of Branches in an Integration Flow

2.5.11 Wrap up and Saving the Integration Flow

The final steps before the integration flow can be activated, is saving the current setting and activating the flow. Before doing so, the user should be sure that the steps described before are completed. For better overview they are mentioned here once again:

- Name and description represent the purpose of the integration flow sufficiently.
- All components are in the main area of flow manager (at least one square for each component).
- For each component the correct function / event is selected (lower half of the graphical representation of the component).
- All required links are in place (orange arrows).
- All mappings are set as required for each link.

- All conditions (if required) are set (can only be checked in the respective detail views by clicking the orange nobs in the middle of the arrows and switching to the *Conditions* tab).

If this is the case, the user can click the orange save button on the bottom right of the browser window. As a result, the name of the integration flow in the list of flows on the left half of the screen will change from light gray to dark grey, as seen in Figure 34.

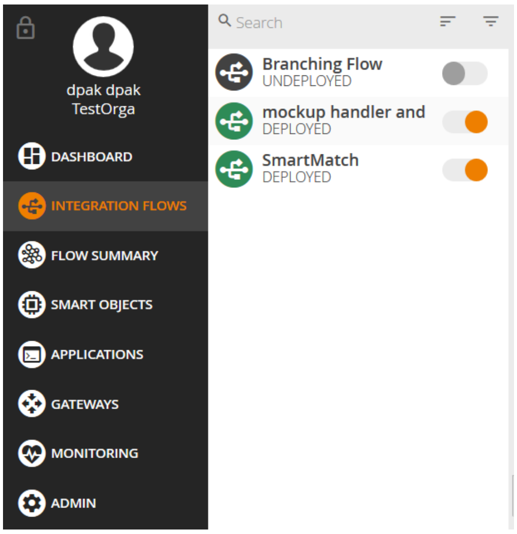

Figure 34: Integration Flow after Saving

By default, the integration flow is still deactivated. To activate it, the user has to hit the toggle button to the right of the name of the integration flow. The status of the flow can also be seen by a quick glance on the color of the icon to the left of the name. Activated flows are indicated by a green icon, while deactivated ones are indicated by a black icon. The activation of the integration flow in the backend of the MSB takes between 1 s – 10 s.