FIRST REPORT ON OPERATING MARKETPLACE

Deliverable D2.3

CIRCULATION
Choose an item

VERSION
1.0

DATE
03-03-2022

AUTHORS
Jeroen Broekhuijsen, Armir Bujari, Roman Dolhai, Damiano Fialconi, Gladys Gallot, Michał Kulczewski, Adam Olszewski, Anna Sumereder, Wilfrid Utz, Robert Woitsch

LEAD PARTNER
PSNC

CONTRIBUTING PARTNERS
BOC, ClOUDBROKER, TNO, UNIBO

QUALITY CONTROLLERS
Jeroen Broekhuijsen, TNO
Gladys Gallot, IR

HORIZON 2020

Document History

Version[1]	Issue Date	Stage	Content and Changes
0.1	22-11-2021	1	Initial TOC
0.2	17-12-2021	1	TOC agreed
0.3	23-12-2021	1	First draft, including Section 3 and shopping cart
0.4	05-01-2022	1	Draft on Section 4
0.5	11-01-2022	1	Draft on Section 5, updated Section 3
0.6	13-01-2022	1	Draft on Section 2
0.7	14-01-2022	1	Executive summary
0.8	20-01-2022	1	Updates to Section 4
0.9	21-01-2022	1	Addressing internal review comments
0.10	22-01-2202	1	General update
0.11	28-01-2022	1	Addressing QC comments
0.12	22-02-2022	1	Addressing 2nd QC comments
1.0	03-03-2022	1	Final version

[1] Integers correspond to submitted versions

EXECUTIVE SUMMARY

This first report on operating Marketplace provides insight on the status of the Marketplace, its development and performance of its operation. The current version is a step forward comparing to the landing pages, described in previous D2.1 and D2.2 deliverable. **The number of offerings were increased to 36**, by following the already shaped onboarding process and processing offerings which are external to the project. A lesson has been learned that **the governance process for onboarding needs to be swifter**, and corrective actions are proposed.

Improvements under development are focused on three areas: allowing direct usage of the offerings listed for increased uptake, increased search capabilities for better findability and measuring the effectiveness using an evaluation framework. These are based on input from the review committee, partners experience from previous projects, the assumptions made at project-level, analysis conducted in WP1 and initial feedback from DIH. Further feedback from DIHs provided by WP6, offerings owners and WP1 will be used for future improvements, and discussed in next reports.

User Interface and User eXperience (UI/UIX) have been improved to increase the uptake of marketplace items and to make DIHs and marketplace owners uptake our marketplace solution. In particular, they allow DIHs to use and reshape the marketplace according to their needs. This targets the main WP2 objective: *Create Marketplace Portal for Digital Twin offerings*. Also the users are missing possibility to actually use the offerings listed in the Marketplace. To this end, the middleware is being extended with an introduction of shopping cart concept, to eventually allow the deployable services to be directly used. Supportive tools are provided, such as identity managements, contributing to *Establish and adapt marketplace middleware, shop, and identity management*, and we aim this solution to be ready by the next report.

Advanced search capabilities are under development. These will allow users to benefit from the re-usable marketplace item model. The ADOxx platform is provided for interactions with models, contributing to the objective: *Provide a method for creating model-based marketplaces*. For companies that seek applications for digital twinning that are the most useful given their goals, an assessment and compass tool are provided, which in the future will be used to filter the marketplace by selecting components relevant to the user. And to increase the chance of finding the right solution, we target the objective: " *Establish cooperation and content exchange with existing relevant marketplaces*" to populate the Marketplace with 3rd party offerings by developing and using the crawler tool. Still, there are pending legal issues to be solved.

From operational point of view, an overview of the usability, community interest and potential issues are provided, followed by the evaluation framework. The monitoring solution has been provided to analyse the community interest in the Marketplace. Already we have identified that such tools should target different users such as administrators, offering owners and end users. The future developments will be liaised with their expectations and the evaluation framework will be developed in close collaboration with WP1 and WP6.

TABLE OF CONTENTS

1 INTRODUCTION

This document presents the first report on operating the Marketplace, describing the current version of the Marketplace and providing insight on the status of tool development for the Marketplace. The content here described naturally evolves from the previous deliverables, in particular D2.1 in which initial operating Marketplace was introduced, and D2.2 where an extensive discussion on Marketplace model was conducted. In contrast to previously delivered documents, a clear separation between the model, middleware and infrastructure is now avoided because they already form a consistent solution. The motivation for this report is to present how we are shifting from landing pages to Marketplace portal with developments concerning the model, the UI/UX, middleware, infrastructure capabilities and supportive tools. Another objective is to present the Marketplace from operational point of view, to draw conclusions on challenges and potential enhancements with respect to the users.

1.1 BRIEF MARKETPLACE OVERVIEW

The Change2Twin (C2T) Marketplace is a portal that allows companies to search and use relevant services and products in their digital twin solution. It is a model-based web application created using the OLIVE Framework for frontend and backend components and the ADOxx platform for interactions with models. On a high-level view (Figure 1), the user interface of the marketplace is an instantiation of web Widgets. The Widgets are provided by the micro-frontend framework and configured to communicate with microservices in the backend created through the instantiation of existing components named Connectors. The Connectors are provided by the microservice framework, implementing the specific business logic. The services are responsible to dynamically retrieve the available marketplace items and their details from a data-lake provided in form of GitLab repositories and to integrate model data retrieved by the ADOxx meta-modelling platform. Underneath there is infrastructure, on which marketplace is operated, and hosting some of the deployable offerings. The middleware is responsible for identity management, shopping card and act as a mediator between the marketplace portal, model and infrastructure.

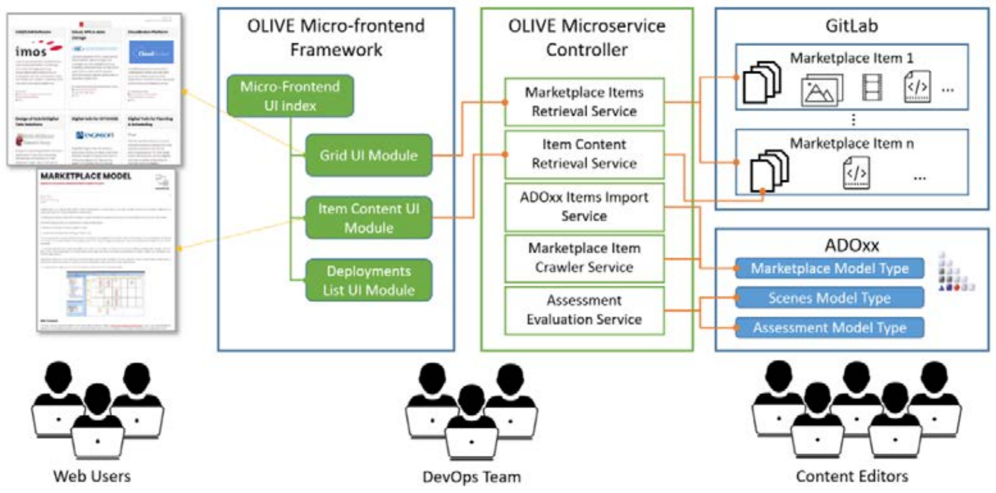

OLIVE Micro-frontend Framework
Micro-Frontend UI index
Grid UI Module
Item Content UI Module
Deployments List UI Module
OLIVE Microservice Controller
Marketplace Items Retrieval Service
Item Content Retrieval Service
ADOxx Items Import Service
Marketplace Item Crawler Service
Assessment Evaluation Service
GitLab
Marketplace Item 1
Marketplace Item n
ADOxx
Marketplace Model Type
Scenes Model Type
Assessment Model Type
MARKETPLACE MODEL
Web Users
DevOps Team
Content Editors

FIGURE 1 MARKETPLACE HIGH LEVEL INSTANCES ARCHITECTURE

1.2 ONGOING DEVELOPMENTS

The Marketplace holds now 36 offerings, including those external to the project, and onboarding process is used to increase this number. More will come with the crawler tool to use 3rd party marketplace offerings, but (legal) agreements will need to be in place with other sites or marketplaces before these can be activated. The Marketplace is extended with monitoring to analyse the community interest in it. The portal is further developed to improve the UI/UX. To this end, additional components are provided and under developments, such as shopping cart, advanced search capabilities, assessment and compass tools. The shopping cart will allow some of the offerings to be directly used in the Marketplace, and we aim to deliver such functionality with the next report. Similarly, the monitoring tools will provide information valuable to different kind of users. Corrective actions have been proposed to onboarding process to make it less involving, and fully automated solution is to be provided with next report. The remaining developments need to be liaised with marketplace users, thus an evaluation framework is proposed to survey different users for their feedback.

1.3 READING GUIDE

This remaining document is organized as follows: Section 2 presents the Marketplace from the operation point of view. Section 3 gives details on the development of tools. The challenges, issues and future work are discussed in Section 4. Technical details of the portal are presented in Appendix A. List of identified requirements and expectations is given in Appendix B.

2 OPERATING THE MARKETPLACE

2.1 CURRENT STATUS

The Change2Twin Marketplace is hosted and operated by PSNC, and it is available under http://marketplace.change2twin.eu. Since launching the marketplace landing pages in M12, the number of offered services increased, the onboarding process is shaped to ease the governance process, assessment tools are under development, information exchange between different marketplaces is possible, and shopping cart is slowly under investigation to introduce deployable services to the Marketplace. Appendix B lists discovered expectations and requirements.

Currently, the Marketplace is holding 36 offerings: 27 are the internal offerings from project partners, while remaining 9 are external ones already onboarded. More offerings are under the revision process, waiting for additional information to be provided by the owners. In terms of the TRL, 13 solutions are innovation products (level below 8), and the remaining ones are fully qualified products (level 8-9). Table 1 is summarizing available offerings.

TABLE 1 SUMMARY OF THE OFFERINGS

Name	Owner	TRL	Type
CAD/CAM Software	Imos AG	9	Solution offering
Cloud, HPC & Data Storage	PSNC	9	Iaas, Paas, SaaS
Cloudbroker Platform	CloudBroker GmbH	8	Software as a service
Design of Hybrid Digital Twin Solutions	Mobile Middleware Research Group	6	Software - docker
Digital Twin for Offshore	EnginSoft SpA	9	Solution offering
Digital Twin for Planning & Scheduling	Dijitalis Ltd.	9	Bundle offering for individual deployment
Digital Twin of Process	BOC Group	6	Software - package
Digital Twin of Production Process	BOC Group	9	Software as a service
Digitalization Assessment	TNO	8	Consultancy / Online tool / Software tool
Documentation tool	Author-e BV.	7/9	Software as a service, on-premise
Dominus EYE	Dominus Tech Ltd.	7	Software as a service
EDMsdk	Jotne	9	Solution offering
EDMtruePLM	Jotne	9	Solution offering
GoTools	SINTEF Digital	5	Software library
Interface Development	CG-Ingenieros	8	Software and consultancy
IOT Platform	MONOM S.L. – ALAVA GROUP	9	Software as a service
Low-code machine-control	Cordis Products B.V.	9	Software package
Manufacturing Monitoring	ICB	9	Software offering
Marketplace Download Package	CloudBroker GmbH	5	Software package
Marketplace landing page service	PSNC	6	Software as a service
Marketplace model	BOC Group	4	Software package
Nerve	TTTech Industrial	9	Solution offering
OMiLAB Innovation Corner	BOC Group	5	Service
Online Community	FundingBox	9	Service
Open Call Management	FundingBox	9	Software as a service
Open Innovation	FundingBox	9	Service
Prespective Digital Twin Software	Unit040 Ontwerp B.V.	9	Software
Quality-aware Clud-to-thing Management & Control Platform in Support of Digital Twins	Mobile Middleware Research Group	6	Software docker
Seven Step strategy for Digital Twins	TNO	7	Methodology & workshops
Simulations models	Space Structures GmbH	9	Solution offering: consultancy
SISL	SINTEF Digital	8	Software library
Smart Connected Factory	TNO	7	Software tool / Software as a service
Service Industry 4.0	CT-Ingenieros	8	Software and consultancy
TRI Soft Digital Twin	TRI Soft	9	Solution offering
Your limitless digital twin	Ingenieria y Diseno Estructural Avanzado SL	8/9	Software offering
Your robotics 3D Digital Twin	Eleven Dynamics GmbH	9	Solution offering

Speaking of the sustainability, the Marketplace is prepared to be offered in two different versions. A more mature solution is available as a software-as-a-service, where marketplace can be hosted on the Change2Twin infrastructure and operated by its owner in similar way the Change2Twin Marketplace is operated. A less mature solution is offered as a software package for users, DIHs in particular, willing to run the marketplace on their own and on their / 3rd party infrastructure.

In order to provide some basic statistics to Marketplace operator, and to provide feedback to offerings owners, the Matomo analytics platform[2] is used starting from December 1st 2021. By the end of February, there were 202 visits, with an average duration of 2 min 59s and 4 actions per each visit (page view, download, outlink, etc.), see Figure 2.

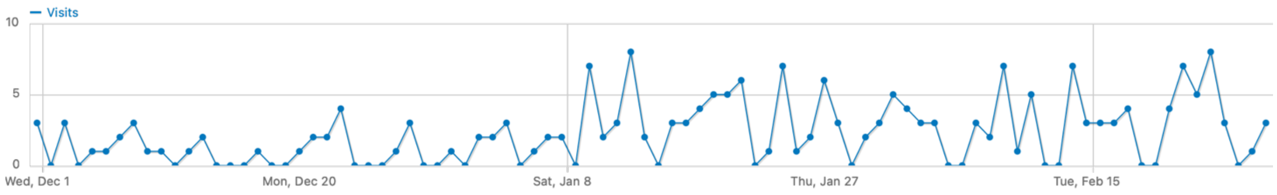


FIGURE 2 MARKETPLACE VISITORS IN PERIOD DECEMBER 2021 – FEBRUARY 2022

Although the number is not yet impressing, it is worth mentioning that the Marketplace is recognizable all over the world – during first two months visitors came from 30 distinct countries, though Europe is vastly leading on this one, see Figure 3.

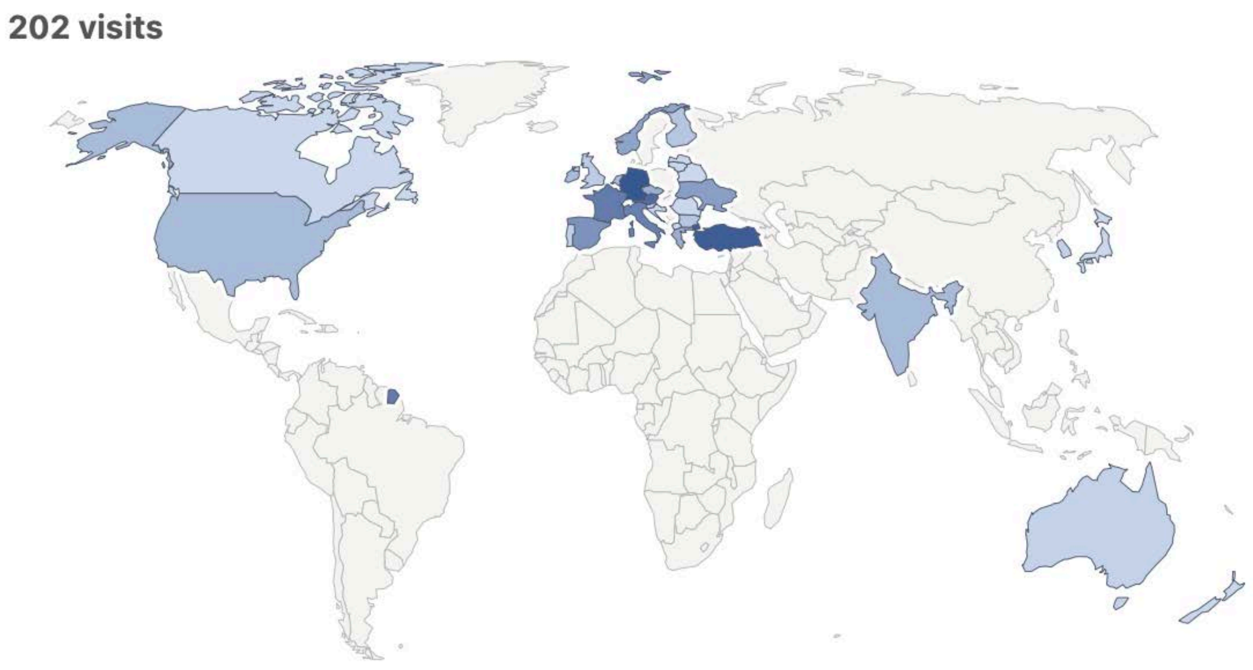


FIGURE 3 MARKETPLACE VISITS PER COUNTRY – DECEMBER 2021 TO FEBRUARY 2022

The most visited offerings are listed in Table 2. A significant increase in visitors is foreseen with more advanced tools made available, which are discussed in next sections.

TABLE 2 MOST VISITED OFFERINGS IN PERIOD DECEMBER 2021 – JANUARY 2022

Name	Owner	TRL	Type
Digitalization Assessment	TNO	8	Consultancy / Online tool / Software tool
Digital Twin for Planning & Scheduling	Dijitalis Ltd.	9	Bundle offering for individual deployment

[2] https://matomo.org

Digital Twin of Process	BOC Group	6	Software - package
Design of hybrid Digital Twin solitoons	Mobile Middleware Research Group	6	Software docker
Manufacturing Monitoring	ICB	9	Software offering
CAD/CAM Software	Imos AG	9	Solution offering
Cloudbroker platform	CloudBroker GmbH	8	Software as a service
Digital Twin of Production Process	BOC Group	9	Software as a service
IOT Platform	MONOM S.L. – ALAVA GROUP	9	Software as a service
Your limitless Digital Twin	IDEA	8/9	Software offering
Smart Conntected Factory	TNO	7	Software tool / SaaS
Cloud, HPC & data storage	PSNC	9	Iaas, PaaS, SaaS
Digital Twin for offshore	EnginSoft SpA	9	Solution offering

Operating the Marketplace for several months now identified several challenges, some of them are discussed in details in following Sections:

- Governance process issues, Section 2.2;
- Infrastructure efficiency for different types of offerings, Section 2.3;
- Monitoring of resource usage, Section 3.3
- User management, Section 3.1
- Advanced search capabilities, Section 3.6
- Deployable services and shopping cart, Section 3.4

2.2 ONBOARDING

The onboarding is a process defined during the scope of the project to facilitate appearance of the new offerings on the Marketplace. At a glance, inviting a new provider is straightforward, but taking into account all the different contextual information of an item is cumbersome. An item needs to describe the relation to Digital Twinning, be recognisable for end-users, specify effort required, which type of process/manufacturing is involved, how an item can be acquired, etc. In order to facilitate the first stage – accepting a new offer into the Marketplace, we proposed an onboarding process presented in Figure 4.

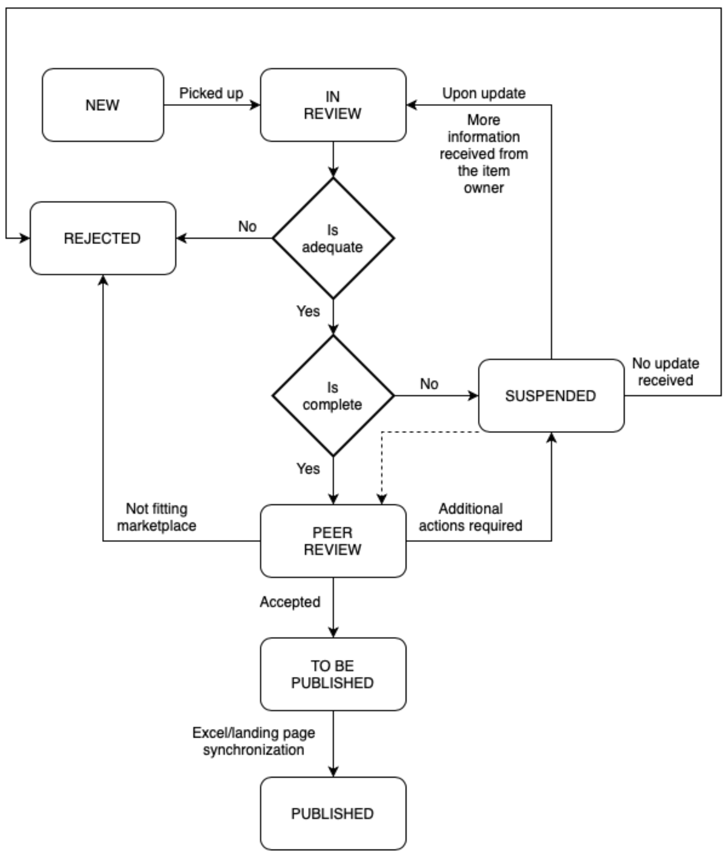


FIGURE 4 ONBOARDING PROCESS

The first stage is to provide necessary information about the offering by sending email to the onboarding team – marketplace@change2twin.eu. Once there is a new inquiry, the team responsible for the onboarding process provides details of the offer into the ticketing system and review of the content takes place. The workflow in ticketing system is presented in Figure 5. In case of any doubts, the onboarding team may contact the offering owner for details (*waiting* status), reject the offer (because no required information was provided or the information does not fit the Marketplace profile – *rejected* status) or pass it for technical review. Now, the technical review is required in case deployment matters or technical requirements need to be discussed. At this stage the offer again can be rejected, put on hold or be given a green light to become part of the Marketplace. After the offer is accepted, the Marketplace is updated accordingly, finalizing the onboarding process.

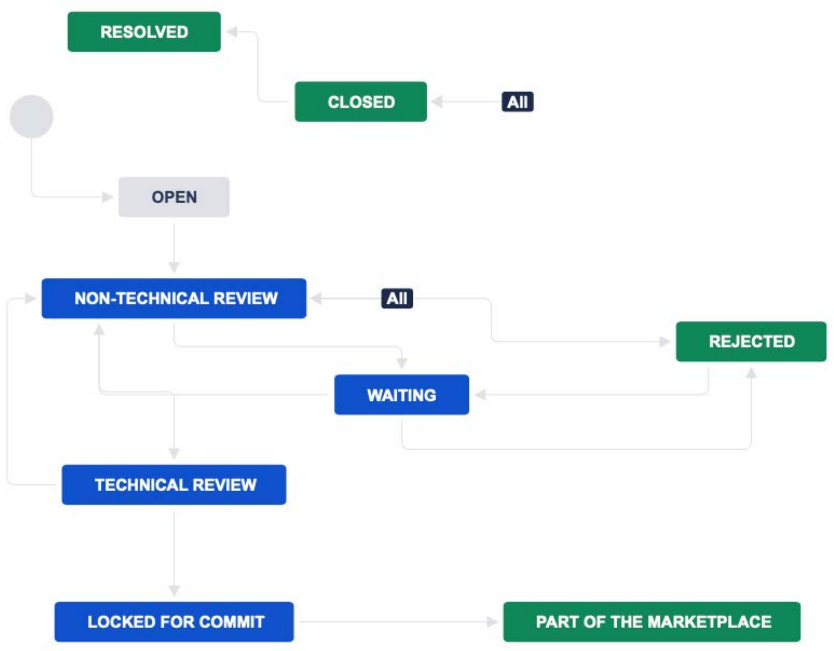


FIGURE 5 ONBOARDING PROCESS WORKFLOW

During the first couple of months of using this process we have learned that there is room for improvement by making things less human-busy and more automatic. Figure 6 presents our vision for this improvement. Instead of sending a document with required information describing an offering, there will be an online webform that will include initial content check (1). Once submitted, the offering will be automatically provided into the ticketing system (2). The onboarding team will then have just to revise the content, just like it is done currently (3). After accepting an offer, it will automatically become part of the marketplace by the CI/CD (continuous integrations / continuous development) approach (4).

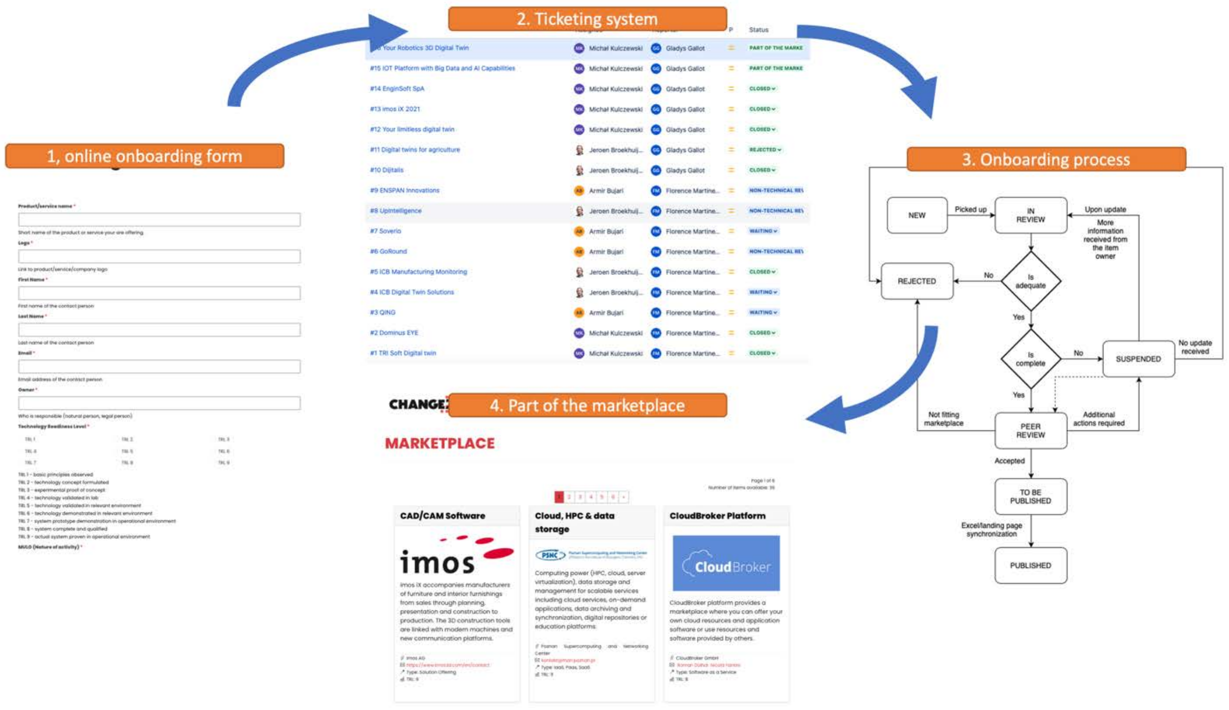


FIGURE 6 AUTOMATION OF THE ONBOARDING PROCESS

The onboarding process is based on the Atlassian Jira cloud solution, which is available for free, but other ticketing system can be used. The proposed workflow and transitions can also be changed, depending on the requirements of the Marketplace owner. In the next report it is planned to propose extension to this process, describing how an offering can be updated, or deleted if required.

2.3 INFRASTRUCTURE

The infrastructure is used for the purposes of running the Change2Twin Marketplace, including hosting some of the internal and WP3 pilots offering, managed by the middleware. Currently, two different types of services are exploited by the infrastructure – a 24/7 and on-demand service. The 24/7 type is represented by Jotne and Robopac cases. The on-demand service is provided to SPS, who allows users to run Optistruct simulations per on-demand basis. Each of these services is using provided infrastructure, which is then managed by the provided middleware.

Offerings being deployed require: 1) high throughput, 2) large and efficient storage, 3) computational efficiency. W**e have identified potential issues and bottlenecks from the perspective of infrastructure owner**, which impacts the end user experience. The 1 and 2 are guaranteed with: the hardware availability (storage), the proper high-bandwidth network components, and the overall orchestration. For computational demanding services, e.g. computational simulations, we observed issues with application efficiency comparing to local machines or those available in HPC environment. This is due to the fact that infrastructure providers aim to minimize the number of CPUs being utilized and share CPUs across different services. For most scenario's dynamically using and switching between CPUs is acceptable. For the computational-demanding applications this isn't the case, a non-shared CPU and memory resources needs to be provided. This can be done by providing a separate region to the cloud where only computationally insensitive applications may run. Another, yet complimentary approach, is to provide a higher priority to such on-demand services, so that they can receive CPUs exclusively, while the previously running applications can be migrated to the co-shared CPUs. These solutions are currently under revision, so that in the future no efficiency issue shall be observed, regardless of the offering type and requirements.

2.4 EVALUATION FRAMEWORK

The aim of the evaluation of operating the marketplace is to validate developed components, tools and features against different users needs, expectations and requirements to foster future developments. This evaluation will more closely align the inputs from different work packages.

WP1 focuses on enabling technologies to make them suitable for DIHs and SMEs, and to become part of the Marketplace eventually. A living document has been provided as an outcome of tasks T1.3 and T1.4 detailing possible extensions and enhancements to be introduced to the Marketplace along with some requirements. These requirements are now under analysis to evaluate their feasibility and importance.

WP6 provides direct connection to DIHs, so it is possible to survey them to receive feedback on Marketplace look and feel. The Marketplace, promoted as the main place for DIHs to find digital twin solutions for the SMEs, seems currently underused during the Assessment process within the Change2Twin project. A survey conducted in November 2021 following the end of the first round of Assessment process, revealed that 9 out 14 DIHs involved used the Marketplace to find solutions/technologies for their digital twin recipes. They wish they could have sufficient information and training to use this marketplace appropriately. According to them, the number of offers available has to increase, and following should be added:

- Solution to design complex embedded system and circuits involving optical sources and sensors;
- Solutions for academic environment;
- Training services and tools to facilitate the introduction and sustainability of Digital Twin;
- Solutions for decision-making and AI;
- Data integration tools;
- Search engine;
- Success stories;
- API/platform integration;
- Pricing schemes.

In order to strengthen the knowledge of these actors a specific training session on the Marketplace will be proposed on the occasion of the certification of the DIHs prior the launch of the next Assessment Open Call.

The onboarding team provided feedback regarding the way of offering presentation and requirements. Once there is user management and shopping cart introduced to the Marketplace, we will need to survey users for further feedback.

For the next report we aim at discussing outcomes of surveying different users and how it reflects on developments around the Marketplace.

3 IMPROVING THE MARKETPLACE

3.1 USER MANAGEMENT TOOL

Status: operational, for administrative purposes solely at the moment

The user management is handled by the Keycloak[3] – an open-source identity and access management for applications and services. The Marketplace can be configured to use this kind of user management system, which can be installed locally or remotely to the Marketplace site. Currently, the identity management system is configured to be used for administrative purposes of the Marketplace and by the middleware. It can already handle users of the Marketplace, but this feature will be enabled once the shopping cart is introduced to the Marketplace (see Section 3.4).

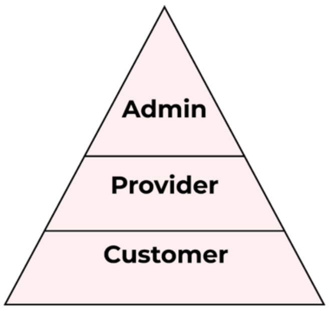


FIGURE 7 USER ROLES HIERARCHY

Three different user roles will be available on the marketplace: Admin, Provider and Customer, as presented in Figure 7. Differentiation between users can be represented through “key access points” – a list of access permissions which defines whether user is able to work with only own content or manage entire marketplace. The descriptions are following:

<u>Key access points</u>

- Items management (being able to edit or delete marketplace items)
- Item history (being able to track changes of marketplace items, e. g. change of description or its type)
- Operations management (being able to track purchases and shopping carts)
- User management (being able to edit user information and grant users with access)

<u>Admin has full access, which means he is able to:</u>

[3] https://www.keycloak.org

- Edit or delete **all** marketplace items;
- Track changes of **all** marketplace items;
- Able to track **all** purchases and see **all** shopping carts;
- Can edit **all** user's information and grant **all** users with access.

Providers have limited access. It means they are able to:

- Edit or delete **only** marketplace items he published;
- Track changes **only** of **his published** marketplace items;
- Manage and edit **only** his own user profile;
- Track customer's purchases **only** of item he published.

Customers have the most limited access, and are able to:

- Manage and edit **only** his own user profile;
- Track only **own** purchases.

3.2 ONBOARDING TOOL

Status: Partially operational, with manual governance

It is planned to provide onboarding tool as described in Section 2.2. This tool will be based on freely available ticketing system in the cloud, and will come with possibility to automatically integrate with onboarding online form and CI/CD of the Marketplace. In this way the onboarding process will be facilitated to the possible extent, yet the process may be altered according to marketplace owner requirements.

3.3 MONITORING TOOL

Status: operational, at middleware level only

The monitoring is currently handled by the middleware. For the moment being, following metrics can be tracked:

- Amount of time an instance is running;
- Costs: based on cloud-provider prices, fee type and VAT (optional);
- Logs based on three columns (Date/Time, Severity of message, Message text). It tracks communication between broker-side and cloud provider as well as checks if the instance and the connection to it work correctly.

Additional endpoints can be configured to retrieve additional monitoring information. An agent-based approach could also be introduced. On each monitored virtual machine, two agents can be installed – one for the operating system, the other one for docker. Collected metrics will be scraped by the Prometheus[4] open-source monitoring solution, and then exposed to the middleware for further processing. This will guarantee that the low-level monitoring can be introduced to any kind of the infrastructure.

The information need on monitoring varies between users. Marketplace owners are interested in general statistics, including how many visitors there were, how many users registered or which offerings are most in demand. A service provider is interested in detailed statistics of their items being offered to improve their service. Reliability, returning customers, how the offer is positioned against items alike, how a user reaches the offer are just few examples. End users may be interested in some analysis of historical purchases, items used, results obtained. **We aim at providing monitoring data to each of the aforementioned user type.**

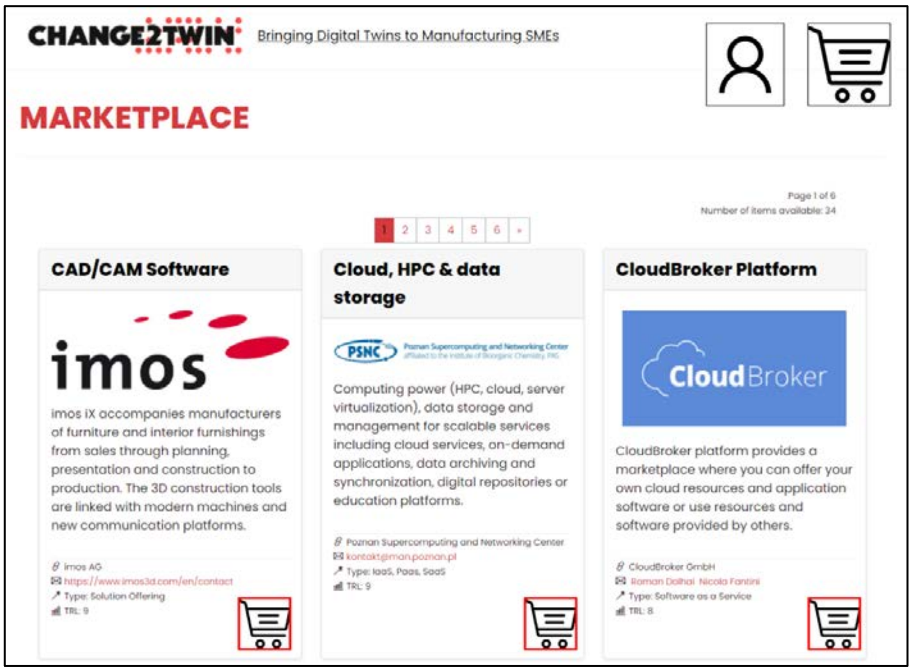
3.4 SHOPPING CART

Status: concept phase, implementation starts

The next version of Change2Twin Marketplace aims at allowing end-users to use the offerings, in particular to purchase items. To this end a shopping cart functionality is required, which concept is here described.

The idea is to implement shopping cart for Marketplace users and adapt it towards usability patterns to ensure better user experience. The key concept is therefore described through a single user story: "As a marketplace user (customer), I want to add an item in my cart, so that I can purchase it". Considering Ui/UX, users will need to be able to add offerings from catalogue (*add to basket* button on every offering possible to be bought), verify shopping cart (basket button) and update user profile (*user* button). An exemplary mock-up is demonstrated in Figure 8.

[4] https://prometheus.io


CHANGE2TWIN Bringing Digital Twins to Manufacturing SMEs

MARKETPLACE

Page 1 of 6
Number of items available: 24

1 2 3 4 5 6 »

CAD/CAM Software

imos iX accompanies manufacturers of furniture and interior furnishings from sales through planning, presentation and construction to production. The 3D construction tools are linked with modern machines and new communication platforms.

imos AG
https://www.imos3d.com/en/contact
Type: Solution Offering
TRL: 9

Cloud, HPC & data storage

Computing power (HPC, cloud, server virtualization), data storage and management for scalable services including cloud services, on-demand applications, data archiving and synchronization, digital repositories or education platforms.

Poznan Supercomputing and Networking Center
kontakt@man.poznan.pl
Type: IaaS, Paas, SaaS
TRL: 9

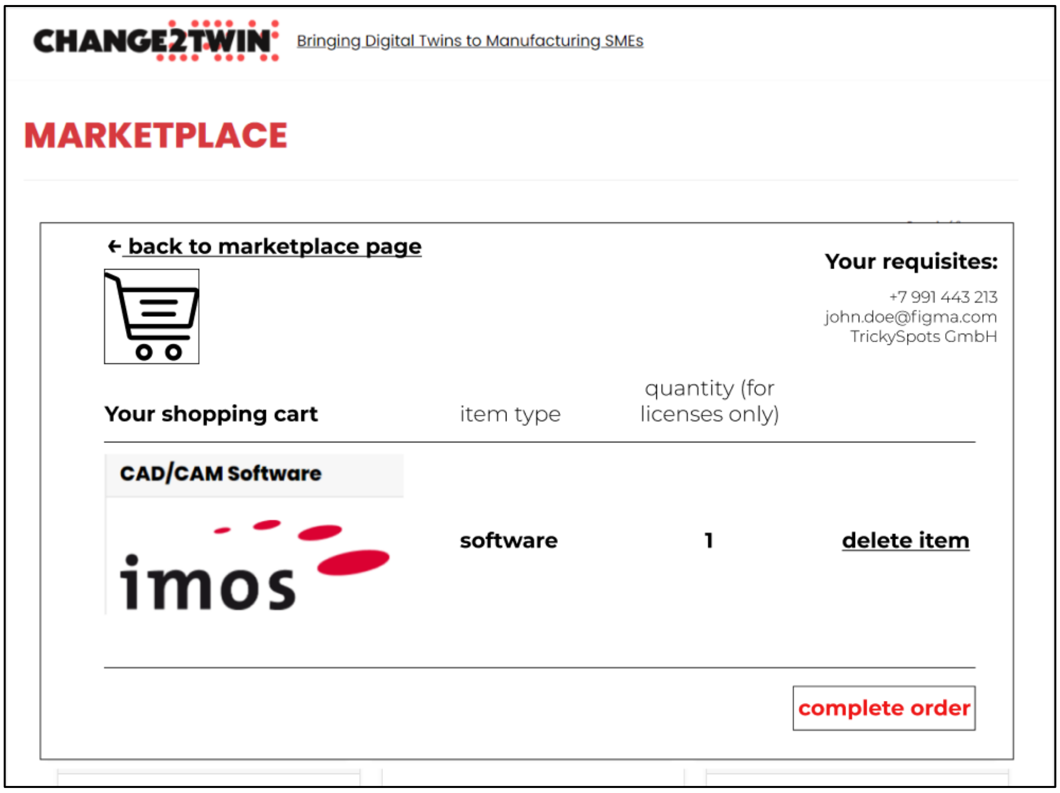
CloudBroker Platform

CloudBroker platform provides a marketplace where you can offer your own cloud resources and application software or use resources and software provided by others.

CloudBroker GmbH
Roman Dolhai Nicola Fantini
Type: Software as a Service
TRL: 8

FIGURE 8 MARKETPLACE MAIN PAGE WITH SHOPPING CART MOCK-UP

After offerings have been added to the basket, user is presented with a page for a final revision before completing the order. An exemplary mock-up is presented in Figure 9.


CHANGE2TWIN Bringing Digital Twins to Manufacturing SMEs

MARKETPLACE

← back to marketplace page

Your requisites:
+7 991 443 213
john.doe@figma.com
TrickySpots GmbH

Your shopping cart	item type	quantity (for licenses only)	
CAD/CAM Software	**software**	**1**	**delete item**

complete order

FIGURE 9 COMPLETING THE ORDER MOCK-UP

Based on the MasterCard & UsabilityLab research[5] we propose the following usability patterns to be used:

1. THE STEP OF ADDING AN ITEM TO CART SHOULD NOT CONTAIN UNNECESSARY STEPS AND ACTIONS.

Additional pages that appear when adding to cart makes you feel like you're wasting time and making unnecessary steps.

The page for adding an item to cart should have an opportunity to proceed with order placement without looking at the items in the cart.

2. THE INFORMATION THAT THE PRODUCT HAS BEEN ADDED TO CART SHOULD BE OBVIOUS TO THE USER.

After clicking on the "Add to cart" button the user should be provided with clear feedback that the item has been added to cart. If a store is more likely to buy only one item of the same name, then after clicking on the button a second time should take you to the cart page.

3. THE CART PAGE SHOULD HAVE DETAILED INFORMATION ABOUT THE PRODUCTS.

In our case it’s offering type, website, description of item, contacts and TRL level.

4. THE CART PAGE SHOULD HAVE INFORMATION ABOUT TERMS OF PURCHASE.

Information about the conditions of purchase can be presented by a link and lead to a page with detailed information about the conditions of purchase of marketplace items.

5. ITEMS MUST BE STORED IN THE CART UNTIL THE CHECKOUT IS COMPLETE.

If a user interrupts a purchase, for example because of a technical problem, he should be able to continue the order from the same place. Often the cart is cleared and the purchase process has to be started all over again. In such situations there is a high risk that the customer will abandon the purchase.

6. THE ABILITY TO REMOVE ITEMS FROM THE CART SHOULD BE PRESENT BY DEFAULT ON THE CART PAGE.

Many users use the shopping cart as a way to put off some items for later selection. And once in the shopping cart, they decide which items from the list they will purchase. That is why in the cart it is important to designate the possibility removal of unclaimed items.

[5] https://newsroom.mastercard.com/ru/files/2015/06/MasterCard-USABILITYLAB-отчет-Процесс-оплаты-картой-в-интернете1.pdf

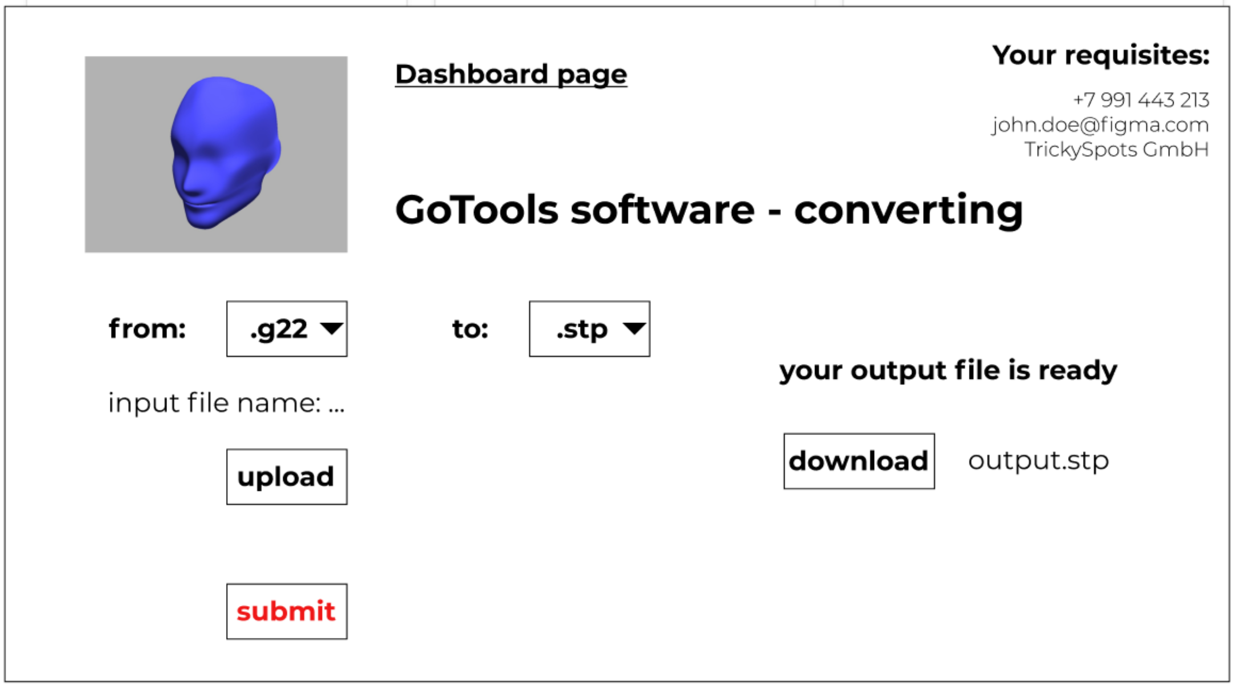
Use of the offering in the marketplace

One of planned implementation is providing GoTools software as one of the first usable offering on marketplace.

GoTools is the name of a collection of C++ libraries related to geometry. The libraries are organized as a core module with additional modules added on top. The core module contains generic tools and spline functionality. The additional modules contain functionality for intersections, approximative implicitization, parametrization, topology, and more.

GoTools software concerns CAD-system related objectives so the converting process was chosen as a basic computation task. Our idea is to let user uploads the file to marketplace and receive result of converting without additional efforts, so that all computation is being done in the cloud. Initially converter will work with .g22 and .stp files and amount of file types can scale up in future.

User will have an interface where he specifies filet types to be converted and upload the one. Once computation is done, it would be possible to download the output. The preliminary mockup is presented in Figure 10.


Dashboard page

Your requisites:
+7 991 443 213
john.doe@figma.com
TrickySpots GmbH

GoTools software - converting

from: .g22 to: .stp

input file name: ...

your output file is ready

download output.stp

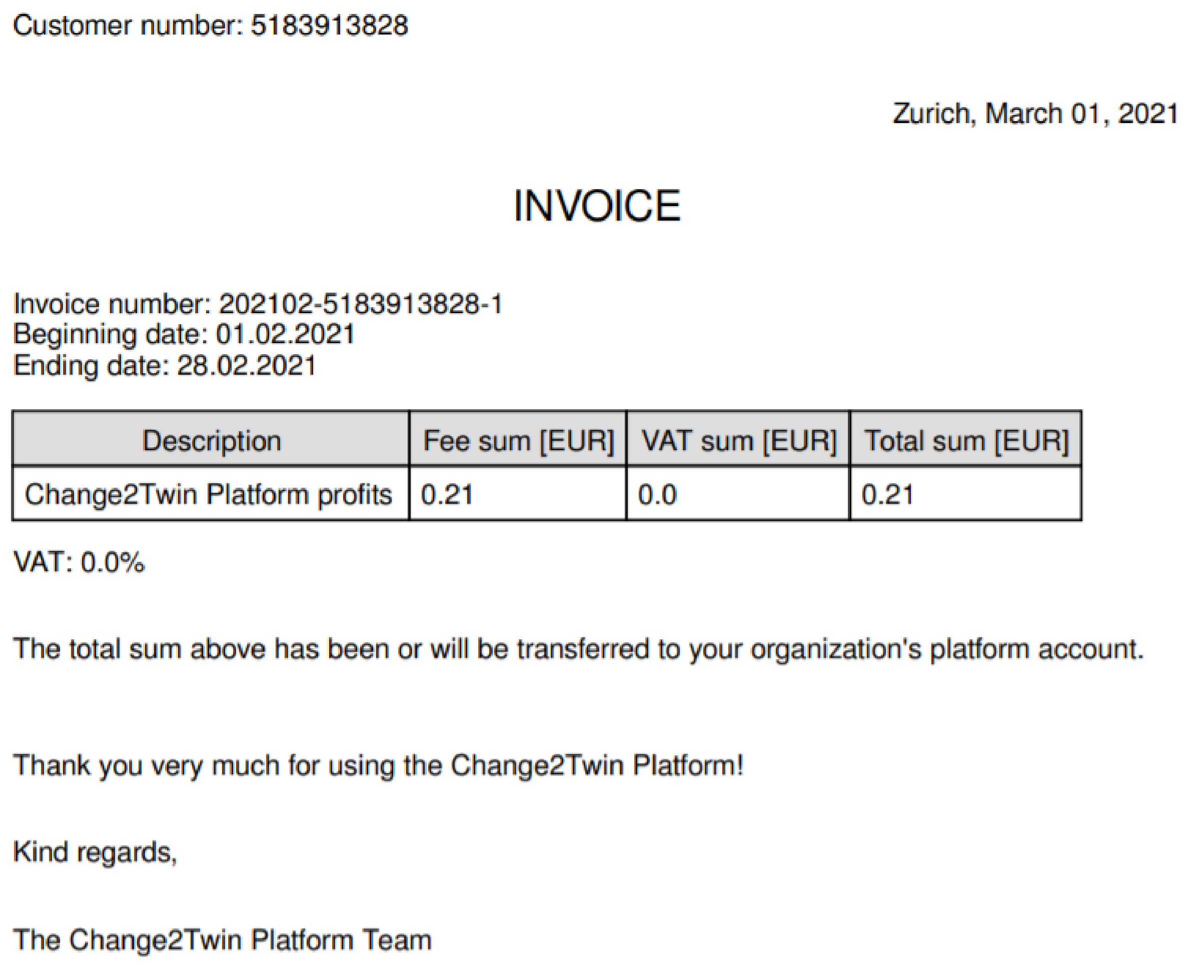
FIGURE 10 GOTOOLS MOCKUP

User management

It is planned to use a single user management tool for Marketplace administrative/operation purposes and for managing different type of users in the Marketplace (end-users, service providers, offering owners, etc.). The technical details of the tool are discussed in Section 3.1.

Invoice generation

Middleware functionality can be extended to use the current invoice generation on CloudBroker platform. Platform calculates the sum with respect to user's infrastructure use, cloud provider's cost (licenses, storage, hardware), platform fees and VAT.



Customer number: 5183913828

Zurich, March 01, 2021

INVOICE

Invoice number: 202102-5183913828-1
Beginning date: 01.02.2021
Ending date: 28.02.2021

Description	Fee sum [EUR]	VAT sum [EUR]	Total sum [EUR]
Change2Twin Platform profits	0.21	0.0	0.21

VAT: 0.0%

The total sum above has been or will be transferred to your organization's platform account.

Thank you very much for using the Change2Twin Platform!

Kind regards,

The Change2Twin Platform Team

FIGURE 11 INVOICE EXAMPLE

Such method could be adapted to middleware side, to calculate usage/purchase of marketplace items and generate monthly invoice directly to user. In PDF file, see Figure 10, user will be able to see what services were used and what is the total cost for it.

Offering payments / usage plans

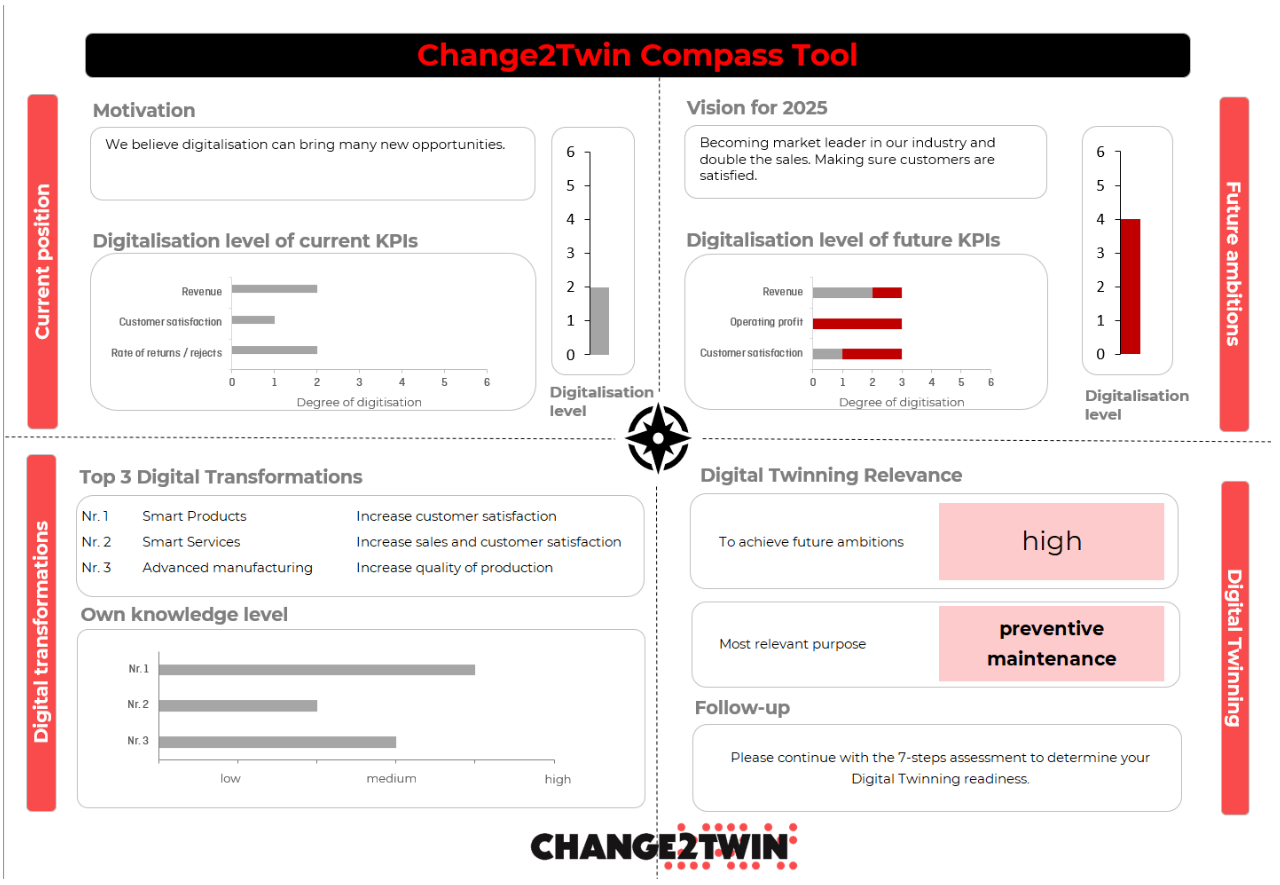
Deployable service and direct usage of an offering via the Marketplace will require introducing a solution to define how the offering can be use and how it should be charged. Such functionality will be provided after shopping cart feature is enabled, and discussed in the next report.

3.5 ASSESSMENT TOOL

Status: operational

The compass tool helps companies to determine which application of digital twin might be most useful for them given their business goals and the readiness assessment indicates which components companies need to create that type of application. The assessment tools are already part of the marketplace offering.

Below are two screenshots of the end-results of the compass tool (which application of digital twins., Figure 11) and the readiness tool (which components, Figure 12).



Change2Twin Compass Tool

Motivation

We believe digitalisation can bring many new opportunities.

Vision for 2025

Becoming market leader in our industry and double the sales. Making sure customers are satisfied.

Top 3 Digital Transformations

Nr. 1	Smart Products	Increase customer satisfaction
Nr. 2	Smart Services	Increase sales and customer satisfaction
Nr. 3	Advanced manufacturing	Increase quality of production

Digital Twinning Relevance

To achieve future ambitions	high
Most relevant purpose	**preventive maintenance**

Follow-up

Please continue with the 7-steps assessment to determine your Digital Twinning readiness.

FIGURE 12 COMPASS TOOL

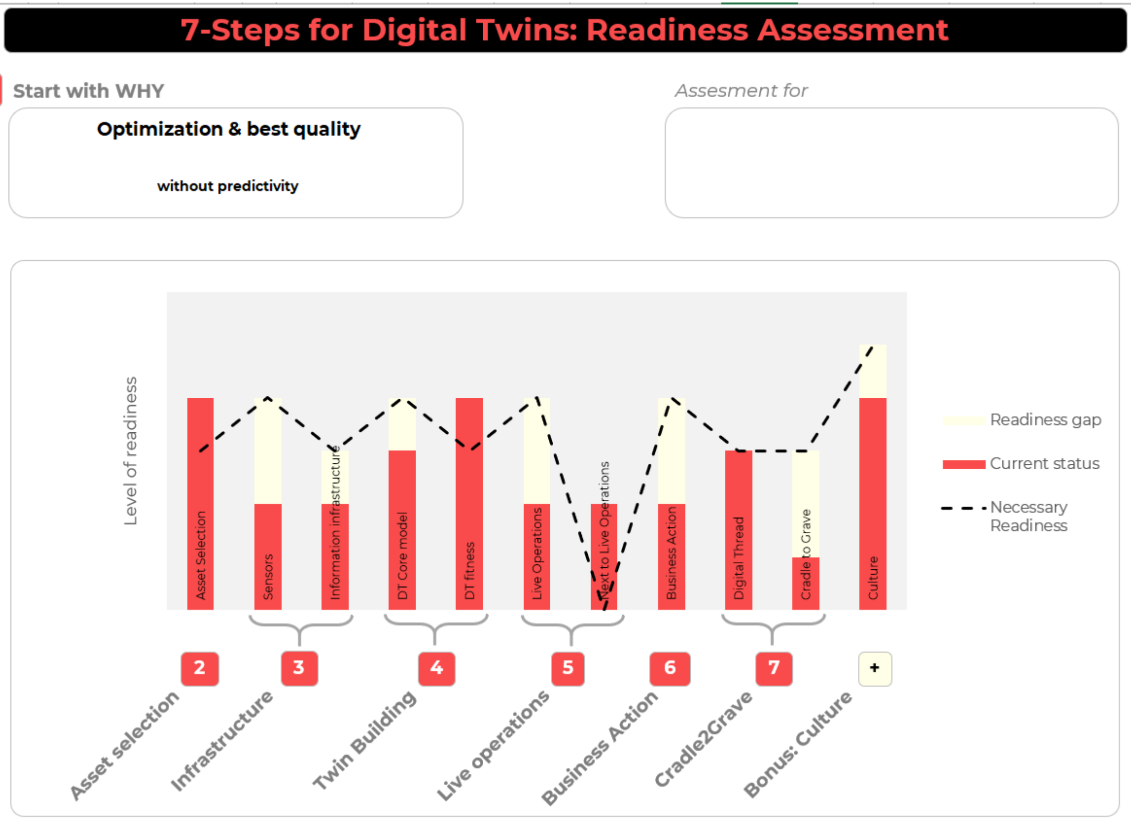


FIGURE 13 READINESS ASSESSMENT

The output of both tools can be used as an (automated) filter for the use of the marketplace, selecting the right type of components relevant for a specific type of application a company needs. Since the assessment can be done online, it can be directly connected to the marketplace. In its most simple form this requires a feature that specifies key tags/filters/search-words in the URL of the marketplace that automatically activate filters so a user doesn't need to go through all items by himself. This way of integration can also be easily scaled to other solutions that want to highlight specific components or parts in the marketplace, allowing for more referrals.

3.6 ADVANCED SEARCH TOOL

Status: Operational, in progress

The Marketplace model defined in ADOxx integrate a semantic search for marketplace items based on an association between each item and semantic concepts defined in a Triple Store. **This enables the user to find the most fitting marketplace item for a specific need**, exploring not only from items of a single marketplace but from all the items imported from different marketplaces (Figure 13).

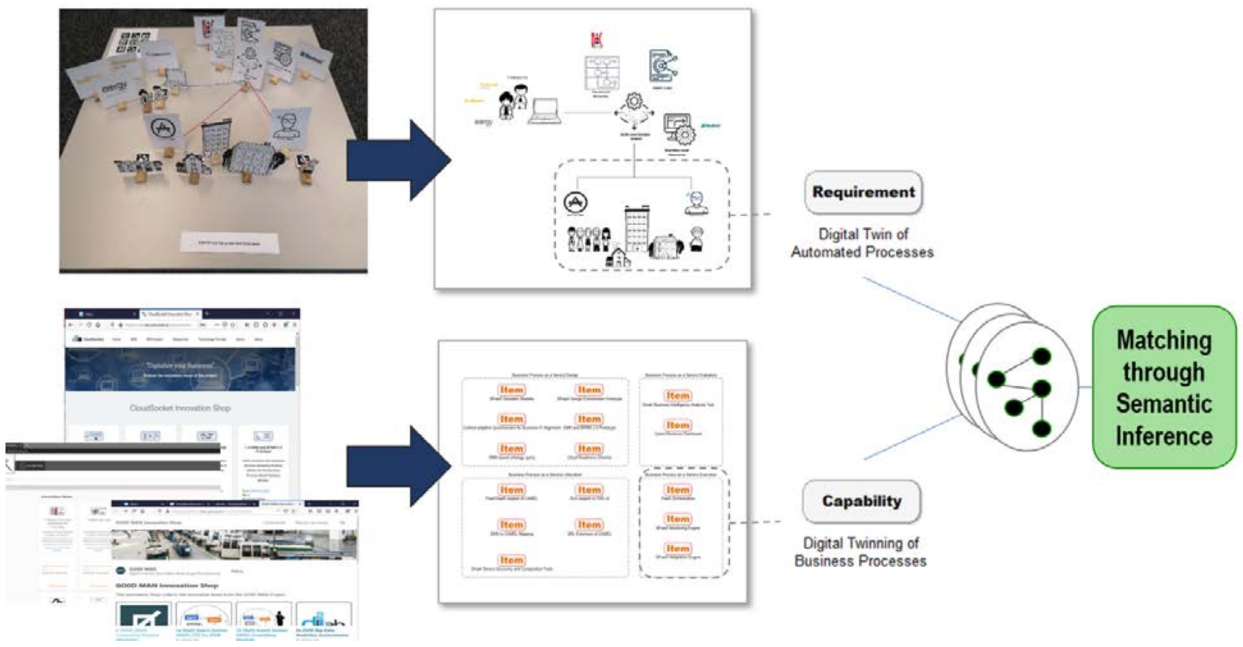
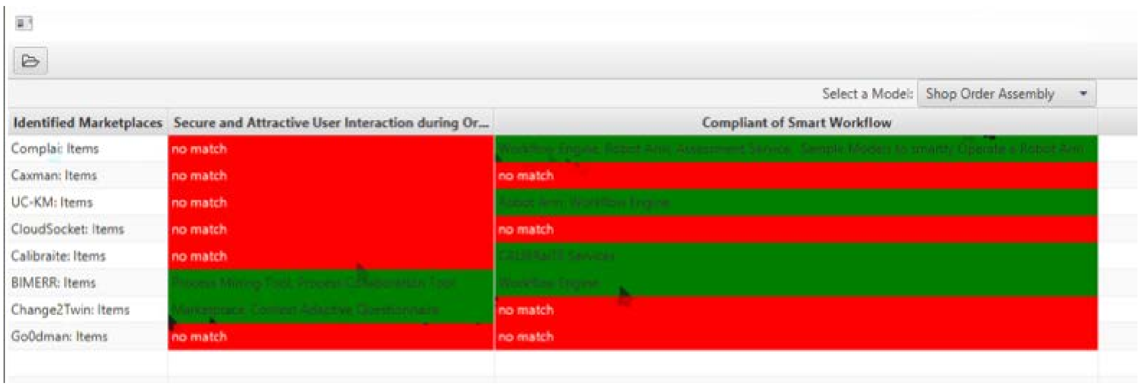

Requirement
Digital Twin of Automated Processes
Capability
Digital Twinning of Business Processes
Matching through Semantic Inference
CloudSocket Innovation Shop
GOOD MAN Innovation Shop
Item

FIGURE 14 REQUIREMENTS MODEL

A specific model type is introduced for this scope in order to define the capabilities of the marketplace item and the user requirements. The requirements can be defined manually or extracted and associated to scenes representing the use case. A semantic matching is then performed in order to infer which item fit better the needed requirements. The matching return a matrix view where, for each marketplace model available (in the rows), all the items matching each requirement (in the columns) are visualized (Figure 14). In the next release it is planned to provide ranks of the semantic matching.


Select a Model: Shop Order Assembly

Identified Marketplaces	Secure and Attractive User Interaction during Or...	Compliant of Smart Workflow
Complai: Items	no match	Workflow Engine, Robot Arm, Assessment Service, Sample Models to smartly Operate a Robot Arm
Caxman: Items	no match	no match
UC-KM: Items	no match	Robot Arm, Workflow Engine
CloudSocket: Items	no match	no match
Calibraite: Items	no match	CALIBRAITE Services
BIMERR: Items	Process Mining Tool, Process Collaboration Tool	Workflow Engine
Change2Twin: Items	Marketplace, Context Adaptive Questionnaire	no match
Go0dman: Items	no match	no match

FIGURE 15 SEMANTIC MATCHING

The functionality goes to enrich the query search feature of ADOxx that enables a search based on keywords and item metadata like TRL or license type.

3.7 3RD PARTY MARKETPLACE CRAWLER

Status: operational, in progress

The Marketplace Crawler is a microservice created using the OLIVE framework, **able to retrieve marketplace items from different type of online marketplaces and add them to a Marketplace model** defined in the ADOxx platform (Figure 15).

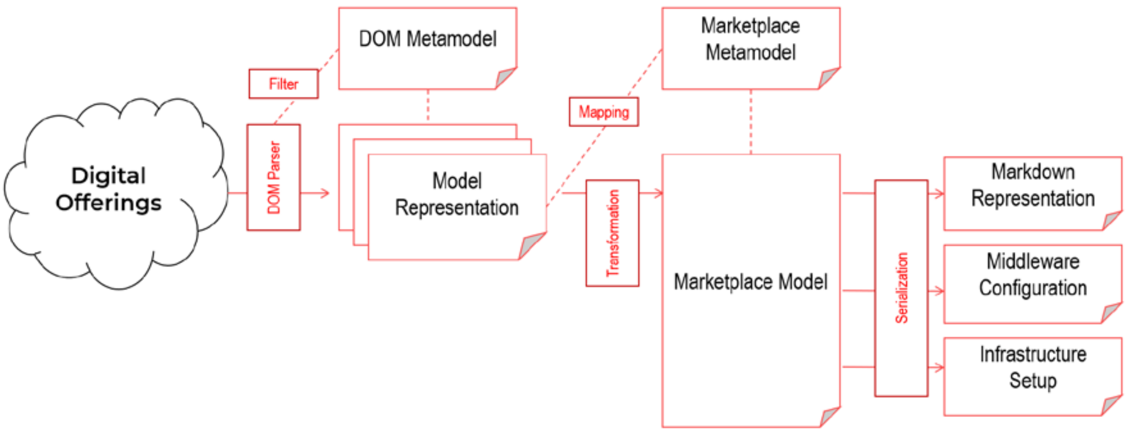


FIGURE 16MARKETPLACE CRAWLER ARCHITECTURE

The service performs this operation in different steps: first the marketplace HTML document object model (DOM) tree is parsed and mapped imported into a generic DOM model in ADOxx, then each marketplace item in the DOM model is mapped and referenced to an item object in a specific Marketplace model. This mapping process is not fully automatic as each marketplace has its own HTML structure and the user input is required in order to identify the marketplace item in the DOM tree. Finally, the referenced DOM item is converted in Markdown format in order to align with the marketplace item format of the Change2Twin Marketplace.

As soon as a marketplace model is generated, all its items can be copied and merged with the Change2Twin marketplace model that can finally be exported as markdown items and published in the marketplace portal. The onboarding process can be applied in the middle of the process for the sake of curation.

4 SUMMARY

In this deliverable we discussed marketplace portal, current status of the tools and features built inside or around the Marketplace. These developments are going according to plan. During the next months our work will focus on:

- **Improving UI/UX.** A specific modelling method for the UI, following the approach of the modelling method for microservices is still missing and has to be released in future versions of the framework. Currently the Marketplace Model type in ADOxx, described in deliverable D2.2, enables the definition of the marketplace items and their content using models. This model type will be extended in order to also support the definition of the layout and the UI for the visualization of the marketplace. Improvements in the widget catalogue and with respect to the configurations are also planned in future releases in order to support more use cases. An extension will also be created for the orchestration component in order to support composition of existing services. Finally more target platforms will be supported. The focus here will be in particular to native mobile applications for smartphones and IoT devices, like Arduino based embedded screen devices and augmented reality smart-glasses.

- **Providing a shopping cart**. This will allow users to take advantage of the Marketplace, by being able to register, browse the catalogue using advanced search capabilities or directly use the offering deployed on the infrastructure. This is the entry point to extend other components, such as monitoring to analyse users and offerings usage, or user management to advance with new roles and privileges.

- **Finalize the onboarding tool.** Providing almost-fully automation to this process will allow to save significant amount of time, required currently for providing content to the ticketing system, revising it and eventually pushing to the Marketplace. Saved time will be used to survey end users or to find ways to increase number of available offerings.

- **Introducing deployable services.** Technical solutions will be provided to allow users to use offerings directly via the Marketplace. To this end, issues related to keeping history of usage, providing input for the services, notifications or retrieving results have to be solved.

- **Exchange information with other marketplaces.** This is one the most important added value. It will bring possibility not in just having more offerings in the marketplace, but in particular to build marketplace around offerings within a given domain, specifically tailored for its users. And it will avoid having a global marketplace, shifting towards open, federated environment. The work is in progress, though there are legal issues to be solved before announcing such feature to the public.

Coming closer to the middle of the project, it is high time to shift focus on user side of the story and their expectations. We have already identified this as a challenging and non-trivial task. The actions need to be conducted by WP6 with help of WP1, some of them may require additional work inside WP2, while others remains under WP6 wings. Following challenges are identified:

- To define roles for different users, survey them, analyse their requirements and address their needs;
- Conclude with a common understanding on taxonomy;
- Increase number of offerings and make them more competitive;
- Liaise with other marketplaces to exchange information;
- Encourage DIHs to use our marketplace model;
- Improve UI/UX by better understanding different user expectations.

APPENDIX A TECHNICAL DETAILS OF THE PORTAL

A1 ARCHITECTURE OVERVIEW

The micro-frontend framework enables the definition of a multichannel UI and in the C2T Marketplace the UI has been defined only to target web browsers for personal computers and mobiles. The widgets used includes a Grid UI configured to use data coming from a microservice that retrieve the list of all the marketplace items, and a Content UI that is visualized every time an item is selected, configured to communicate with a microservice responsible to retrieve the marketplace item details. The items details are defined as Markdown files, enriched with item specific content (images, zip packages, pdf, etc.), in separate GitLab repositories. There is one Markdown file for each marketplace item, in order to handle users' authorization on content edit rights. The main index UI manages the composition of the UI logic and is responsible to provide the UI for common features like user login while the deployment list UI visualizes the available cloud deployments for the current user. In addition to the microservices for retrieving the marketplace item list and the item details, in the microservice controller have been instantiated three microservices communicating with the ADOxx platform. Two of those microservices for generating marketplace items form a specific marketplace model type: one for populating a marketplace model with items retrieved by crawling existing marketplaces, and one for evaluating the assessment of requirements defined in the scene model type with respect to the capabilities provided by the items in the marketplace model. The marketplace content editors can in this way provide their contribution both by editing the GitLab spaces as well as by defining the items in the marketplace model in the ADOxx platform. They can find the most fitting marketplace item for a specific use case using the assessment and scene model types. A detailed description of the marketplace functionalities and the specific ADOxx model types defined, is available in Deliverable D2.2. Figure 16 introduces the details of the OLIVE framework components used to create the instances of the microservices and the UI elements for the C2T Marketplace web application.

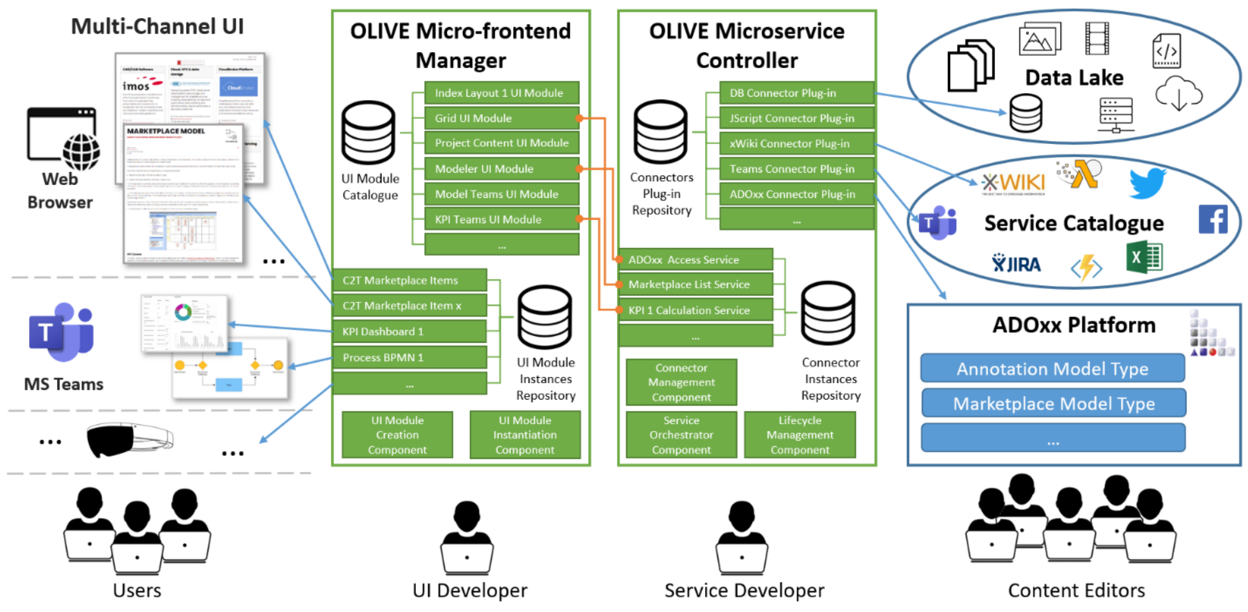


FIGURE 17 MARKETPLACE HIGH LEVEL COMPONENT'S ARCHITECTURE

The OLIVE framework allows to create model aware web applications through configuration of existing components, both for the backend and for the frontend side. For the backend side such components are named Connectors, and their configuration results in ready to use REST microservices. For the frontend side such components are named Widgets and their configuration results in a multi-channel, ready to use, user interface. Both the Connectors and the Widgets are part of the OLIVE platform but can be extended if needed by using plug-ins. Connectors provide the functionalities of the backend services enabling the connection to external systems like databases, CMS, or message buses. The framework provides out-of-the-box connection to more than twenty services and data storage systems. As soon as a Connector is configured a new microservice instance is created, executed, and exposed through a REST interface with standardized input and output format. The lifecycle of the service can then be controlled by the framework. Instantiated services can be orchestrated for a more complex business logic if needed. This configuration approach enables the definition of specific models that reflect the microservice definition. The integration with the ADOxx platform provides a complete environment where (a) the microservices and UI elements can be created from scratch starting from models and (b) models can be used as data with recognized semantic by the microservices, like in the case of the marketplace model type.

Widgets on the other side are reusable components for the frontend and provide the UI for sections of the application targeting a specific channel. The presentation channels most supported by the framework are web browsers, for both desktop and mobile devices, and MS Teams pages. Currently, only limited support is provided for other channels like IoT devices and mobile apps. Widgets can be generic like visualizing a grid layout or more specific like visualizing a marketplace item page and supports the connection with microservices defined in the framework in order to obtain the needed data or provide the user experience for the microservice feature. Like the Connectors for the microservices, also the Widgets can be configured with specific parameters.

Their configuration results in new UI instances that could be then combined and exposed to a specific endpoint or device.

The strength point of Olive is its model awareness in the sense that such configurations are abstract enough that can be represented as models and the out-of-the-box integration with the ADOxx modelling environment allows to create the whole looks and behaviour of the application, drawing models. This integration allows also ADOxx to communicate with the external world through a common interface in a bi-directional way, so using the features of existing microservices to enrich the modelling platform and using models as data for microservices. The Figure 17 shows the approach used by the framework for the definition of applications starting from conceptualization in models, which is used for configuration of existing components, resulting in different running instances of the application.

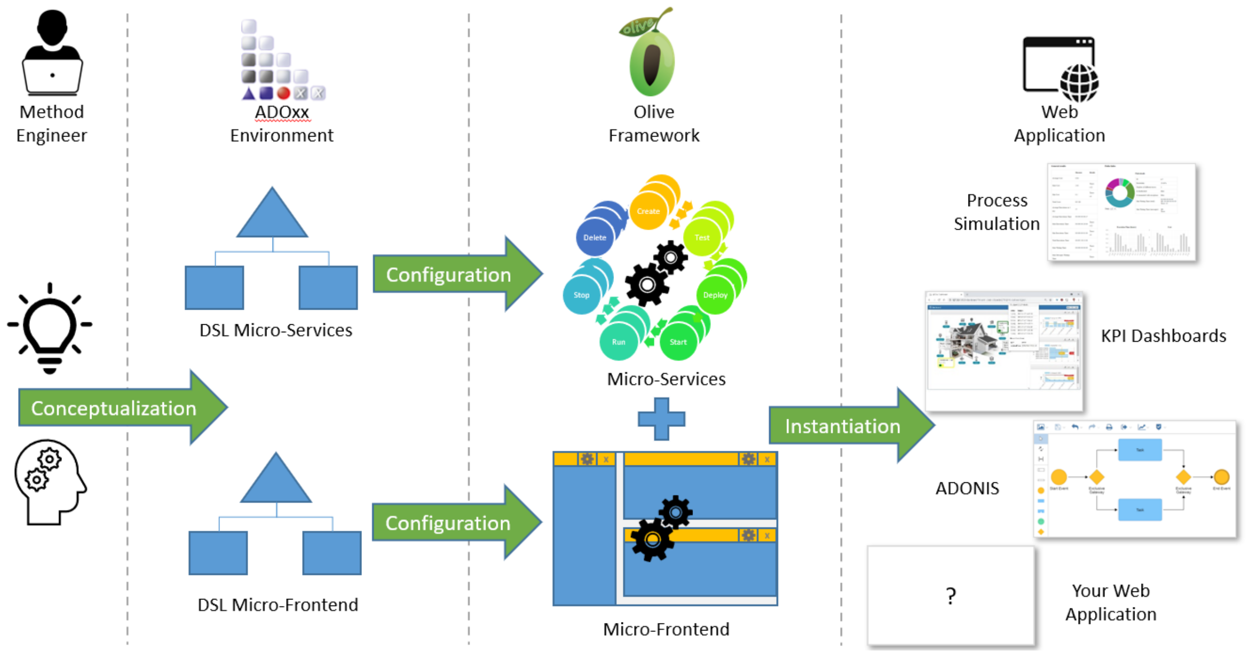


FIGURE 18 OLIVE WEB APPLICATION DEFINITION METHODOLOGY

In this way, the Olive platform provides a cloud environment where the user, instantiating existing components with specific configuration, can define the microservices (that can use and be used by ADOxx) and the user interfaces of its application, expose it to the public and control its lifecycle.

A2 OLIVE MICROSERVICE CONTROLLER

The Olive Microservice Controller is a backend component that allows to define and manage Microservices in a novel way, following the configuration approach. A Microservice in Olive is defined only through the configuration of an existing platform component named Connector.

A Connector is a component developed in form of OSGi plug-in that allow to provide a specific functionality, like perform a query on a MySQL database or publish a post on Twitter. The name Connector derives from the fact that usually such functionalities depend on external systems (like the database) and the Connector is responsible to connect to such systems to exploit their features.

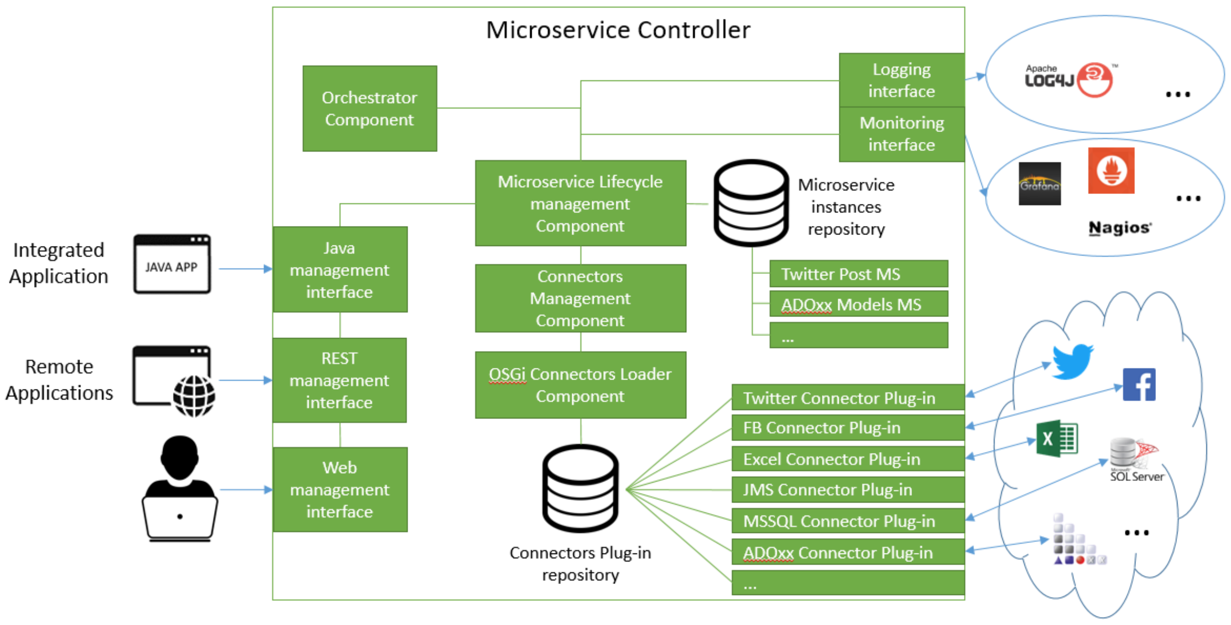


FIGURE 19 OLIVE MICROSERVICE CONTROLLER ARCHITECTURE

Olive Microservice Controller (Figure 18) allows to manage the configurations of such Connectors, giving the possibility to create Microservices and control their whole lifecycle. It is responsible of the Lifecycle management component to (1) generate an instance of the REST microservice from the configuration, (2) allow to start the microservice, (3) keep the microservice running in an isolated environment, (4) allow to stop the microservice and (5) allow to dismiss it.

The OSGi Connectors Loader component is responsible to load all the Connectors and make them available to the platform. It is built on the OSGi framework Apache Felix and will dynamically check the presence of the OSGi bundles (plug-ins) defining Connectors, loading, and unloading them on request.

As soon as the Microservices have been defined, they can be combined to achieve the business logic task thanks to the Orchestrator component. This component is responsible to combine existing microservices using in future the Enterprise Integration Pattern notation. To support a higher level or freedom the orchestrator allows also to use the JavaScript scripting language to combine microservices following so a more programmatic approach.

The Olive Microservice Controller exposes all of these functionalities both with Java and REST APIs. The former is used to integrate the Olive platform in local and desktop application. The latter are used to integrate the Olive platform with remote applications. All the microservice controller features are available also through a management web UI, built on top of the microservice controller REST APIs, that enables the management of each microservice lifecycle.

A2.1 OLIVE MICROSERVICE DEFINITION

A microservice in Olive is defined through a JSON file that contains the configuration of a specific connector. The Olive platform can use this configuration file to create an instance of the connector and expose it through a REST API. A connector is a function provided by the Olive framework, responsible to perform a specific operation. An example is the MySQL connector that allows to query a MySQL database, or a Twitter connector that allows to post and retrieve posts on Twitter.

The connectors are structured in a way that allow to specify which part of the specific operation must be performed at microservice start, execution and stop. As example, the MySQL connector will establish the connection with the database during the microservice start phase, will perform the query during the execution phase and will close the connection during the stop phase. This allows to support connection pooling and reuse the same database connection for all the microservice requests, increasing the response time. The configuration of the connectors is relevant to the start and execution phases. Considering again the MySQL connector, the configuration of the start phase (that perform the connection to the database) allows to specify the database endpoint address and port, the database name, and the access credentials, while the configuration for the execution phase allow to specify only the query to perform.

With this configuration Olive generates a REST service that connects to the specified database and perform the configured query, returning the results in a tabular format as defined by the connector.

Olive integrates out-of-the-box 24 connectors. Custom connectors can be added to the platform as OSGi plugins. This allows to reuse existing OSGi based connectors like all the one provided by the Apache Camel project.

Olive distinguishes two kinds of connectors, depending on the communication pattern required:

- Synchronous connectors: types of connectors that provide a functionality on request. Such kind of connectors are used to create REST microservices that once called, perform some operations and return the results to the users. An example is the MySQL connector that is used to create microservices that on user request will perform a query to the database and return the result to the user.
- Asynchronous connectors: types of connectors that perform operations mainly in background. The Olive platform creates REST microservices also from that connector, but they do not return the operations results to the users but in

contrast start to process the request in background. Such kind of services may also not require the interaction with the users at all. Due to that fact, the Olive platform allows to attach to such microservices a previously defined synchronous microservice used to process their results. A typical example is a microservice that listens to a message bus. This microservice continuously checks in background the presence of new messages and as soon as a message is received will forward it to a microservice responsible to store it in a MySQL database.

We can summarize that the synchronous connectors are used to create microservices that start to work as soon as the user requests them, while asynchronous connectors create microservices that start to work as soon as the microservice is started.

Even though the main business logic of the microservice is provided by the used connector, the inputs, and the output format of the microservice can be adapted. Those adaptations are also specified defining them in the microservice configuration.

Olive allows also to check the status of the microservice. By default, Olive automatically recognizes if a microservice is started, stopped and if its connector incurred in an error. In addition, the user can define how the output should look like in terms of format and data content. This allows to perform a deeper status check taking into account also the semantic of the output.

Microservices in Olive are organized in structures named Operations. Operations are methods that relate to the same microservice and are the objects that contain the configuration of the connectors, the definition of the inputs and the adaptation of the outputs. Operations are uniquely identified by their name inside a microservice, which is instead identified through a unique ID.

A2.1.1 MICROSERVICE INPUTS DEFINITION

The definition of inputs for a microservice allows to have the microservice configuration partially customizable by the user through its inputs. In the microservice definition it is possible to define which inputs are required by the final users and how these inputs will impact the microservice configuration. Since the final user interacts with the microservice only during its execution (and not during the start and stop phases), the inputs can affect only the microservice configuration section related to the execution phase. As an example, we consider a microservice that uses the MySQL connector to perform a query to a database. This connector allows to configure the database endpoint and credentials used during the starting phase of the microservice. This means that such information cannot be customized asking the final user for inputs. The configuration of the query to perform is defined instead in the execution phase, so we can create a microservice input definition that customizes the query. This means that we can or ask the whole or only a part of the query to the final user as microservice input.

The customization of the microservice configuration through microservice inputs is done using a match and replace mechanisms. In the configuration of the microservice it is possible to specify placeholders that will be replaced (previous validation) at execution time with the matching microservice input provided by the final user. So, the definition of a microservice input requires only a unique name for the input and the name of the placeholder that is used in the configuration for the execution phase.

Using the previous example and imagining querying a database with the following SQL statement:

```
SELECT name FROM users WHERE mail="Damiano.falcioni@boc-eu.com";
```

If we want to have the mail as microservice input, we must first add a placeholder to the query like:

```
SELECT name FROM users WHERE mail="$mail_input_placeholder";
```

And then define an input with name "`mail_ID`" and placeholder "`$mail_input_placeholder`".

When the final user calls the microservice REST endpoint, an input like the following JSON object is part of the POST request:

```
{
  "mail_ID": {
    "value": "Damiano.falcioni@boc-eu.com"
  }
}
```

The value provided for the input "`mail_ID`" will replace the placeholder string "`$mail_input_placeholder`". The resulting query will be performed, and the result returned to the final user.

A2.1.2 MICROSERVICE OUTPUTS DEFINITION

When defining a microservice it is possible to adapt the output of the configured connector used in the microservice, providing an algorithm in the JavaScript programming language. Such an algorithm can be used to parse the connector output and convert it in the required format or do complex data processing operations. Due to the high level of freedom left to the user, in this case there are some configurable restrictions about the maximum allowed execution time of the algorithm and about the allowed operations (for example operations on file system are denied). Despite such restrictions the following additional data variables and functions are available:

- *output*: a variable containing the JSON object returned by the connector;
- input: a variable containing the JSON object provided as input by the final users to the microservice POST endpoint;

- *out({..})*: a function used to return the final adapted output. This function must be called as the last instruction of the adaptation JavaScript. it accepts as an input a JSON object;
- *callMicroservice(microserviceId, operationId, microserviceInputs*): a function used to call an existing microservice and obtain its output. It is used to ease the creation of complex adaptation scripts by reusing existing microservices and can be used also to chain together microservice functionalities. It requires as input the unique id of the microservice, the id of the microservice operation to perform and a JSON object containing the required microservice inputs. It returns a JSON object containing the output of the called microservice.

A2.1.3 MICROSERVICE DEFINITION WITH ADOXX MODEL

The Microservice Definition Model type is an ADOxx library that allows to model the exact behaviour of microservices and publishes them in the Olive Microservice Framework. The definition of microservice using models is possible thanks to the nature of the Olive platform that defines microservices through configuration. In this way the models can be used not only to document but also to configure the microservice behaviour.

A specific meta-model in ADOxx allows to model the definition of microservices and how they should be orchestrated. The Olive microservice framework can communicate with ADOxx in order to retrieve such models and define based on them the different microservices.

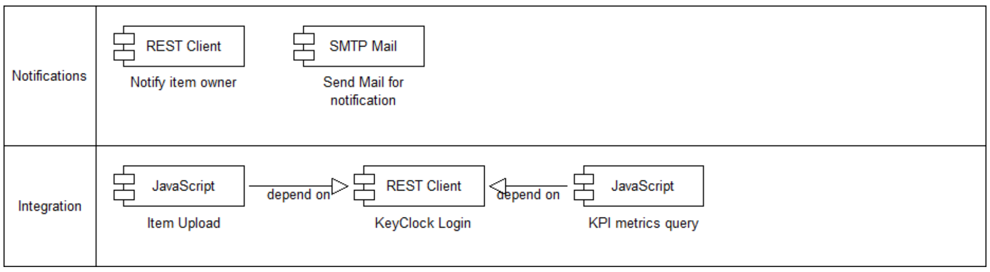
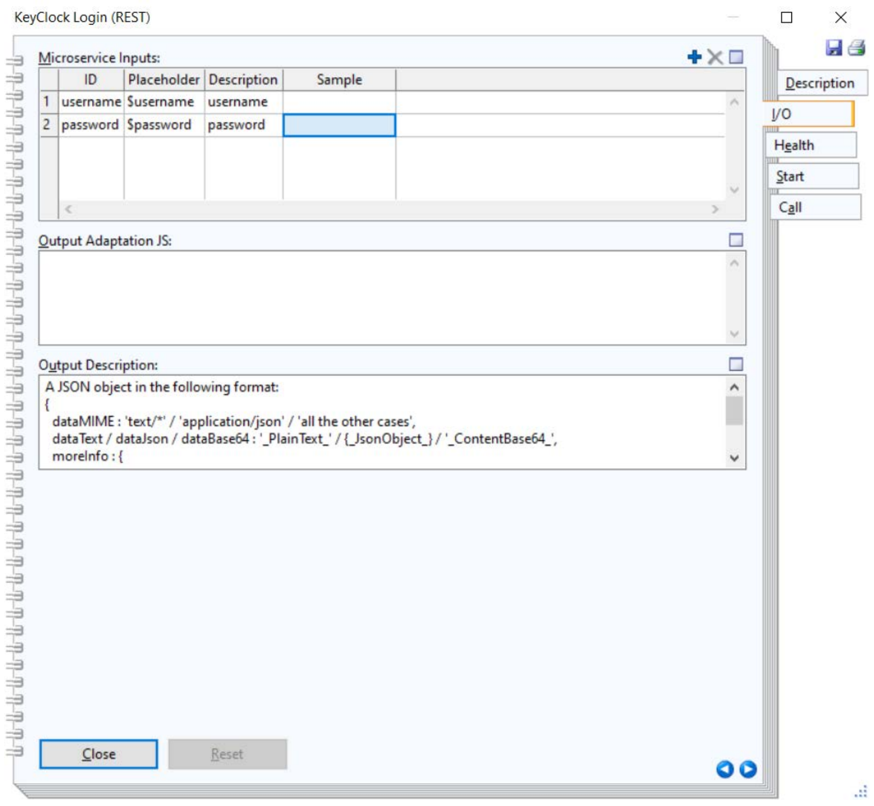


FIGURE 20 MICROSERVICE DEFINITION MODEL SAMPLE

A Microservice Definition Model is composed of objects representing microservices of different types, based on the Olive Connector used, and relations representing dependencies between microservices, see Figure 19 for an example. Each of the specific microservice type object has a set of common and specific attributes accessible from its notebook. Common attributes are the description of the microservices and its auto-start value as well as all the attributes related to the input and output. The microservice inputs can be specified in a tabular form where each row is an input with information about its id, placeholder, description, and sample. The output can be instead adapted providing a specific JavaScript algorithm with the relative description of the new output. Figure 20 presents microservice definition with inputs and outputs.



Microservice Inputs:

	ID	Placeholder	Description	Sample
1	username	$username	username	
2	password	$password	password	

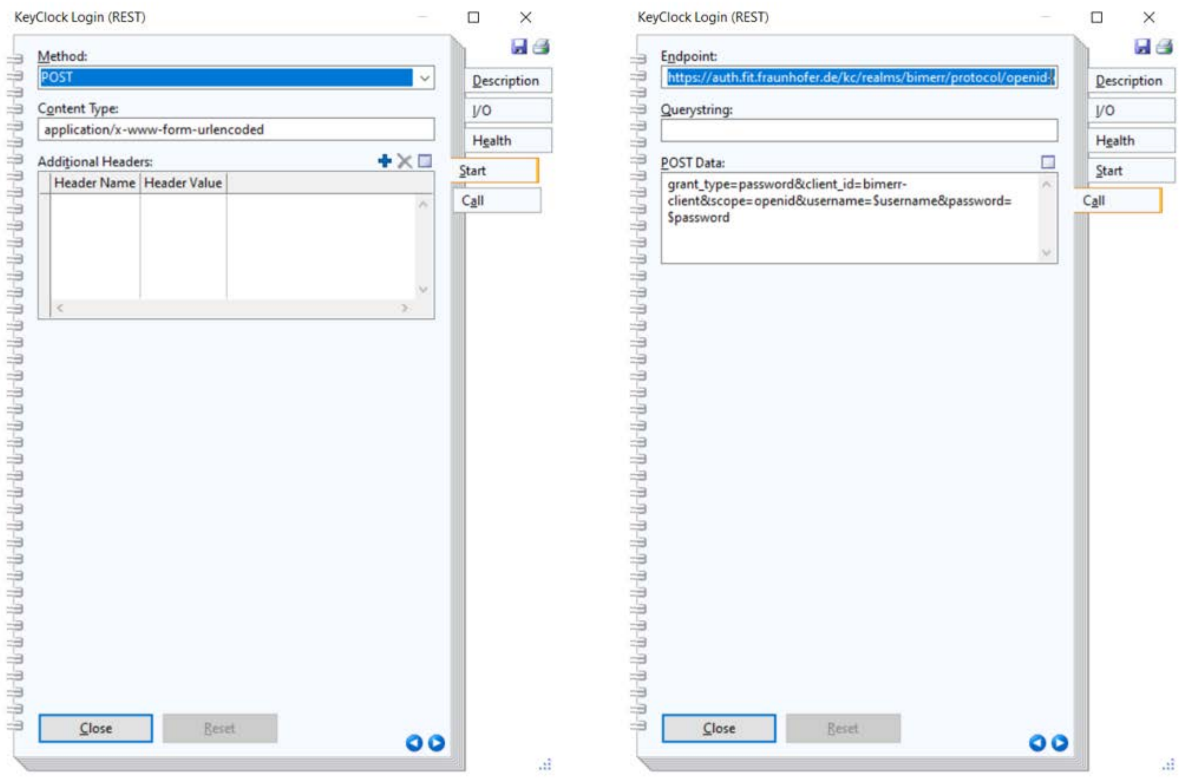
Output Adaptation JS:

Output Description:

```
A JSON object in the following format:
{
  dataMIME : 'text/*' / 'application/json' / 'all the other cases',
  dataText / dataJson / dataBase64 : '_PlainText_' / {_JsonObject_} / '_ContentBase64_',
  moreInfo : {
```

FIGURE 21 MICROSERVICE DEFINITION MODEL, INPUT AND OUTPUT ATTRIBUTES

Health information is the last set of common attributes to all microservice objects and, like in the web interface, the user is allowed to specify the JavaScript algorithm that will be applied in order to validate the microservice output and check if the service is running properly or not.


KeyClock Login (REST)

Method: POST

Content Type: application/x-www-form-urlencoded

Additional Headers:

Header Name	Header Value

KeyClock Login (REST)

Endpoint: https://auth.fit.fraunhofer.de/kc/realms/bimerr/protocol/openid-

Querystring:

POST Data: grant_type=password&client_id=bimerr-client&scope=openid&username=$username&password=$password

FIGURE 22 MICROSERVICE DEFINITION MODEL, START AND CALL ATTRIBUTES

The notebook sections named “Start” and “Call” contain instead the microservice specific attributes dependent on the Olive Connector used. In the example in Figure 21, the microservice is based on the Olive REST Connector and the “Start” related attributes allow to specify the HTTP method, Content type and headers of the REST request to perform, while in the “Call” section there are attributes related to the REST endpoint, query-string, and the optional data to post.

As soon as the microservice model is completed it can be published directly from the modelling environment thanks to the “Publish Microservice” function available in the “Extra” menu bar. The function will prompt the endpoint of the Olive Microservice framework used for the deployment, and it will automatically make it live.

A2.1.4 CONNECTORS COLLECTION

Olive provides out-of-the-box the following connectors:

- **ADOxx Classic Connector**: This connector allows to communicate with the ADOxx Modeler Classic (desktop application) exploiting its SOAP interface. This allows to execute custom AdoScripts (only a restricted set is allowed) remotely and create microservices that interact with models.
- **Camunda DMN Engine Connector**: This connector allows to evaluate a DMN model using the DMN engine provided by Camunda.

- **Content Provider Connector**: This connector allows to provide an arbitrary content to download. It is used to create microservices that return previously uploaded data.
- **Content Receiver Connector**: This connector allows to upload an arbitrary content and store internally in the platform. It is used to create microservices that upload data needed by other microservices.
- **KPIs Engine Connector**: This connector allows to calculate the KPIs, and metrics defined in a KPI model and return their value. The connector can interpret the model, identifying dependencies between KPIs and evaluate their value and success status.
- **Excel Connector**: This connector allows to read values available inside an Excel document, evaluating formulas and controlling the cells to read.
- **Facebook Connector**: This connector allows to interact with the Facebook Graph API. This allows as example to publish and retrieve post, comments, find users, etc.
- **Fuseki Triple-store Connector**: This connector is used to publish and retrieve content from the Fuseki Triple-store. It allows to perform SPARQL queries to interact with the ontology defined in the triple-store.
- **HP Printer Monitor Connector**: This connector allows to retrieve the colour level of a network connected HP Printer. This is used to create microservices that continuously monitor the status of an HP printer and notify the administrator in case of problems.
- **JavaScript Engine Connector**: This connector allows to execute a JavaScript code evaluating it using the Java Nashorn engine. The JavaScript is executed in a restricted environment for security reasons, and this allows to create general purpose microservices that perform any kind of operation available in the JavaScript programming language.
- **JMS Publisher Connector**: This connector is used to connect to a message bus compatible with the JMS standard and publish some content on a specific topic.
- **JMS Subscriber Connector**: This connector is used to connect to a message bus compatible with the JMS standard and listen for new messages on a specific topic. This is a connector that follows the asynchronous communication pattern.
- **Microservice Connector**: This connector is used to schedule the execution of an existing Microservice and run it in background. This is used to convert every synchronous microservice in an asynchronous one.
- **MSSQL Connector**: This connector allows to perform a generic SQL query in a remote Microsoft SQLServer database. This allows to both insert and retrieve data from the database.
- **MySQL Connector**: This connector allows to perform a generic SQL query in a remote MySQL database. This allows to both insert and retrieve data from the database.
- **OMiLAB Reservation Service Connector**: This connector allows to control the access to another existing microservice, using the functionalities of the OMiLAB Reservation service that allows to reserve a device for a specific time slot generating a secret token that must be used to access the services on that device only during such a time slot.

- **REST Connector**: This connector allows to communicate with existing remote services exposed through the REST protocol. It can be used to create microservices that exploit the features of external services through a wrapping/proxying around the original service functionalities.
- **SMTP Connector**: This connector allows to interact with a remote SMTP server to send an e-mail programmatically.
- **SOAP Connector**: This connector allows to communicate with existing remote services exposed through the SOAP protocol. It can be used to create microservices that exploit the features of external services through a wrapping/proxying around the original service functionalities.
- **Twilio SMS Connector**: This connector allows to exploit the SMS sending functionalities provided by Twilio. It can be used to create microservice that send SMS programmatically like in response to specific events, acting as notification mechanism.
- **Twitter POST Connector**: This connector allows to interact with the Twitter API to post a tweet in the configured account.
- **Twitter Search Connector**: This connector allows to interact with the Twitter API to find tweets matching a specific search query.
- **Twitter User Info Connector**: This connector allows to interact with the Twitter API to retrieve information about a specific tweet user.

A2.2 MICROSERVICE INSTANTIATION

As soon as the microservice has been defined, Olive makes available a REST endpoint that exposes the microservice. At this point the microservice is ready to be used. The REST endpoint requires the unique id of the microservice and the name of the microservice operation to be executed as query parameters. In contrast, microservice inputs must be provided in form of a JSON object passed as POST data. The output is returned also as JSON object with a standard structure that encapsulates the microservice output. Olive uniforms the interfaces of all the defined microservices exposing them in a REST endpoint with a POST method based on fixed query string parameters, POST data and output format.

If the microservice is called before being started, it will start automatically. The Olive Microservice framework allows also to manually control the starting and stopping phase of a microservice to deallocate resources on the machine. These operations are all manageable through the APIs of the Olive Microservice Controller framework as well as through the management interface.

During the starting phase of a microservice a new thread is executed and kept active till its stopping. In the thread, the configured connector used by the microservice is initialized. The configuration relative to the starting phase is used for this initialization. As soon as all the initialization operations are completed the microservice is in a started phase and ready to be executed. When the microservice is not used since a long time or on user request, it can be stopped. All the stopping operations of the connector are executed, and the thread will be terminated.

The thread isolation level is something that is not so common in microservice development due to its insecurity over shared variables. This is true but only if the user writes insecure code. The Olive Microservice Controller, following the approach of defining Microservices through connector configuration, does not allow the user to create insecure code. The vulnerable point is in the development of new connectors that the user may require. In this case the user is responsible for the deployment of a local instance of the Olive Microservice Controller and of its security. In contrast, if the new connector is proposed to the community, it will follow a deep review process before being released and available to download in the Olive Microservice Controller package.

A3 OLIVE MICRO-FRONTEND FRAMEWORK

The Micro-frontend framework, presented in Figure 22, applies the same methodologies used in the backend by the microservice controller to the UI level. The application's look and feel is constructed, using the micro-frontend framework, through configuration of available UI components named Widgets.

A Widget for the UI of a web application is provided in form of a standard Web-Component module that can be rendered in a Web Browser (on both desktop and mobile devices). The framework includes widgets that target different platforms, but mostly focus on Web Browsers and MS Teams pages.

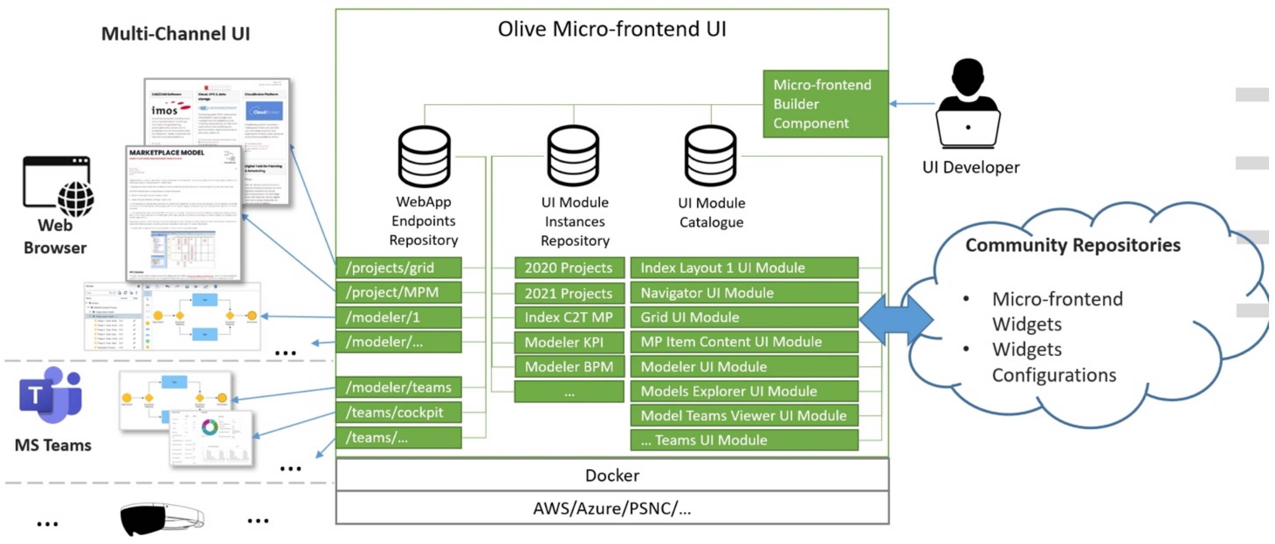


FIGURE 23 OLIVE MICRO-FRONTEND ARCHITECTURE

The OLIVE framework allows to define the configuration for each widget resulting in a specific instance of the widget. As a sample, the marketplace item visualization widget is instantiated several times, each one with the specific marketplace item content. The different instances can then be combined through the configuration of widgets specifics for the layout.

Both the widgets and their configurations are provided in a community-based repository that enables the component sharing and that can be imported into the framework for extension.

Each instantiated widget can be finally exposed publicly by the framework to a specific endpoint, responsible to deliver the widgets and its configuration to the supported clients.

A3.1 UI WIDGET DEFINITION

A Widget instance is provided in form of a JSON file containing the configuration for the specific Widget. Each widget targeting web platforms is in form of a standard Web-Component enriched with a manifest that specifies what are the required inputs and the expected results of the widget. This allows to align with latest HTML standards and reuse existing components provided by other frameworks. Additionally, as currently the most relevant web frameworks, like React or Vue, support the creation of Web-Components, their development does not require additional specific knowledge related to the OLIVE framework.

In the manifest of each widget can be optionally defined another widget responsible to provide a customized configuration UI, specific for the widget. This allows to use widgets not only for the final web application but also to enrich the user experience of widget configuration inside the OLIVE framework.

The OLIVE Micro-frontend framework enables the collection of the defined configurations for each user and make them available to other widgets, in particular to the ones responsible for the layout of the interface. This enables the composition of the final UI using small feature specific UI elements in accordance with the micro-frontend methodology.

A3.2 UI WIDGET INSTANTIATION

As soon as a widget is configured, a new entry in the widget instances repository is created and the new UI element is ready to be exposed to a specific endpoint for client delivery.

The micro-frontend framework enables the definition of different endpoints, associating them to the widget to expose the service and the configuration to use it. The page is then automatically created by delivering the Widget component to the client in form of a JavaScript file (for widgets targeting Web Browsers and MS Teams pages), the Widget configuration in form of a JSON file and rendering it in the right view. Each endpoint is associated with exactly one widget instance (including the widget and its configuration) that usually delivers the final layout.

A3.3 INTERACTION INSTANCES – AN OUTLOOK

The Micro-frontend framework supports several interaction functionalities by instantiating specific service instances like but not limited to crawling, search engine or notation and context capabilities. Following three functionalities are work in progress:

First, a crawling functionality aims at achieving developments towards the integration of existing marketplaces. The idea is to use web scrapping techniques to discover, digest, and import items from other marketplaces and hence provide content from external sources to the marketplace. This includes a model-to-model transformation. A crawling microservice is considered to read external marketplaces and/or webpages. As a result of the crawling, the content will be retrieved and rewritten for creating marketplace items. This is done by a transformation to Markdown files describing the individual marketplace items. Currently, this functionality consists of two main components, a parser retrieving the content from external sources and an AdoScript file transforming the content for being further processes in form of marketplace items.

Second, a context and notation functionality aim at allowing an improved interaction between external processes and systems with the marketplace model items. Currently, this functionality is implemented by using annotations semantically leveraging the marketplace items. The items in the marketplace model are annotated with context details in form of RDF ontologies. This annotation approach enables a targeted search for capabilities related to the annotations. Furthermore, semantic similarities can be easier retrieved between marketplace items and external processes or systems.

Third, a search engine functionality aims at improving the user-friendliness of the marketplace. On the one hand, traditional keyword searches can be supported as well as on the other hand a smarter search behavior can be offered based on the aforementioned annotations. In contrast to traditional search functionalities where often exact keyword or full text searches are applied, the smart annotation-based search approach also retrieves “similar” (in terms of semantic distance) among the search results.

APPENDIX B IDENTIFIED REQUIREMENTS

TABLE 3 INDETIFIED REQUIREMENTS AND EXPACTATIONS FOR MARKETPLACE

From	Requirement/expectation	Status
WP2 partners	Advanced search capabilities	Last developments
	Crawling 3rd party marketplaces	Last developments
	Direct use of offering via the marketplace (shopping cart)	In development
	UI improvement	Under revision
	User management	In development
	Offering's infrastructure requirements	Concept phase
	Marketplace statistics	Ready for owner, concept phase for users and offering owners
	Onboarding	Shaped, but still in progress
Review committee	UX improvement	Under discussion, some work in development
DIH	API for platform integration	Under review
	Search engine	Last developments
	Pricing schemes	In development
	Success stories	Work in progress (UX, search functionality, user management)
WP1 partners: T1.4 on stakeholder's entry points to Marketplace	Different search criteria for each stakeholder	Under revision
	Categorize offerings per region	
	Users recommendation system	
	Solution provider to update offering description	
	Solution provider to monitor effectiveness	