

OPEN_NEXT

Deliverable 3.2

Platform demonstrator for collaborative engineering



This project is funded by the European Union’s Horizon 2020 Research and Innovation Programme under the Grand Agreement no. 869984.



OPEN_NEXT – Transforming collaborative product creation

Consortium:

#	Participant Legal Name	Short Name	Country
1	TECHNISCHE UNIVERSITAT BERLIN	TUB	DE
2	INSTITUT POLYTECHNIQUE DE GRENOBLE	GINP	FR
3	ALEXANDER VON HUMBOLDT-INSTITUT FUR INTERNET UND GESELLSCHAFT GGMBH	HIIG	DE
4	UNIVERSITY OF BATH	UBA	UK
5	ZENTRUM FUR SOZIALE INNOVATION GMBH	ZSI	AT
6	FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	FHG	DE
7	DANSK DESIGN CENTER APS	DDC	DK
8	WIKIMEDIA DEUTSCHLAND - GESELLSCHAFT ZUR FÖRDERUNG FREIEN WISSENS EV	WMDE	DE
9	WIKIFACTORY EUROPE SL	WIF	ES
10	STICHTING WAAG SOCIETY	WAAG	NL
11	MAKER	MAK	DK
12	AGILE HEAP EV	FLB	DE
13	SONO MOTORS GMBH	SOM	DE
14	OPNTEC GMBH	OPT	DE
15	STYKKA APS	STY	DK
16	TILL WOLFER	XYZC	DE
17	FICTION FACTORY	FIF	NL
18	M2M4ALL	SOD	NL
19	INNOC OSTERREICHISCHE GESELLSCHAFT FUR INNOVATIVE COMPUTERWISSENSCHAFTEN	HAL	AT

Duration: 09/2019-08/2022

Grant: H2020-869984

Contact (coordinator): Prof Dr-Ing Roland JOCHEM

Address: Technische Universität Berlin, Sekretariat PTZ 3, Pascalstr. 8-9, 10587 Berlin

E-mail: roland.jochem@tu-berlin.de

Disclaimer: The contents of this document are not intended to replace consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at his/her sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the views of the author(s).

D3.2 – “Platform demonstrator for collaborative engineering”

Review and approval status:

	Name and Surname	Role in the Project	Partner
Author(s)	Sonika Gogineni, Martin Häuer	Work Package lead, Research Assistant	FHG
	Erik Paul Konietzko, Cansu Tanrikulu	Work package development, Student Assistant	FHG
	Max Kampik, Diego Vaquero, Andrés Barreiro	Work package development	WIF
Reviewed by	Mehera Hassan	Research Associate	TUB
	Pen-Yuan Hsing	Postdoctoral Researcher	UBA
Approved by	Robert Mies	Project coordinator	TUB

History of changes:

Version	Date	Description of changes	By
0.1	21.06.2021	Initial draft	Sonika Gogineni
0.2	14.07.2021	Second draft version	Martin Häuer, Diego Vaquero
0.3	30.07.2021	Draft for review	Sonika Gogineni
1.0	26.08.2021	Final document for final project management verification	Sonika Gogineni, Cansu Tanrikulu

Details:

Dissemination level	Open Access
Due date	31.08.2021
Issue date	31.08.2021
Contract No.	869984
Responsible Partner	FHG
File name	D3.2_ Platform demonstrator for collaborative engineering.docx

Keywords:

Open source hardware, ICT for OSH, open source development, community management, skill ontology, interoperability, collaborative production, guidelines, documentation, match making

Executive Summary:

This document aims at presenting the different demonstrators and software tools developed to support company-community collaboration (C3) in Open Source Hardware (OSH) projects. The developments have been carried out by Fraunhofer IPK and Wikifactory. Based on the prioritized needs of the target group a set of concepts (referred to as solutions in the deliverable) were developed to address the needs. This led to the development of solutions under four main categories namely: community management, interoperability, collaborative production, and documentation and guidelines. The target groups identified for the solutions are companies, communities and platform owners/ creators.

Under community management, the focus was on matchmaking the company and community with skills and interests as the common connector. A first version of the solution as a demonstrator integrated in Wikifactory is presented in this deliverable. The aim of this demonstrator is to facilitate companies or communities to find and collaborate with suitable community members.

Under the category interoperability a stable version of an import-export tool has been developed to facilitate transfer of files from one platform to another. Hence, providing the community and company to move projects across platforms without facing a vendor lock-in.

Under the category of collaborative production, in order to assist a company/community member looking for producers/prototype manufacturers a first prototype of a matchmaking tool was developed on Wikifactory. The first step was to identify production metadata required to ensure a seamless communication, followed by automatically identifying metadata in project text and then finding suitable collaborations.

Under the category documentation and guidelines, four different templates and guides were developed to assist companies and contributors to work on OSH projects.

For each of the above-mentioned solutions, the problem it addresses, its aim, its contents, main features, usage and a step-by-step example are discussed. The details about how the solutions were selected and developed are mentioned in a separate annexure document along with the installation guide to use the solutions. All the results here represent a first release, which need to be evaluated and further developed during the course of the project.

The main target groups of the deliverable are companies (SMEs) and OSH communities. The secondary target groups are the platform owners, who can further integrate the open source solutions in their platforms. The companies and OSH communities, are provided with these new solutions to

- Develop, document and manage their OSH projects (by referring to and using the documentation and guidelines, and using the import-export tool). SMEs will have an easy way of migrating their files to other platforms. This prevents vendor-lock in and allows testing how a certain project might take advantage of the alternative workflows of a different platform with minimal friction.
- Develop growing communities to work on their projects, while enabling discovery of projects and people related to the interests of the user (by using the skill-based matchmaking demonstrator)
- Find relevant partners in order to develop prototypes or manufacture products world-wide, enabling local manufacturers while increasing the accessibility of worldwide specialists in key technologies (Collaborative production solutions)

The impact expected from these solutions and their benefits for the target group can range from individual, organisational to social impacts. The companies and communities are supported with open source hardware projects, through guides, templates and ICT (Information and communication technology) features to work with. This is illustrated with an example, on the Wikifactory platform. The features are expected to be used by its large community of companies and communities, to work on projects, create communities and collaborate on manufacturing across the world. In addition, the solutions could be used individually or by other platforms, hence expanding the impact.

Table of contents

1. Introduction	9
1.1. The prioritized needs	9
1.2. The way forward with solutions	13
1.3. What can the solutions do?	14
1.4. Where to find the repositories for the solutions?	15
1.5. How to read this document?	16
2. Import-Export Tool	19
2.1. Usage.....	20
2.1.1. Starting the Import Export service	21
2.2. Step-by-step example	22
2.2.1. REST API	23
2.2.2. Wikifactory Integration.....	25
2.3. Summary & Outlook.....	28
3. Skill based matchmaking	30
3.1. Usage.....	31
3.2. Step-by-step example for contributor and project core team	33
3.3. Step-by-step example for platform owner	38
3.4. Summary and outlook.....	44
4. Collaborative Production.....	46
4.1. Identifying production metadata.....	46
4.1.1. Usage	48
4.1.2. Step-by-step example	50
4.1.3. Summary & Outlook	51
4.2. Identifying production metadata in documentation through machine learning	52
4.2.1. Step by step example.....	52
4.2.2. Summary & Outlook	57
4.3. Collaborative production and matchmaking tools on Wikifactory.....	58
4.3.1. Usage	58
4.3.2. Step-by-step example	59
4.3.3. Summary and outlook	61
5. Documentation and Guidelines	62
5.1. Project/Product overview template	62
5.2. Technology-readiness levels for OSH.....	63
5.2.1. Usage	65
5.2.2. Step-by-step example	66
5.2.3. Summary & Outlook	67
5.3. Licensing open source hardware guide (TL;DR).....	69
5.3.1. Usage	70
5.3.2. Summary & Outlook	70
5.4. Integrating templates in Wikifactory	71
5.4.1. Usage	71
5.4.2. Step-by-step example	71

6. Summary and Outlook 73
7. Licenses 75
8. Attributions 76
9. References..... 78

List of abbreviations and terms

API	Application programming interface
BOM	Bill of materials
C3	Company-community collaboration
CAD	Computer-aided design
ESCO	European Skills/Competences, qualifications and Occupations
JSON	JavaScript object notation
HTML	Hypertext markup language
ICT	Information and communication technology
IRI	Internationalized resource identifier
NT	N-Triples
ODRL	Open Documentation Readiness Levels
OTRL	Open Technology Readiness Levels
OSH	Open source hardware
OSD	Open source development
OWL	Ontology web language
PLC	Product lifecycle
PMI	Product and manufacturing information
RDF	Resource Description Framework
REST	Representational state transfer
SME	Small and medium-sized enterprises
SPARQL	SPARQL protocol and RDF query language
URL	Universal resource locator
TOML	Tom's Obvious, Minimal Language
TTL	Terse RDF Triple Language

List of figures

Figure 1: Definitions with their relationships..... 8
 Figure 2: User story group statistics (blue>75%, yellow: 50-75%) 9
 Figure 3: User stories relevant for deriving community management solutions 10
 Figure 4: User stories and identified actions relevant for maturity assessment of OSH projects 11
 Figure 5: User stories and identified actions for Guidelines and documentation 11
 Figure 6: User stories and identified actions related to legal issues when dealing with OSH 12
 Figure 7: User stories and identified actions relevant for deriving collaborative production 12
 Figure 8: User stories and identified actions relevant for interoperability..... 13
 Figure 9: Solutions to the needs defined in the user stories 13
 Figure 10: Aims of the solutions derived..... 14
 Figure 11: Overview of the solutions structure..... 17
 Figure 12: Schematic view of the Import Export Service 19
 Figure 13: Cloning the Import Export repository 21
 Figure 14: Building the Import Export service..... 21
 Figure 15: Starting the Import Export Service..... 22
 Figure 16: Screenshot of the HTTP POST request for starting an import-export job..... 23



Figure 17: JSON response for a newly created import-export job	24
Figure 18: REST GET response when asking for a certain job	25
Figure 19: Creating a new project in Wikifactory	26
Figure 20: Importing a Google Drive folder from Wikifactory webpage	26
Figure 21: Waiting for the user authorization when importing from Google Drive	27
Figure 22: Oauth authentication popup	27
Figure 23: Inspecting the imported files in Wikifactory	28
Figure 24: Web interface for the Import Export service	28
Figure 25: Skill based matchmaking concept	30
Figure 26: Mapping process	32
Figure 27: Method to create a query string for user interest	34
Figure 28: Query prefixes	34
Figure 29: Setup for query execution using QueryExec.java	34
Figure 30: Execution of query using the query generation method	34
Figure 31: Load ontology from URI via Protégé	35
Figure 32: SPARQL plugin tab in Protégé	35
Figure 33: US2 Query with prefixes	35
Figure 34: Query results US3	35
Figure 35: Query results US2	36
Figure 36: Project suggestions to a user of WIF platform	36
Figure 37: User profile skill enrichment layout on the WIF platform	37
Figure 38: WIF user profile with skills tags	37
Figure 39: Adding skill tags to a project on WIF platform	38
Figure 40: Featured project section on WIF platform	38
Figure 41: Repository namespaces in the ontology code	39
Figure 42: Declaration of annotation property in an owl file	40
Figure 43: Mapping annotation for Class "Project" with pointer string to json input file	40
Figure 44: skillIRI and mappingIRI variables in CreateInstances.java	41
Figure 45: instanceIRI variable in CreateInstances.java	41
Figure 46: directory variable in CreateInstances.java	41
Figure 47: Setting up JSONReader objects for the JSON input in CreateInstances.java	41
Figure 48: Instantiation of mapping annotations in CreateInstances.java	42
Figure 49: Lists to save the annotation properties in CreateInstances.java	42
Figure 50: Set NT output in CreateInstance.java	42
Figure 51: Setting ontology saving location in CreateInstances.java	42
Figure 52: Running the CreateInstances.java class	43
Figure 53: Saving statement for the ontology on the console	43
Figure 54: Table 3 query results	43
Figure 55: Production metadata per part allowing for a fast assessment of manufacturability and matchmaking with external production partners	47
Figure 56: Provision of production metadata by publishing a plain text file („manifest file “) in a git repository	49
Figure 57: A picture of the clamp ring	50
Figure 58: Resulting production metadata for Clamp Ring	51
Figure 59: Front end of the production metadata identification application	53
Figure 60: Copying entries from Wikifactory	53
Figure 61: Pasting the entry as text in the application	54
Figure 62: Results of text entry	54

Figure 63: Copying the URL from an Appropedia website 55
 Figure 64: Pasting the URL and running the model..... 56
 Figure 65: Results of the URL 57
 Figure 66: An example of the CAD Rooms tool 59
 Figure 67: Screenshot of the form used to start the manufacturing process..... 60
 Figure 68 Inspecting the manufacturers that could make the selected part..... 61
 Figure 69: Snippet of project overview template 63
 Figure 70: Providing users with a fast assessment of maturity of technology & documentation for a certain OSH product 64
 Figure 71: View on an assembled OHLOOM v1.0.0 in use 67
 Figure 72: Low-threshold documents (e.g. in form of a tl;dr) may provide the missing link to connect project team members with essential (legal) information 69
 Figure 73: Documentation categories 71
 Figure 74 Screenshot of the documentation and guidelines imported in a Wikifactory project 72
 Figure 75: Solutions and their benefits across project phases..... 73

List of definitions

- Company:** An organisation or entity
- Community:** A community is a group of people or organisations. They can take up different roles in projects. Some of them works on projects online using online platforms to carry out certain tasks (Dai et al. 2020).
- Community member:** Community member is described as anyone who is interested in projects or other users for example on an online platform. They can duplicate the project without any contribution for their own purpose (Li und Seering 2019)
- Project team:** An open source project team consists of the core team and contributors (Dai et al. 2020)
- Core team:** Core team usually builds and manages the backbone of a project; they usually are responsible for major decisions. They are a part of the project team.
- Contributor:** From community members, the ones who are interested and contributing in a project, called contributors (Dai et al. 2020)

The above definitions are further illustrated in Figure 1.

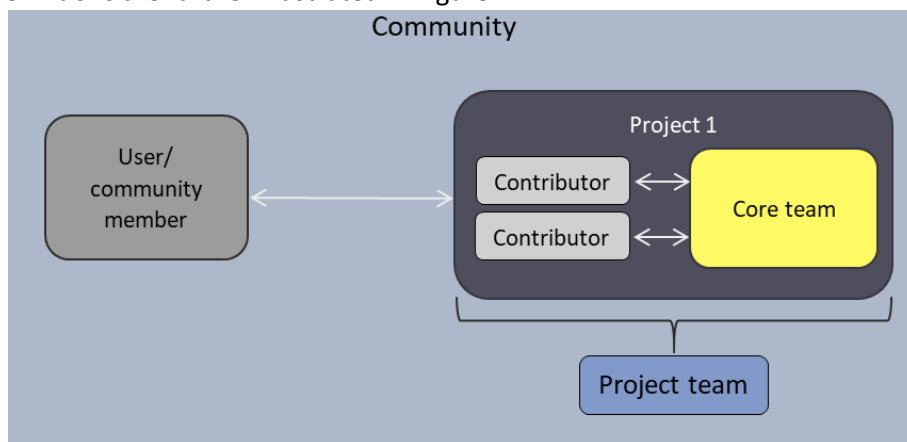


Figure 1: Definitions with their relationships

1. Introduction

The main objective of the deliverable is to identify and develop ICT solutions for OSH development and (Company-Community Collaboration) C3 needs. The needs referred to here were derived in the previous deliverable D3.1¹ where 80 needs were found in the form of user stories. These were spread across 20 categories. As a next step, these needs were prioritized: based on the number of times it was mentioned and also considering the results from WP2 Interviews². For detailed information about the methodology used to prioritize please refer to the Annexure (Section 1.1).

1.1. The prioritized needs

For all the user stories, general actions on how to address the needs were identified and transferred to an issue board on GitHub³. Figure 2 shows the statistics of the user story categories, based on their mentions and the numbers of user stories that they summarise.

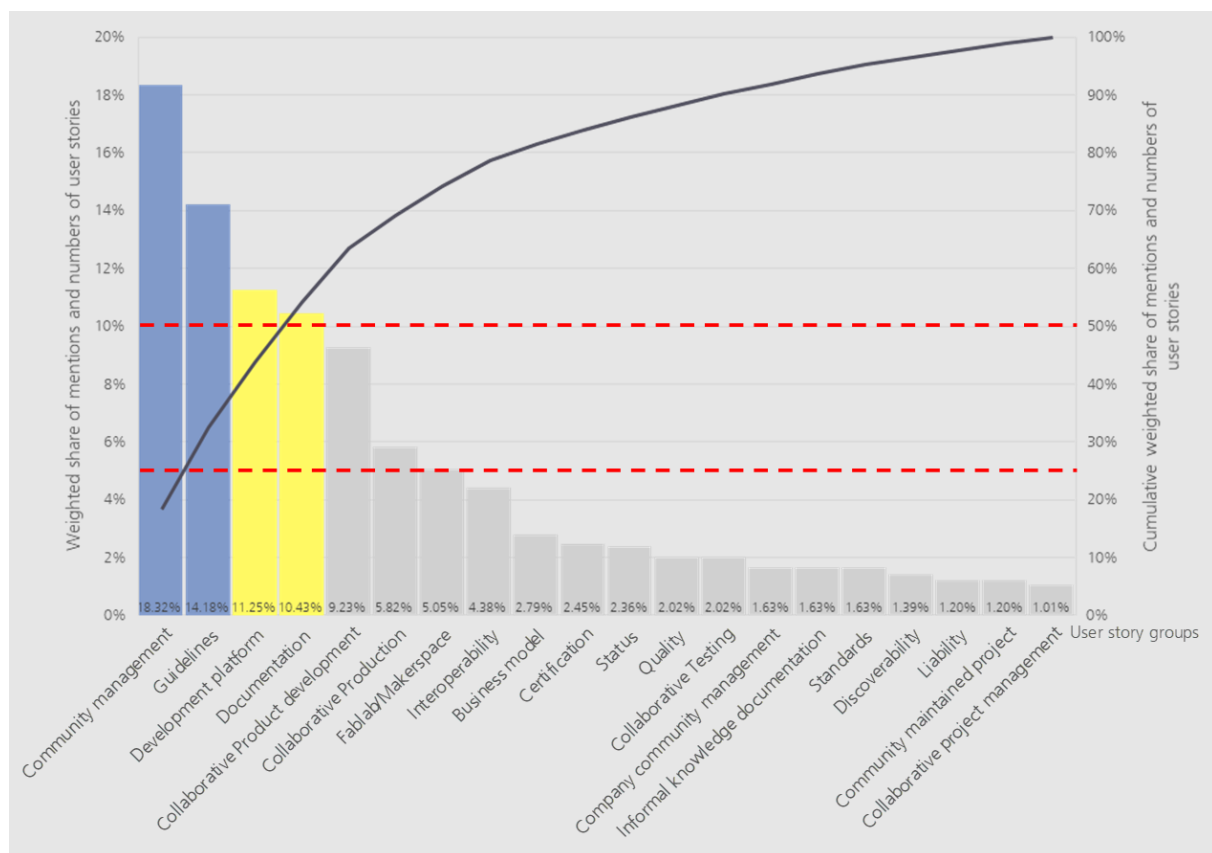


Figure 2: User story group statistics (blue>75%, yellow: 50-75%)

In addition to the prioritisation of user story categories, the identified actions were further aligned with the project partner Wikifactory (WIF) for feasibility and usefulness based on the structure and functions of the WIF platform. Based on this the four final prioritized categories were community

¹ <https://depositonce.tu-berlin.de/handle/11303/10962>

² <https://www.sciencedirect.com/science/article/pii/S2212827121005047>

³ https://github.com/OPEN-NEXT/wp3_pub/projects/2

management, documentation & guidelines (2 categories were combined), collaborative production and interoperability.

In these categories, the following user stories and their respective actions were derived for further development:

For the **community management** the highest rated user stories *U1/2: Finding and attracting the right collaborators* and *U1/3: Motivating community to contribute* were chosen for the development which influences collaborative product development (*U13/x*), the basis of OSH development. Figure 3 additionally shows the aligning of the user stories with corresponding activities identified in a user journey on the WIF platform. For example, to find, attract and motivate right contributors it is important to work on project information (on WIF) so that it is easy to find and in turn it is also important to have complete profile information of the contributor (on WIF) to find them easily. Adding features such as user and project skill tagging can help in finding and matchmaking among communities.

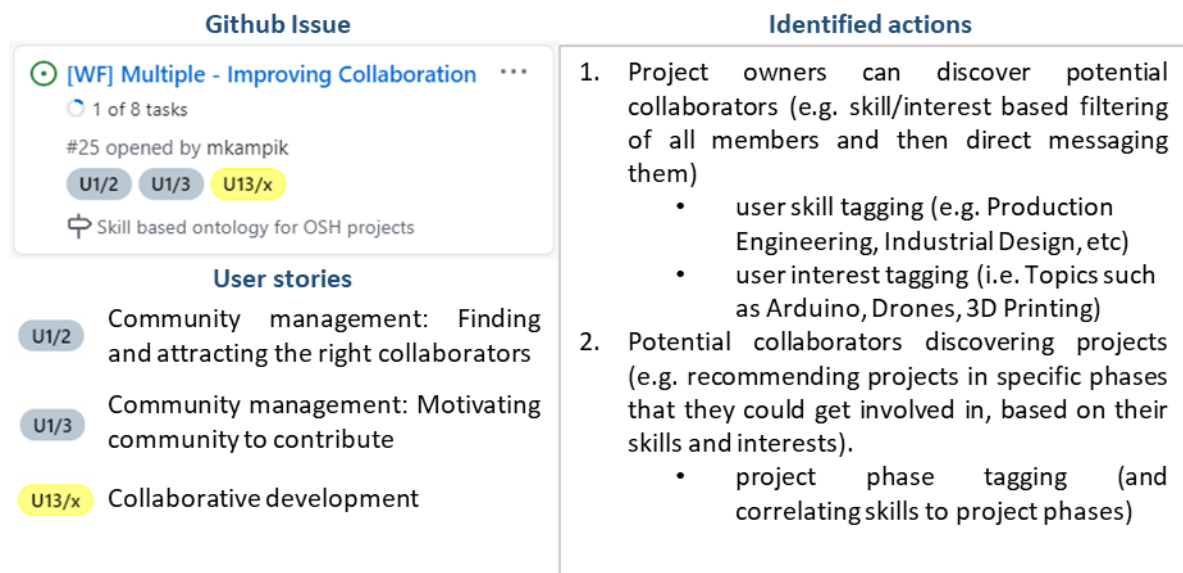


Figure 3: User stories relevant for deriving community management solutions

An essential filter criterion for users and potential collaborators is the maturity of an OSH project, specifically regarding the technology and the technical documentation. This is represented by the user story *U7/1: Identify status of project*, where it is essential for projects to indicate their current status in project development and in terms of documentation (refer to Figure 4). This in turn, helps motivated collaborators (*U1/3*) to find interesting projects to work on. In addition, representing the status also indicates the quality and completeness which can be expected from the documentation (*U8/1*). Hence, developing a set of maturity levels to indicate status of project and documentation is an identified action. In addition, for a qualitative and complete documentation, identifying basic necessary metadata for projects is also beneficial.

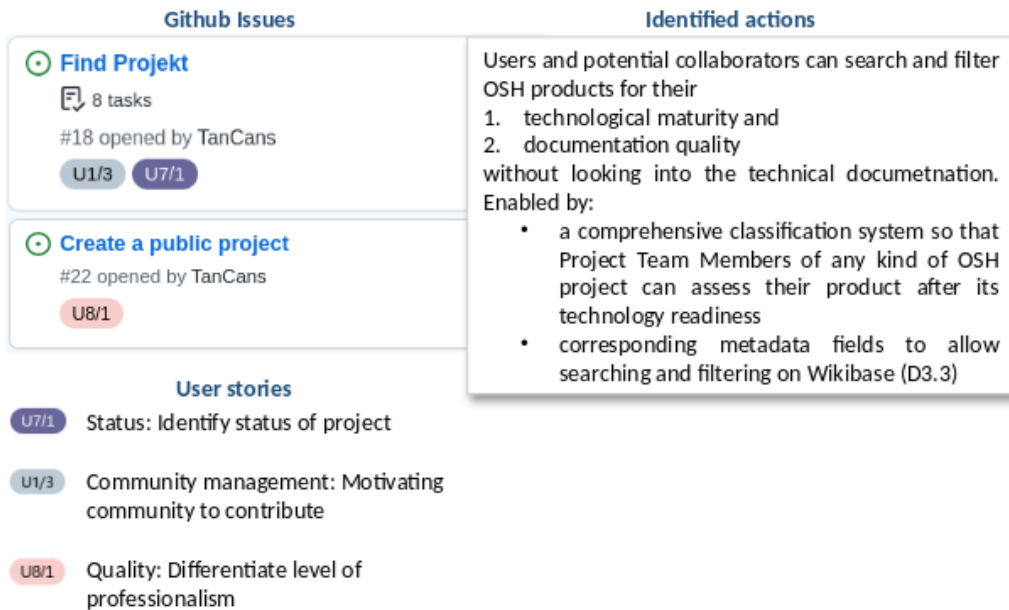


Figure 4: User stories and identified actions relevant for maturity assessment of OSH projects

The **documentation and guidelines** user stories include all the User stories under U12x: *Guidelines*, U3/x: *Documentation: Motivation for documentation* and U3/7: *Documentation: Up to date documentation*. This section, does not directly deal with software tools, but templates and guidelines to help the companies with documentation. Figure 5 shows the various topics for which a guide is desired from the community. In addition, to motivate documentation, actions such as displaying percentage of completeness and gamification were also identified as a possible solution.

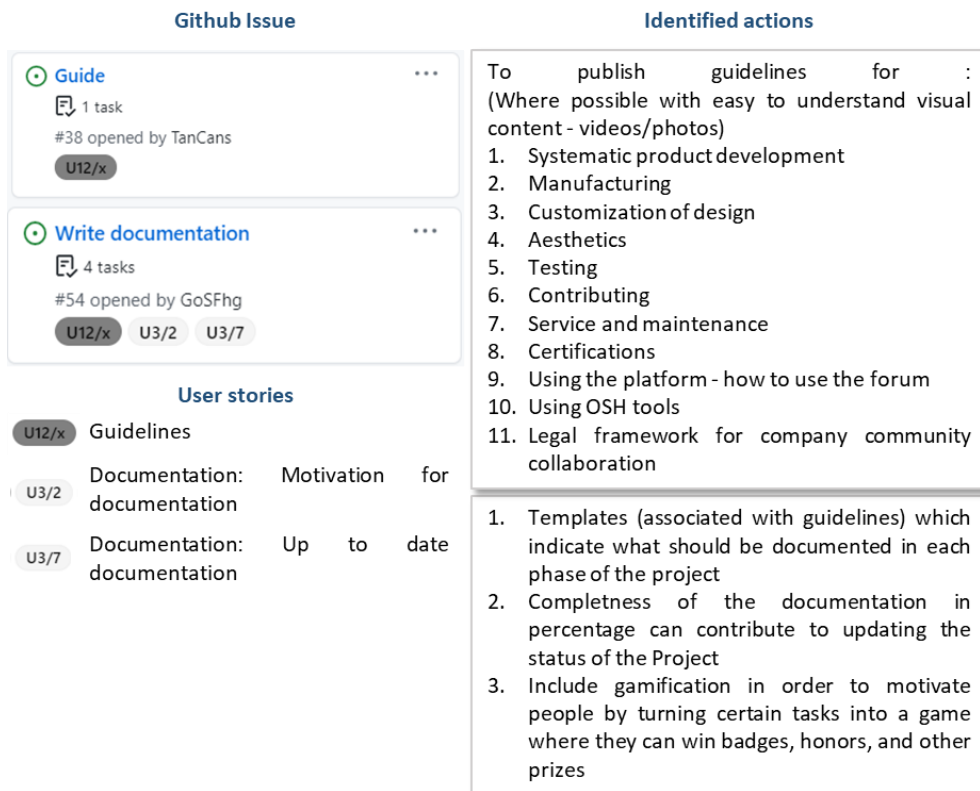


Figure 5: User stories and identified actions for Guidelines and documentation

As OSH is relatively new, the licensing legalities are not yet very clear and well known for companies and communities. This is represented by the user story *U12/11* as shown in Figure 6. In close relation to the same, user story *U14/5: Fair acknowledgment of contributor* is also an important aspect for collaborations. The identified actions include researching, gathering and developing a guide for companies and OSH communities to understand licenses and their use.

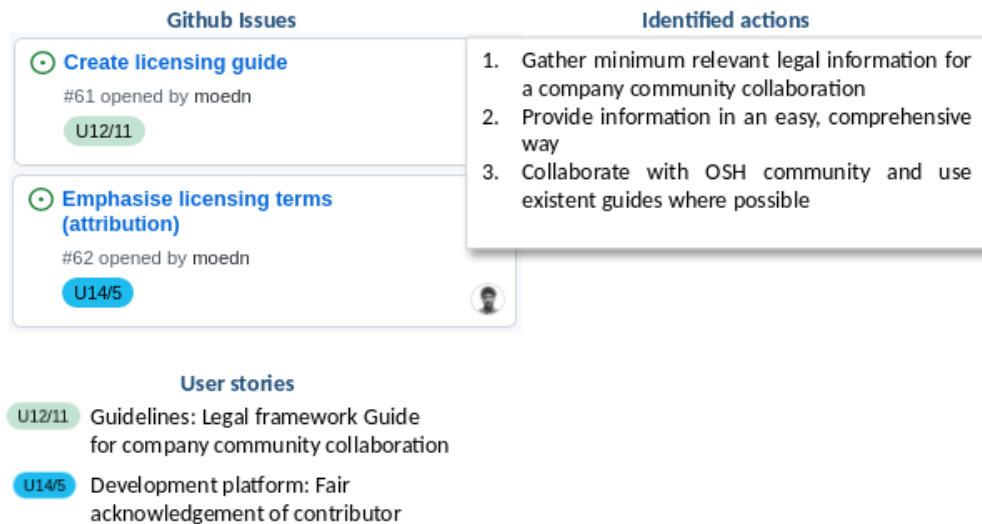


Figure 6: User stories and identified actions related to legal issues when dealing with OSH

For **collaborative production** the user stories stated in Figure 7 were grouped to be addressed with solutions. As WIF already supports forking and documentation of variations of products and to create linked BOMs the third user story *U6/3* was focused upon. To find right partners for manufacturing, prototyping and distributing through online platforms worldwide was a need for which a solution needed to be developed. The identified actions included developing manufacturer listings, hosting and

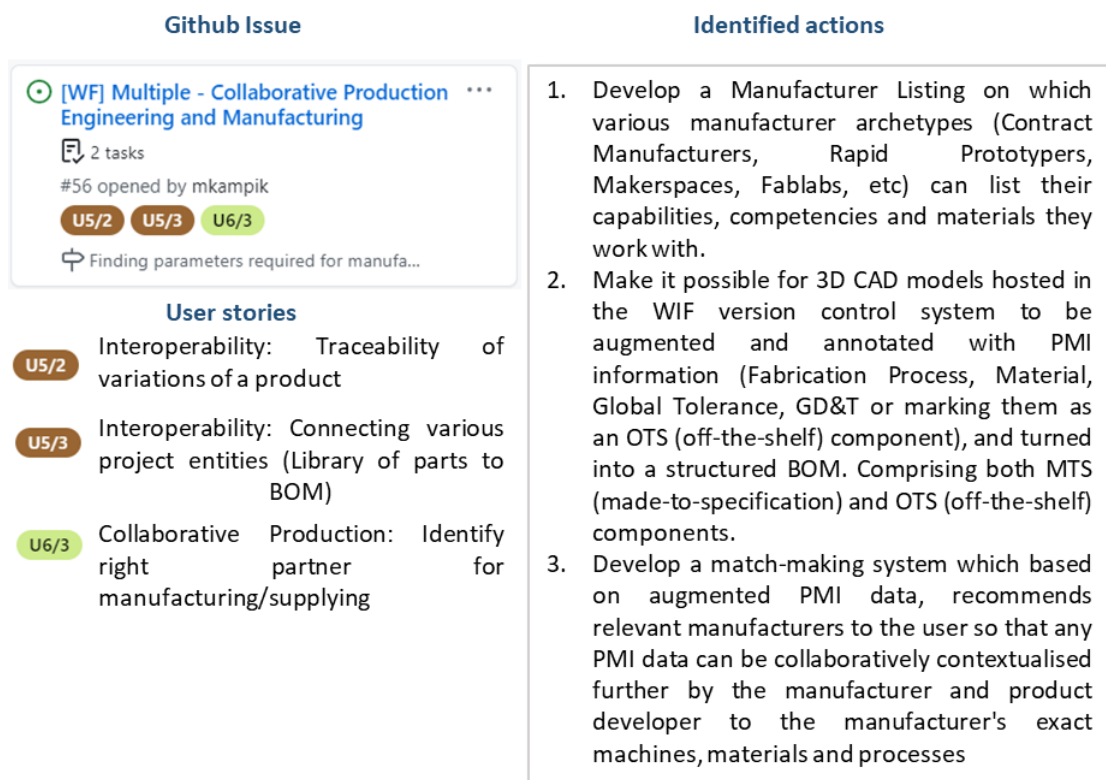


Figure 7: User stories and identified actions relevant for deriving collaborative production

communicating with 3D models and related product information. In addition, enabling matchmaking based on Product and manufacturing information (PMI) and manufacturer capabilities (from manufacturing listings) was also identified.

With increasing number of open source platforms, and individualized project websites and forums, it is important to provide an option of **interoperability**. This provides the community with the flexibility of moving projects and their data from one platform to another. This was captured in the user story *US5/1: Interoperability of IT tools*.

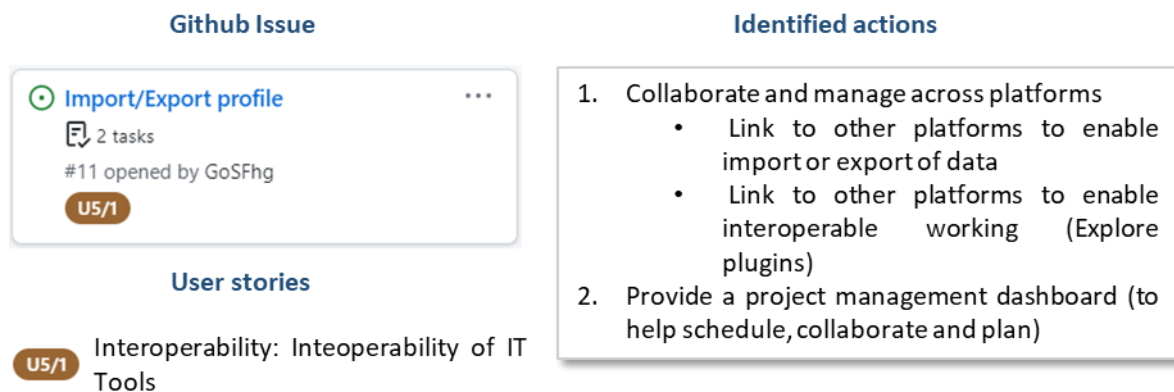


Figure 8: User stories and identified actions relevant for interoperability

1.2. The way forward with solutions

After selecting the categories based on user stories and prioritisation, the actions were further detailed to form ICT solutions. The solutions associated with the categories can be seen in Figure 9, these were worked on and developed.

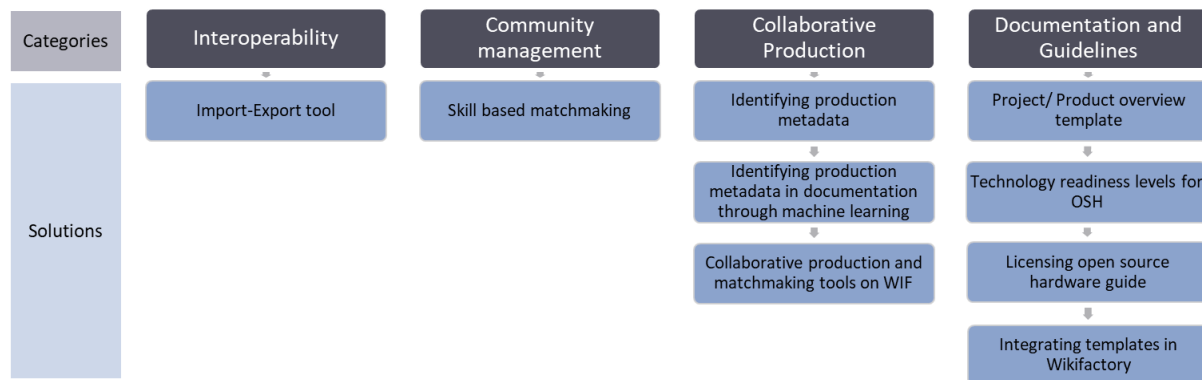


Figure 9: Solutions to the needs defined in the user stories

Under the category **Interoperability** is the **Import-Export tool**. The tool is an open source solution to enable companies and communities to easily transfer their project repositories from one platform to another.

Under the category **Community management**, **skill based matchmaking** solution is under development. The solution aims at developing a central ontology of skills which can be used to tag user profiles and issues to enable a semantic based skill matchmaking.

Under the category **Collaborative production**, a multiple step approach was followed. The first step was to **identify production metadata**. This step resulted in finding the main metadata necessary to produce parts. The second step, was to **identify these metadata in OSH documentation through machine learning**. The reason for selecting the machine learning approach is because the information about the metadata can be found anywhere in the documentation. The final step to support collaborative production, is to enable **matchmaking** with producers based on the production metadata. This includes a communication opportunity using annotations on a 3D model and selecting and communicating directly with producers online.

Under the last but not the least category of **Documentation and Guidelines**, four main focus areas were worked upon. The first one, was to generate a **template for documenting a project/ product overview**. The second focus area, was to develop **technology readiness levels for OSH** to enable the community to understand the readiness of the projection. The third focus area, was the **license documentation guideline**, which provides an insight into the most asked question: how to go about with OSH licensing. The last developments, is a guide to **Integrating templates in WIF**, which can be used and edited by the core project team to help guide them through documentation.

1.3. What can the solutions do?

The solutions are developed for providing assistance to the target groups mainly concentrating on companies, communities and the collaboration between them. In addition, some of the solutions can also be adapted by platform owners for their own OSH platforms. Figure 10, summarizes the benefits of the solutions in grey.

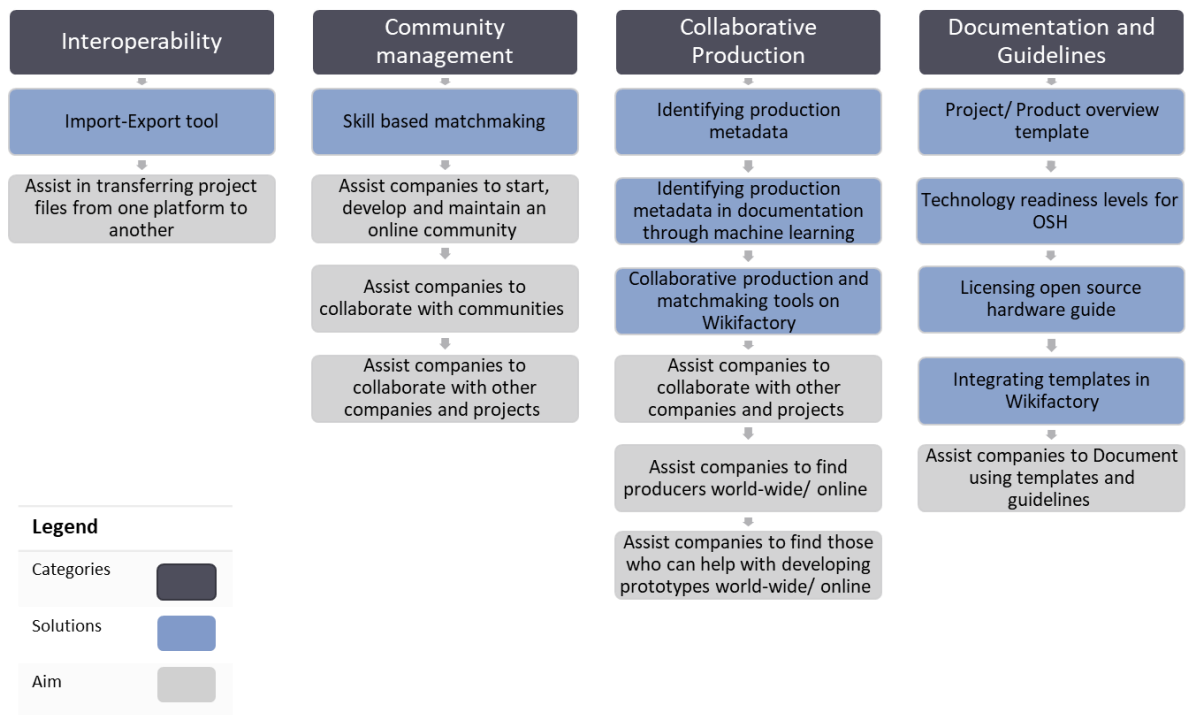


Figure 10: Aims of the solutions derived

The import-export tool, assists companies and communities to transfer project files to a platform of their choice, without having to have issues such as vendor-lock in. This also benefits companies and community to use new platforms and their features. The skill based matchmaking tool can assist in enabling a mutual collaboration between the company and community. The company can develop the project and assign skills to attract the online community, this in turn enables collaboration with

interested contributors. Not just with contributors, but also other companies, which would like to collaborate on certain issues. For companies looking to find communities or other companies to manufacture their products world-wide, collaborative production tools of matchmaking based on project metadata can be beneficial. In addition, the companies can also find makerspaces or labs interested in prototyping together through matchmaking. Templates and guidelines assist companies and communities to document their projects. In addition, also provide assistance for contributors to collaborate on projects and their issues.

1.4. Where to find the repositories for the solutions?

Each of the mentioned solutions developed can be found online in their respective GitHub repositories or on the Wikifactory platform. The links to the repositories and the platform sources are listed in Table 1.

Table 1: List of repository/documentation links

Category	Solution name	Repository link
Interoperability	Import-Export tool	https://github.com/OPEN-NEXT/import-export
		https://help.wikifactory.com/en/articles/5152398-how-to-import-your-files-from-github-gitlab-or-dropbox
Community Management	Skill based matchmaking	https://github.com/OPEN-NEXT/WP3_Skillmatching
Collaborative Production	Identifying production metadata	https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OKH-LOSH.md#production-metadata
	Identifying production metadata in documentation through machine learning	https://github.com/OPEN-NEXT/Named-Entity-Recognition-for-extracting-Open-Source-Hardware-project-metadata
	Collaborative production and matchmaking tools on Wikifactory	https://wikifactory.com/+wikifactory/beta-testing-new-cad-tool
Documentation and Guidelines	Project/ Product overview template	https://github.com/OPEN-NEXT/WP3-Documentation-Guidelines-for-OSH-Projects/tree/main/Documentation%20%26%20Guidelines
	Technology readiness levels for OSH	https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OTRL.md#concept
	Licensing OSH guide	https://github.com/OPEN-NEXT/tldr-ipr/
	Integrating templates in WIF	https://github.com/OPEN-NEXT/WP3-Documentation-Guidelines-for-OSH-Projects/tree/main/Documentation%20%26%20Guidelines

1.5. How to read this document?

Each of the solutions are placed under four main sections as shown in Figure 11 below. The solutions are structure to have the following format:

- Introduction: In the beginning of each solution, the Problem, aim, target groups and main features are detailed. As some of the deliverables are software intensive, basic pre-requisite knowledge required to understand the following are also mentioned.
- Usage: How the target group can utilize the solution is defined.
- Step-by-step example: How the solution can be used with an example is detailed. For some solutions, separate examples are provided based on the target group.
- Summary and Outlook: conclusion about the discussed solutions and the next steps for the same.

For easy navigation through the document, the Figure 11 below is linked to the respective sections. Based on the topic of interest just click (control key + left click) on the respective box. This will jump directly to the solution or the individual section. To jump back to this overview, click on “Back to Overview” located under every page on the left corner.

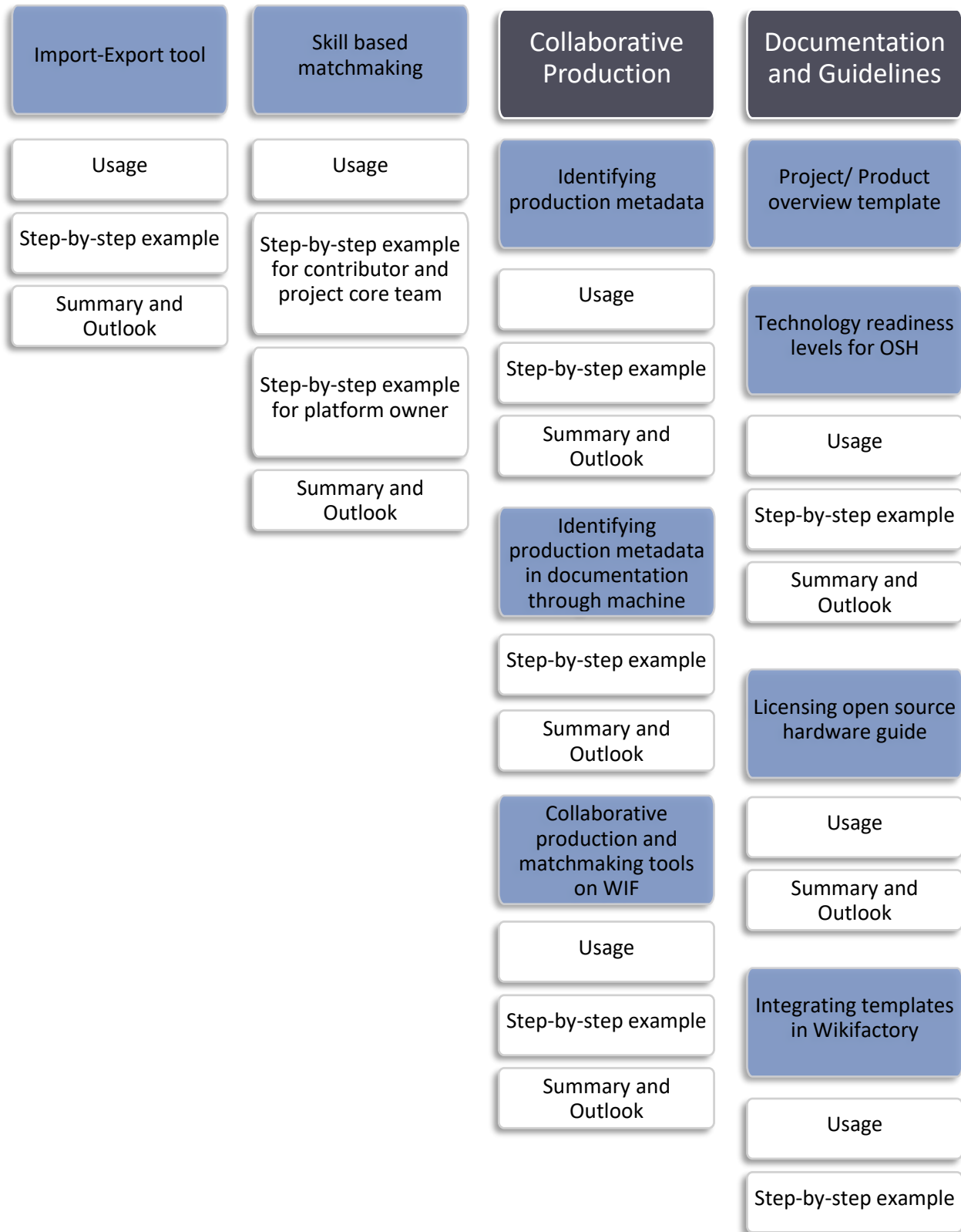


Figure 11: Overview of the solutions structure

This document is supported with an annexure document, which is structured as defined below:

- Development method: the methodological process followed to arrive at the solutions from the identified actions is described here.
- Design notes: to understand the solution and its design for future developments, flow charts or similar support material is presented.
- Installation guide: some of the solutions need to be installed before use, hence a detailed guide is included in the annexure.

In this document, the annexure is referred to with a suitable section number. This is to guide the readers to find detailed information regarding the solutions.

2. Import-Export Tool

Currently, a number of collaborative platforms exist that allow their users to store, manage and share their data in multiple ways, each with their own set of advantages and drawbacks. While collaborating within the boundaries of one of these platforms is usually a smooth experience, it can be difficult to move elsewhere or spin off a related development based on the same set of files.

The aim of the Import-Export tool is preventing platform lock-in and ease up collaboration between platforms. As a first step towards achieving that goal, this tool implements a backend service written in Python that allows importing projects from a certain source platform (*import service*) and exporting them to a target one (*export service*) in an automated way. Platform providers themselves or third parties can deploy this service to their own servers, which can be then interfaced through a REST API by other backend services.

Using the API, a new import-export *job* can be created. Each job is a one-off process that specifies a source and a target project through their URLs. The tool will process all that information, check if the projects are hosted in any of the supported platforms and then queue a task which will copy the contents of the project from one place to the other (first downloading them from the source so they can be uploaded to the target later), requesting authentication information or returning errors if any requirement wasn't fulfilled, allowing the process to be retried or cancelled in those scenarios. Once the import-export *job* has finished successfully, the user will be able to see a copy of the original files on the *export service*. All these different statuses are stored and logged in a relational database, referencing the relevant *job*.

At this point of the implementation, four platforms are supported by the tool for both importing and exporting: i) *Wikifactory*, ii) *GitHub/Gitlab (and in general git-based platforms)*, iii) *Dropbox* and iv) *Google Drive*. However, the tool is flexible enough for allowing the easy addition of more services in the infrastructure. Figure 12 shows a schematic view of the process, with the *import services* located on the left side and the *export services* located on the right side. Note that there is an icon in the *import services* column belonging to the *Instructables* platform⁴ and another one in the *export services* column for the *MyMinifactory* platform⁵. This serves to illustrate the new kind of services that could be integrated in the Import-Export tool.

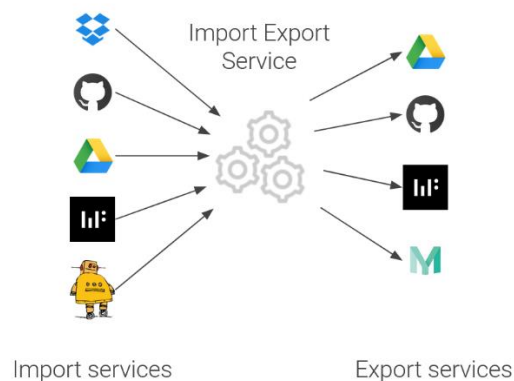


Figure 12: Schematic view of the Import Export Service

⁴ <https://www.instructables.com>

⁵ <https://www.myminifactory.com/>

In summary, the main features of the Import-Export tool can be summarized as:

- Fully independent web *service* ready to be deployed and used.
- Automatic handling of the *importing* (i.e., downloading) and *exporting* (i.e., uploading) processes.
- High extensibility since new services can be added in the future.
- From the point of view of the final user, the abstraction of credentials handling for accessing the files.
- Highly configurable, since the project owner could decide the sets of importer and exporter services available to be used.

2.1. Usage

This section is intended as a guide for a platform developer who wants to integrate the Import-Export service functionalities into their own platform. It also serves as a guide for companies/ community members who want use the tool on their own device for moving their files between different platforms. For those looking forward to move files into Wikifactory, a step by step example for the same is detailed in [Subsection 2.2.1](#).

As mentioned in the introduction, the Import-Export tool is a backend service (implemented in Python using [FastAPI](#)⁶'s framework) that requires a [PostgreSQL](#)⁷ relational database (for storing job status) and runs background tasks (using a [Celery](#)⁸ task runner with a [Redis](#)⁹ broker). In order to abstract out all these tech requirements, the application has been containerized using [Docker](#)¹⁰ and the different services used expressed in a [Docker Compose](#)¹¹ configuration file. What this means for the purposes of the integration is that just by having docker and docker - compose tools installed plus an internet connection, a basic service plus all its dependencies can be installed and then executed just by running a single command: `docker - compose up`.

While the usage of Docker and Docker Compose out abstracts away the details required to set up and run the required tech stack (and additional details have been moved to the Annex), knowledge about the aforementioned technologies (Python, FastAPI, PostgreSQL, Celery, Redis, Docker and Docker Compose) is required to properly understand usage and implementation of the Import-Export tool.

The backend service exposes a REST API, which means the actions supported (job creation, retry or cancellation, checking job status, etc.) are triggered by sending HTTP requests. The full set of requests supported and its parameters are listed and accessible through a documentation endpoint which can be accessed through the browser via `http://localhost:8000/redoc` once the service has been started.

⁶ <https://www.myminifactory.com/>

⁷ <https://www.postgresql.org/>

⁸ <https://docs.celeryproject.org/en/stable/>

⁹ <https://redis.io/>

¹⁰ <https://www.docker.com/resources/what-container>

¹¹ <https://docs.docker.com/compose/>

Since interactions between this service and other backend services happen through a REST API, we call this service a microservice. This is the term used for small applications with a single responsibility that can be deployed, scaled, and tested independently in opposition to a monolithic system. Because of this, over the following section (*the*) *microservice* might be used to refer to (*the*) *Import-Export Backend service* or *Import-Export tool*.

2.1.1. Starting the Import Export service

The first step for using the Import Export tool consist of cloning (using [Git](#)¹²) the tool’s repository in the machine that will provide the service. The repository is available on <https://github.com/wikifactory/import-export>¹³ or <https://github.com/OPEN-NEXT/import-export>¹⁴. This step can be done from the command line (using the “*git clone*” command) or using a graphic tool. Figure 13 shows the command required to clone the repository:

```
rievo@rievo-linux:~/development/opennext$ git clone https://github.com/wikifactory/import-export
Clonando en 'import-export'...
remote: Enumerating objects: 3429, done.
remote: Counting objects: 100% (819/819), done.
remote: Compressing objects: 100% (623/623), done.
remote: Total 3429 (delta 537), reused 392 (delta 184), pack-reused 2610
Recibiendo objetos: 100% (3429/3429), 577.98 KiB | 4.82 MiB/s, listo.
Resolviendo deltas: 100% (2356/2356), listo.
```

Figure 13: Cloning the Import Export repository

After the repository has been cloned, the next step consists of building it, using the `docker-compose build` command. Figure 14 shows how the command is executed from the terminal and the beginning of its output:

```
rievo@rievo-linux:~/development/opennext/import-export$ docker-compose build
WARNING: The JOBS_BASE_PATH variable is not set. Defaulting to a blank string.
redis uses an image, skipping
postgres-db uses an image, skipping
Building backend
Sending build context to Docker daemon 1.291MB
Step 1/7 : FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7
--> 86ade7dea2c7
Step 2/7 : WORKDIR /app
--> Using cache
--> 541a21b694e4
Step 3/7 : COPY ./app
--> c44717662086
Step 4/7 : ENV PYTHONPATH=/app
--> Running in a0032c3cdf42
Removing intermediate container a0032c3cdf42
--> 57f78e8alcff
Step 5/7 : RUN pip install pipenv
--> Running in dbb10326f8b0
Collecting pipenv
  Downloading pipenv-2021.5.29-py2.py3-none-any.whl (3.9 MB)
```

Figure 14: Building the Import Export service

Once all the services have been built, the third step consists of starting all of them. This step is done by means of the `docker-compose up` command.

¹² <https://git-scm.com/>

¹³ <https://github.com/wikifactory/import-export>

¹⁴ <https://github.com/OPEN-NEXT/import-export>

```

rievo@rievo-linux:~/development/opennext/import-export$ docker-compose up
WARNING: The JOBS_BASE_PATH variable is not set. Defaulting to a blank string.
Recreating import-export_redis_1 ... done
Recreating import-export_postgres-db_1 ... done
Recreating import-export_celeryworker_1 ... done
Recreating import-export_backend_1 ... done
Attaching to import-export_redis_1, import-export_postgres-db_1, import-export_backend_1, import-export_celeryworker_1
backend_1 | Checking for script in /app/prestart.sh
backend_1 | Running script /app/prestart.sh
postgres-db_1 | PostgreSQL Database directory appears to contain a database; Skipping initialization
postgres-db_1 |
redis_1 | 1:C 13 Jul 2021 13:31:31.996 # o000o000o000o Redis is starting o000o000o000o
redis_1 | 1:C 13 Jul 2021 13:31:31.996 # Redis version=6.0.10, bits=64, commit=00000000, modified=0, pid=1, just started
redis_1 | 1:C 13 Jul 2021 13:31:31.996 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
postgres-db_1 | 2021-07-13 13:31:32.133 UTC [1] LOG: starting PostgreSQL 12.6 (Debian 12.6-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit
redis_1 | 1:M 13 Jul 2021 13:31:31.997 * Running mode=standalone, port=6379,

```

Figure 15: Starting the Import Export Service

As can be seen in Figure 15, once all the services are recreated, they are ready to cooperate. Two notes must be raised here. First, in the left side of Figure 12 the names of the different services can be seen with different colours (e.g., *backend*, *postgres-db* or *redis*). This coloured output may vary between different terminals. Second, by using the “*docker-compose up*” command all the services are recreated from the terminal and their outputs are tied to it. In order to start the services while decoupling its output from the current terminal, it is possible to use the *-d* (detach) parameter of the command. For example, “*docker-compose up -d*”.

In order to stop all the services, there are two possibilities, depending on how the services were started. If the output of the services is still tied up to the terminal, the key combination *Ctrl + C* can be used. That will stop all the services and return the control of the shell. With regards to the other possibility in which the services have been raised on the background using the *-d* parameter, the required command to stop all the services is “*docker-compose down*”. By using that command, the services will no longer be accessible.

At this point, the Import Export microservice is ready to accept request and start performing import-export *jobs*.

2.2. Step-by-step example

The Import-Export tool exposes its functionality through a HTTP REST API. In order to showcase how it works, we’ll walk through different actions, triggering them using a tool called [Postman](https://www.postman.com/)¹⁵. Any other tool or library for sending HTTP requests, such as command line based [curl](https://curl.se/)¹⁶ would also work.

In a real world-scenario, it would be up to the particular integration how these actions would be triggered. Usually, it’s another backend service who would be sending these HTTP requests to the Import-Export tool.

In order to showcase an integration, after this example we will also walk through the integration done at Wikifactory.

¹⁵ <https://www.postman.com/>

¹⁶ <https://curl.se/>

2.2.1. REST API

The first step consists of performing a REST POST request to the `/job` path. An example of the required request can be found in Figure 16, which has been captured using the Postman tool:

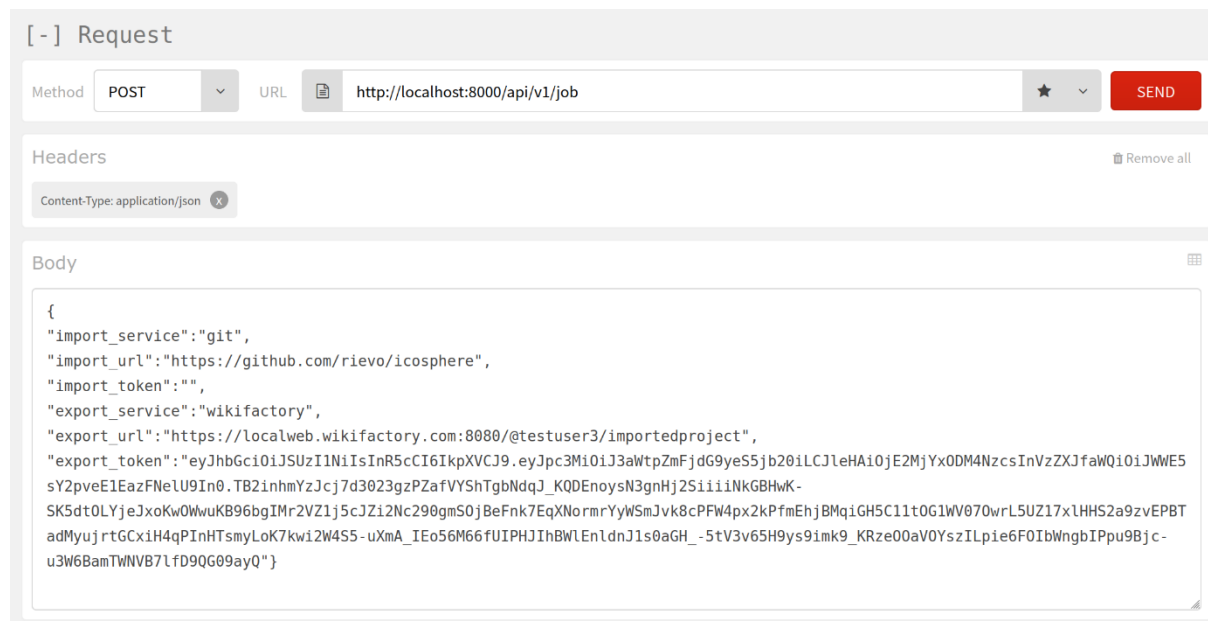


Figure 16: Screenshot of the HTTP POST request for starting an import-export job

The screenshot depicted in Figure 16 can be divided in three different sections, from top to bottom.

In the upmost section appears the REST method (POST) as well as the target URL. In this case, since the Import Export tool has been executed in the same machine as the one used to perform the request, the URL is <http://localhost:8000>. The `/api/v1` component of the URL refers to the different versions of the API that can be served by the microservice. However, the current implementation of the Import Export tool only has version 1 defined. Finally, in order to start a job, the last piece of the URL consists of the `/job` path.

In the central section of Figure 16, the headers of the REST request can be seen. In this case only one is required: *Content Type: application/json*. This header indicates the format of the body, which is defined on the Body section.

Finally, the body section of Figure 13 shows the JSON object that will be sent within the REST request. This example illustrates the process of importing from a public GitHub repository towards a project inside the Wikifactory platform. All the parameters shown in the example are required to start the job. The fields of this JSON object and their meanings can be seen below:

- *import_service*: String identifier of the service/platform from where the tool will import the files. In this case, the selected one is “git”. The remaining ones are “dropbox”, “wikifactory” and “google_drive”. Note that all the identifiers are in lower case.
- *import_url*: URL address from which the project will be imported. This URL must be in accordance with the service specified in the previous field and it must point to an existing resource.
- *import_token*: Authentication token required for accessing the files. In the case of Figure 13, this field has been left empty. This is because the GitHub repository to be imported is public, and it can be cloned without any kind of credentials. For other services such as Dropbox and

Google Drive, this field will be required. More details about authentication tokens and how they can be obtained will be presented in Annex X.

- *export_service*: String identifier of the service/platform towards the tool will export the project. In case of Figure 16, “Wikifactory” has been specified. The possible values are the same as for the *import_service* field.
- *export_url*: URL address for the destination of the files. It is always assumed that this URL points to an already existing resource, be it a GitHub repository, a folder in Dropbox or Google Drive, or an existing project in Wikifactory. As said before, the URL specified in Figure 16 is an example of this last option, as it points towards a project inside the Wikifactory platform.
- *export_token*: Authentication token required for creating the files in the target service (or, in other words, to upload them). The shape of this token depends on the protocol defined by the *export service*. For example, Google Drive and Dropbox use OAuth 2.0 tokens, while Wikifactory has its own format. More details about authentication tokens can be found in Section X.

Once executed the REST request as explained above, the Import Export service returns a response like the one depicted in Figure 17:



```

[ - ] Response
Headers  Response  Preview
1 { "import_url": "https://github.com/rieho/icosphere", "export_url": "https://localweb.wikifactory.com:8080/@testuser3/importedproject", "import_service": "git", "export_service": "wikifactory", "import_token": "", "export_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ3aWtpZmFjdG9yeS5jb20iLCJleHAiOiJlZmJyODM4NzcsInVzZXJfaWQiOiJWWE5sY2pveE1EazFNeLU9In0.TB2inhmYzJcj7d3023gzPZafVYShTgbNdqJ_KQDEnoysN3gnHj25iiiiNkGBHwK-SK5dt0LYjeJxoKw0WwuKB96bgIMr2VZ1j5cJZi2Nc290gmS0jBeFnk7EqXNormrYyW5mJvk8cPFW4px2kPfmEhJBMqiGH5C11t0G1WV070wrL5UZ17xLHHS2a9zvEPBTadMyujrtGCxiH4qPiNHTsmYLoK7kwi2W455-uXmA_Ie056M66fUIPHJIHbWLEnlDnJ1s0aGH_-5tV3v65H9ys9imk9_KRze00aV0YszILpie6FOIbWngbIPpu9Bjc-u3W6BamTWNVB7lfD9QG09ayQ", "id": "e7f38546-e422-47a6-afab-5532b9f0ceee", "status": "pending", "general_progress": 0.0, "status_progress": 0.0 }

```

Figure 17: JSON response for a newly created import-export job

The response in Figure 17 contains a JSON object with the same parameters presented earlier (*url*, *service* and *token* for both, the import and export services) with some new ones. The new parameters are:

- *id*: Unique identifier assigned to this importing-exporting *job*. This id can be used later to retrieve the current status of the job.
- *status*: String identifier of the current step of the process. In case of Figure 17, the status is *pending*, which means that the job has been queued and will start shortly.
- *general_progress*: A numerical representation of the current status of the process ranging between 0 and 1. At this moment, there are four possible values for this parameter:
 - A value of 0 means that the process just started. This is the value returned in the response depicted in Figure 17.
 - A value of 0.25 means that the tool is importing (i.e., downloading the files)
 - A value of 0.5 means that the tool has finished downloading the files from the import service and has started to uploading them to the export service.
 - Finally, a value of 1 means that the whole importing-exporting process has finished, and all the files can be found in the export service.
- *status_progress*: Percentage of the job already done for the current step of the process, ranging between 0 and 1. In other words, if the process is *importing* the files, this field is calculated by dividing the number of already imported files by the total number of files that

must be imported. A similar approach is used for the *exporting* process. In the example of Figure 17, the value of this field is 0.0, because the process just started.

By performing the REST request, the importing-exporting job was initiated and after some time, it will either still be running or finished. The current status of any job in the system can be retrieved at any moment by performing a REST GET request to the service, by specifying the *job's* unique id. Figure 18 shows a sample response for a job that has finished the whole import-export process.

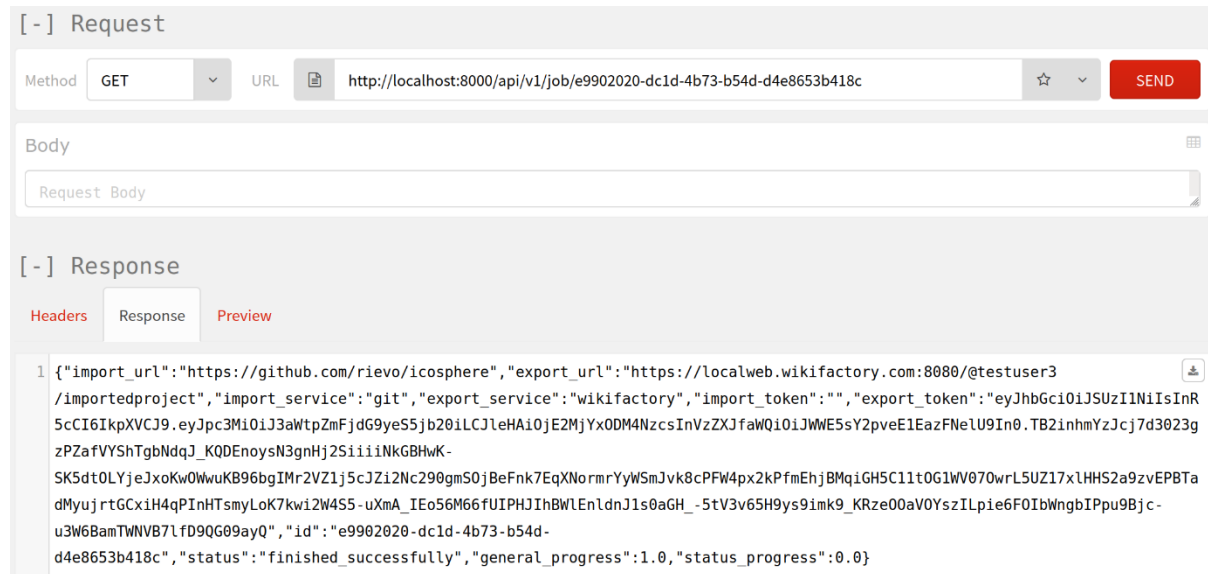


Figure 18: REST GET response when asking for a certain job

Note that the *status* field of the response indicates that the process has “*finished_successfully*” and that the *general_progress* has a value of 1.0 (i.e., it has finished).

2.2.2. Wikifactory Integration

With regards to the second scenario (i.e., the integration of the service within the Wikifactory platform), the demonstrated use case here is then, importing from either GitHub, Dropbox, or Google Drive to Wikifactory.

A new option is shown to the users in the Wikifactory web page when creating new projects in the platform. The user can either start an empty project from scratch or import an already existing one. Figure 19 shows the menu that the user sees when creating a project.

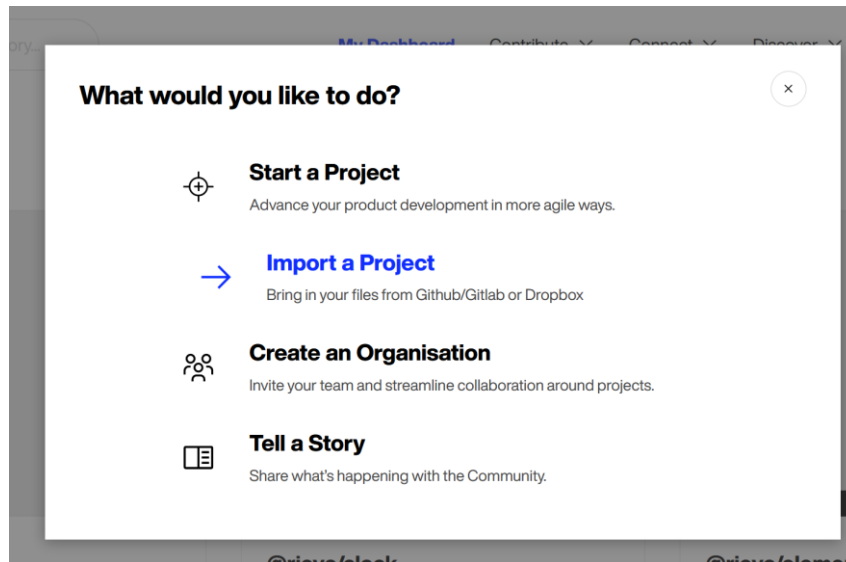


Figure 19: Creating a new project in Wikifactory

In case that the user selects the *Import a Project* option, the web page asks for the details of the project to be imported. Figure 20 shows the parameters that the user must fulfil.

Figure 20: Importing a Google Drive folder from Wikifactory webpage

From top to bottom of Figure 20, the first field is where the user must paste the Project or folder URL, which is the equivalent to the *import_url* of the project to be imported. In Figure 17 it is a shared link for a Google Drive folder. The next field is called “*Workspace*”, and this one is not related with the Import Export service, but with the Wikifactory platform. In short, it means that the project to be created will be associated with the user’s space. Next, the *Name* field is used to define the name that the project will have inside Wikifactory. Besides that, this name will be used alongside the workspace to automatically create the *export_url*. Finally, the user can decide between making the project public or private.

Once those fields have been completed and the user clicks the “*Begin import*” button, a new screen is shown to the user. In the background, the Import Export service has tried to access the files from the *import_url*, with no success. Then, as explained in Section 2.2.1, the import-export job is in the “*waiting*”

for authorization” status. The user is able now to authenticate on the *import* platform and grant access to the import service, as shown in Figure 21:

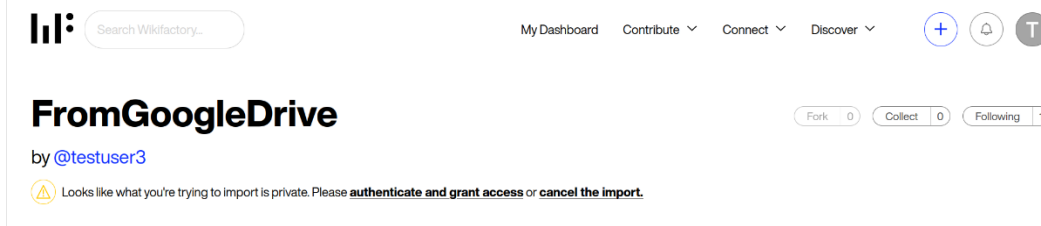


Figure 21: Waiting for the user authorization when importing from Google Drive

When the user clicks in “*authenticate and grant access*”, the authentication popup appears as shown in Figure 22 :

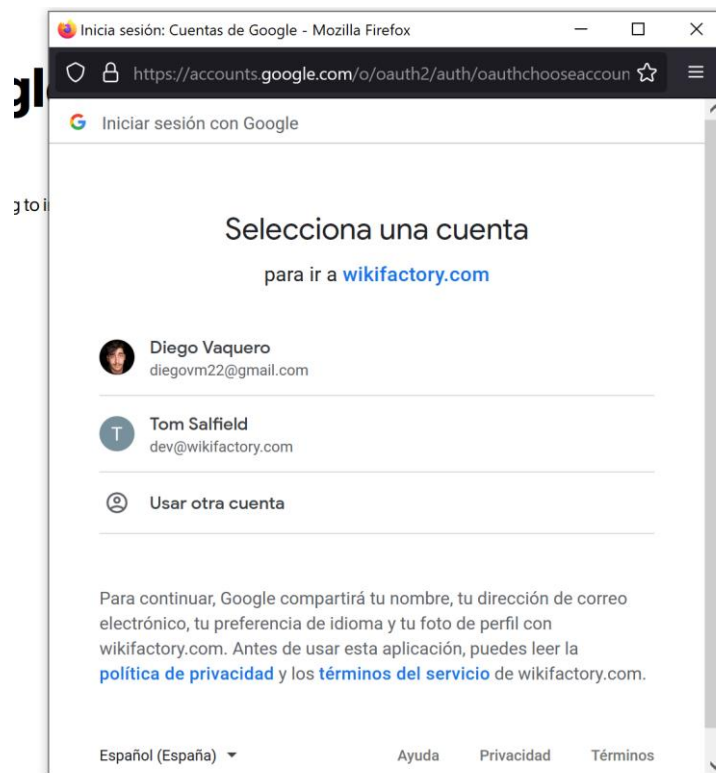


Figure 22: Oauth authentication popup

Finally, once the user has logged in using the credentials, the import-export job is retried and after a while, it will be completed. Then, the user can see a copy of the files inside Wikifactory, as Figure 23 illustrates:

FromGoogleDrive

by @testuser3

Fork 0 Collect 0 Following 1

Overview
Files
Issues
Settings

Drive (latest)

[New folder](#) [Upload file](#)

Name	Last Contribution	Last Modified / Actions
Water_level	Import files (#521df83)	a few seconds ago

[Download](#)

History (1)

Import files

#521df83 | contributed by @testuser3 | a few seconds ago

Figure 23: Inspecting the imported files in Wikifactory

Thus, the two presented scenarios illustrate how the Import Export service can be consumed either from the end-user (using the presented web interface or any other one) and from an external platform, such as in the case of Wikifactory.

2.3. Summary & Outlook

The current version of the deliverable consists of a fully independent microservice that can import files and folders from four different services (i.e., Wikifactory, GitHub/Gitlab, Dropbox and Google Drive) and export to those same four services.

Even though the tool has been developed with a microservices approach, interfacing with other backend services, there’s a reference implementation for a web interface that can be downloaded from here: <https://github.com/wikifactory/import-export-web>¹⁷

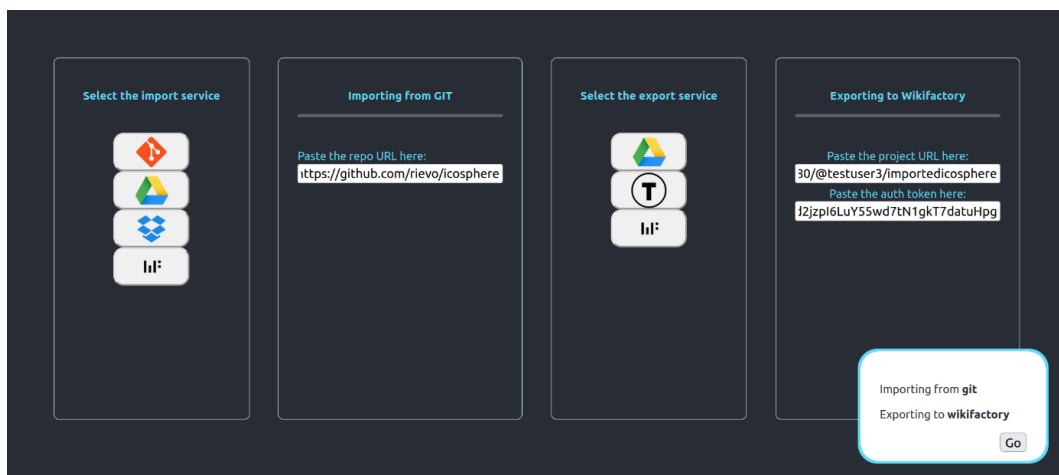


Figure 24: Web interface for the Import Export service

¹⁷ <https://github.com/wikifactory/import-export-web>

As can be seen in Figure 24, the interface has been divided into four different columns, being the two on the left side of the image associated with the *import service* and the two on the right side with the *export service*. Thus, the user can select any service from the first column using the icons present there and then, the interface will show the required elements for that specific service. For example, in the Figure 16 the user has selected *Git* as the *import service* (note that it could be either for GitHub or GitLab), and has pasted the URL of the repository to be imported. Similarly, the user has selected Wikifactory as the *export service* and the service shows two different fields for it: i) a field for URL of the already created project in Wikifactory, and ii) a field for the auth token (which is used to authorize the importing process).

Regarding next steps, the plan is to:

- Support additional platforms such as [Instructables](#)¹⁸, [MyMiniFactory](#)¹⁹ and potentially [Thingiverse](#)²⁰
- Improve the data syncing capabilities so the process can be executed multiple times for successive updates. Ideally those would be performed automatically based on changes to the source
- Allow other integration modes in addition to the microservice approach, such as using the tool as a Python module. As an example, projects could also be loaded into [Jupyter](#)²¹ notebooks, such as [Google Collab](#)²² or results from Python code could be uploaded as projects to the target platforms this way.
- Improve download/upload performance and error handling.

¹⁸ <https://www.instructables.com/>

¹⁹ <https://www.myminifactory.com/>

²⁰ <https://www.thingiverse.com/>

²¹ <https://jupyter.org/>

²² <https://colab.research.google.com/notebooks/intro.ipynb>

3. Skill based matchmaking

Task assignments in a dynamic project environment can be a challenge in collaborative environments, complicating the management of project members and contributors. The skill based matchmaking demonstrator aims at connecting projects and communities based on skills. Hence, the target groups here are the communities and the project team (for example from a company). The scope of this demonstrator is defined by three user flows (defines the flow of activities of a user on an online platform):

1. As a contributor on an OSH online platform, I want to add my skills and interests to my profile, so that others know more about my abilities
2. As a contributor on an OSH online platform, I want to find projects that require my specific skills in tasks, so that I can use, improve or evolve my skills
3. As a project core team, we want to find contributors based on their skills and interests so that they can help in carrying out a specific task.

The demonstrator’s core is based on a skill ontology, which consists of a set of skills which are in turn connected to community members, projects and tasks (present in the OSH project ontology) based on a set of rules and logic. This enables OSH project community members and project teams to find contributors based on their skills. As shown in Figure 25, the demonstrator system also consists of links between the ontology and the platform itself. The platform consists of all the actual data, which needs to be connected to the ontology. This process is called instantiation. When a contributor/ project core team member interacts with the platform and request to find certain information, that is when the querying kicks in and provides matched information from the ontology.

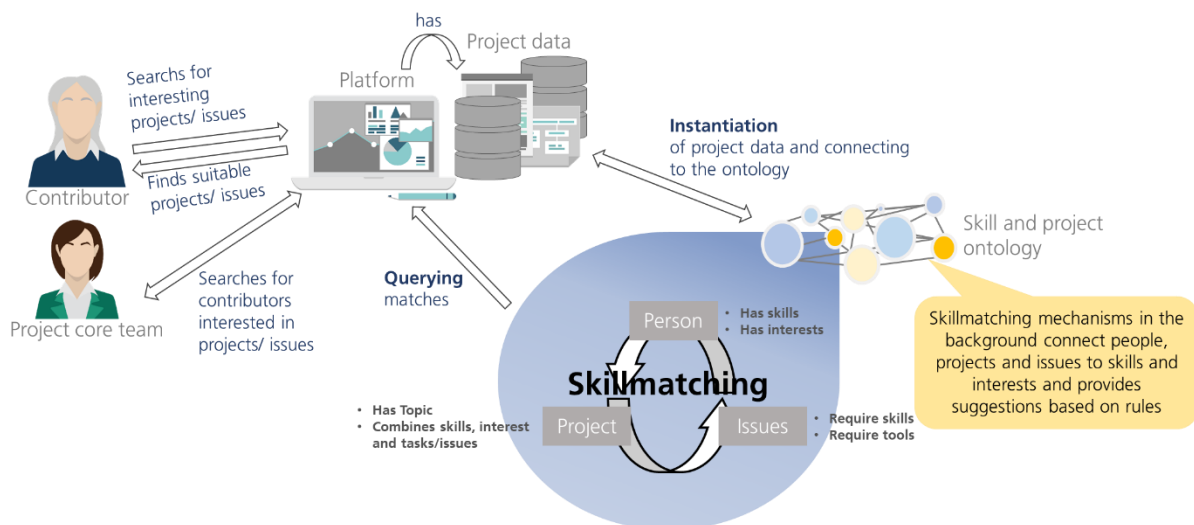


Figure 25: Skill based matchmaking concept

A more graphic example illustrates the above statement: A project needs a customization of a product component. For this, a project member designs some customized part to integrate into the assembly for the project’s purposes. The part is designed to be 3D printed and needs to be tested for functionality and durability in the assembly. The project now needs some contributor experienced in 3d printing to choose the right material and printing parameters. When the demonstrator is connected to the data sources/ platform, the ontology instantiates the data about the users and the projects and connects them to identified skills (which are identified by the users and projects themselves). A project member could use the demonstrator directly (or on the platform) to search (query) for community members that are connected to the process *3dprinting* or query for members that are connected to a

3dprinter instance over the skill action “*operates*”. Hence, enabling finding related community members. This also works in the opposite direction, where the community members can search (query) and find project tagged with skills required. How the demonstrator can be used also in connection with a platform like Wikifactory is further detailed in subsection 3.2 with a step by step example.

The following pre-requisites are recommended for the following sections:

- Basic knowledge about Ontology and semantics. An ontology is a concept used to model semantics (study of meaning, which enables developing parts of a language that can be understood and used commonly). A comprehensive overview can be found in (Guarino et al. 2009).
- Knowledge about the constitution of RDF and OWL graphs, and constants: RDF²³ and OWL^{24, 25} are used to model semantics. Both use triple patterns to create graphs interrelating resources and setting them into an interconnected network. An introduction into the topic is for example explained by (Pan et al. 2017). For the general and further understanding for this document, a shortened description is provided in the annex section 3.4.1. SPARQL^{26,27,28} is a query language for RDF data using graph patterns to restrict query response results. The referenced links provide a comprehensive overview about the functionality of SPARQL.
- JSON and JSON pointer: (Pezoa et al. 2016) provides understandable insights for a quick introduction to the topic: JSON is based on the Java Script programming language and is used for web applications in order to send and retrieve API requests and responses. JSON pointer are both, a query language and a concept used for information retrieval from JSON documents via defined formatted strings, relating to a unique value in the JSON document.
- JAVA and JAVA APIs OWLAPI²⁹ and JENA³⁰ knowledge is helpful for developers who want to contribute to the demonstrator code. JENA API is an RDF open-source-framework for semantic networks. Most functionality is given for RDF graphs (like loading, serializing, saving, reasoning and querying) but functionality is also given to some extent on OWL based graphs. (Antoniou und van Harmelen 2009) indicate which expressions overlap in RDF and OWL and which ones shift the scope of RDF. For OWL ontologies the OWLAPI provides complete functionality.
- Since the repository is stored on GitHub, general knowledge about GitHub and Git is assumed, referring to the GitHub guides³¹.

3.1. Usage

The demonstrator has two the main functions: instantiation and querying. Each of their usages are described below:

i) Instantiation

An ontology was created for the development goal of skill matching. To make it widely applicable, it was created without any instances. Project data from WIF was integrated via data mapping afterwards.

²³ <https://www.w3.org/TR/rdf-schema/>

²⁴ <https://www.w3.org/TR/owl-features/>

²⁵ <https://www.w3.org/TR/owl2-overview/>

²⁶ <https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/>

²⁷ <https://www.w3.org/TR/sparql11-query/>

²⁸ <https://www.w3.org/TR/sparql-features/>

²⁹ <https://github.com/owlcs/owlapi/>

³⁰ <https://jena.apache.org/>

³¹ <https://guides.github.com/>

This offers the advantage of simpler validation in case the data structure changes. If so, a new instantiation can simply be achieved by changing the mappings. The mapping approach is based on (Méndez et al. 2020) and connects the ontology concepts with JSON formatted input data via annotation properties. That means, that the mapping information is directly stored in the ontology itself. The mapping process is visualized in Figure 26. Classes and properties in the semantic network are annotated with JSON pointers to the relating fields in the GraphQL API from WIF whose data was used for validation. For every mapping a new annotation property is created. This gives the possibility to use more than one data source from either the same or a different platform. During later instantiation it is possible to choose in the demonstrator code how the mapping should be used.

For the user flow three different JSON data source files were created from query results of the WIF GraphQL API. This was done, so the first validation was not compromised by changes in the data structure during its constant progression and also because the user data had to be anonymized before used for instantiation.

During the mapping process, the annotation with the JSON pointer is read out of the ontology and used to query the information in the JSON file that contains the input data. Afterwards those query results are instantiated based on which concept (class or property) was annotated in the first place. The instantiated individuals are saved in a separate file that uses the ontology vocabulary.

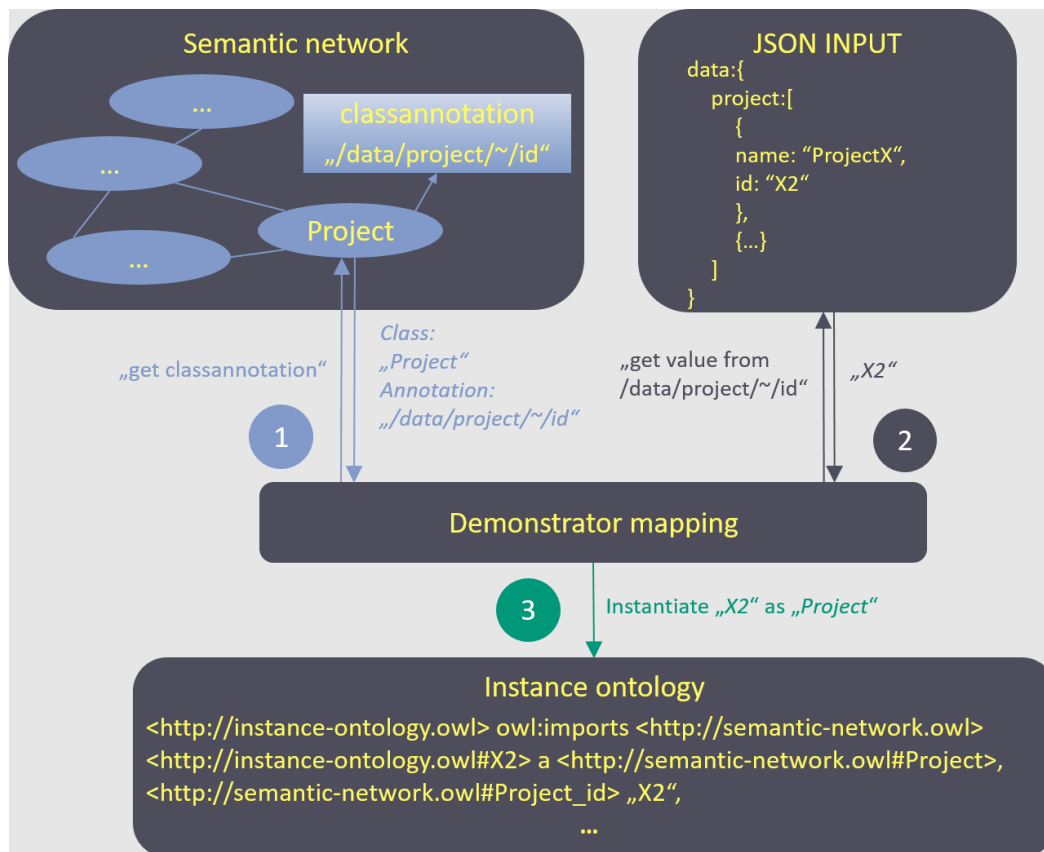


Figure 26: Mapping process

ii) Querying

After instantiation, the semantic network can be queried. The query functions in the demonstrator use a query string as input, therefore individual queries are also possible, if implemented in the code. More on this is explained in the step-by-step example section for querying.

The following subsections refer to Classes and Objects from the demonstrator code. For a better understanding it is advisable to study the class diagram and the classes’ descriptions in the annex section 3.2.1 to get a general overview about the functionality of the separate classes.

3.2. Step-by-step example for contributor and project core team

This sub-section provides an example of how a contributor and a project core team member can interact with the skill-matchmaking demonstrator. The contributor would like to update his/her profile and hence can select the suitable skills from a list of skills provided by querying the skill ontology. The project core team member can select the skills and tools required to fulfil a task in the project by querying the skill ontology in the backend. The current demonstrator is integrated with data from the Wikifactory platform and an example of these query results is shown below.

Direct user interaction with the ontology takes place in user flows 2 and 3. The query process of those will be described in detail. To query the instantiated ontology, the code in the demonstrator can be used or every ontology editor with query interface as well (one suggestion would be the Protégé editor with SPARQL plugin³²). The demonstrator provides queries based on the persona described. For the two user stories, the two queries are shown in Table 2 .

Table 2: SPARQL query examples

UF No.	SPARQL Query
UF2	<pre>SELECT ?User ?Skill_Entity ?Project WHERE{?User a oshpd:User; oshpd:User_id "uid113"; skills:skill_action ?Skill_Entity. ?Skill_Entity oshpd:tags ?Project. ?Project a oshpd:Project.}</pre>
UF3	<pre>SELECT ?User ?Skill_Entity WHERE {?Skill_Entity skills:SkillEntity_name "3dprinting"; skills:possible_action ?User.}</pre>

The query for UF2 looks for projects that are tagged with the same skills a user (e.g. user “uid113”) provides.

The query for UF3 looks for all users that are connected to the *3dprinting Skill_Entity*.

a) Using the query code in the demonstrator

There is also a query class provided in the demonstrator with some example queries. To use those, you need to clone the repository or copy the source code of the *Queries.java* class in the query package. For other queries, those have to be written in strings and given as parameters into the query function. Flow charts in the annex section gives a better illustration of the process.

- 1) At the beginning the repository has to be cloned or downloaded. (This step depends on the environment used and should accordingly be identified on an individual basis.)
- 2) The *Queries.java* class from the query package provides example queries according to the presented user stories that can be used. Otherwise new methods that return an individual query string can be added as well.

³² <https://protege.stanford.edu/>

```
public String UserSkillInterest() {
    String query="SELECT ?User ?Skill_Entity\r\n" +
        "WHERE {?User oshpd:interested_in ?Skill_Entity.\r\n" +
        "?Skill_Entity a skills:Skill_Entity.}\r\n";
    String out= prefixes+query;
    return out;
}
```

Figure 27: Method to create a query string for user interest

- 3) If the ontologies were to be changed, the prefixes in the string variable *prefixes* of class *Queries.java* need to be changed accordingly as well.

```
public class Queries {
    private String prefixes="PREFIX : <https://github.com/konierik/Skillmatching/raw/main/Skillmatching/data/on_Instances.owl#>\r\n" +
        "PREFIX owl: <http://www.w3.org/2002/07/owl#>\r\n" +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\r\n" +
        "PREFIX xml: <http://www.w3.org/XML/1998/namespace>\r\n" +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\r\n" +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\r\n" +
        "PREFIX oshpd: <https://github.com/konierik/Skillmatching/raw/main/Skillmatching/data/on_OSHDPD_schema.owl#>\r\n" +
        "PREFIX skills: <https://github.com/konierik/Skillmatching/raw/main/Skillmatching/data/on_skills.owl#>\r\n";
}
```

Figure 28: Query prefixes

Now the *RunQuery.java* class from the run package has to be specified.

- 4) The internationalized resource identifier (IRI) of the ontology to query has to be set in the *RunQuery.java* class (cf. Figure 29).
- 5) The language of the model to open has to be specified in the *openModel()* method in *RunQuery.java*.

```
QueryExec searchforme= new QueryExec();
searchforme.setIRI("https://github.com/konierik/Skillmatching/raw/main/Skillmatching/data/on_Instances.owl");
searchforme.openModel("OWL");
```

Figure 29: Setup for query execution using *QueryExec.java*

- 6) The query execution method of class *QueryExec.java* needs a SPARQL query as string parameter. For this, the query string generation methods of class *Queries.java* can be used.

```
Queries q= new Queries();
LinkedList<ArrayList<String>> result=searchforme.execQuery(q.UserSkillInterest());
```

Figure 30: Execution of query using the query generation method

- 7) The generated result list can now further be processed, e.g. for frontend display.

b) Using editor with SPARQL functionality

To show the general function, an example of the use is given with the ontology editor Protégé which also provides a SPARQL plugin for query.

- 1) Loading the ontology from its location. Here the repository location was used.

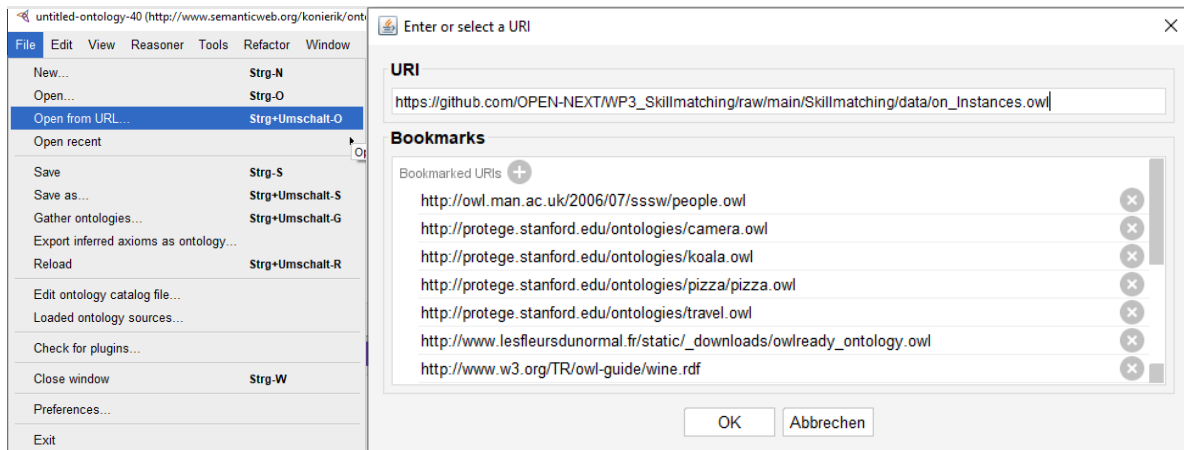


Figure 31: Load ontology from URI via Protégé

2) The editors SPARQL- /Query- interface needs to be opened

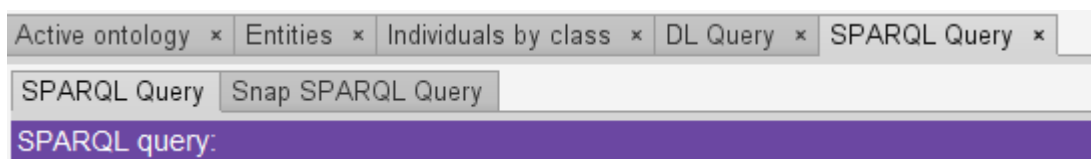


Figure 32: SPARQL plugin tab in Protégé

3) Adding prefixes and query string to the interface and execution of query. Figure 33 exemplary shows the US2 query.

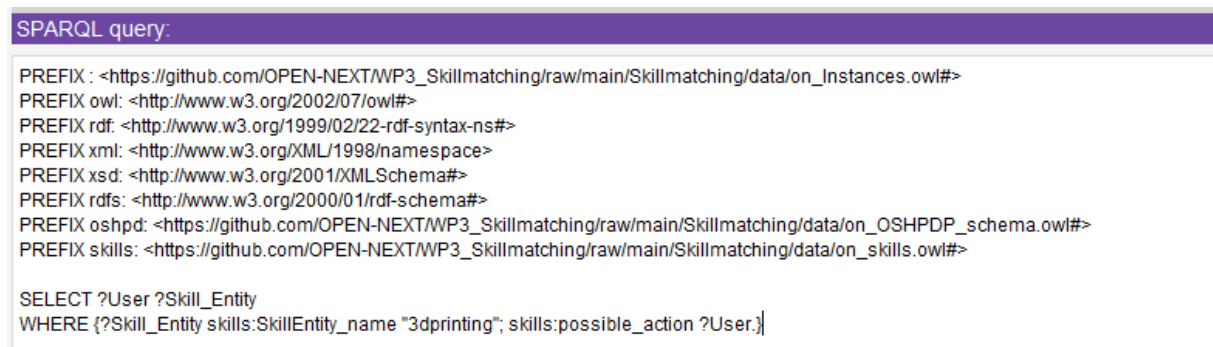


Figure 33: US2 Query with prefixes

4) Query results

User	Skill_Entity
:uid105	:3dprinting
:uid41	:3dprinting
:uid92	:3dprinting

Figure 34: Query results US3

User	Skill_Entity	Project
:uid113	:3d-printing	:UHJvamVjdDoxMjQ5
:uid113	:3d-printing	:UHJvamVjdDo3MjA=
:uid113	:3d-printing	:UHJvamVjdDoxMTE4
:uid113	:3d-printing	:UHJvamVjdDo2NzM=
:uid113	:3d-printing	:UHJvamVjdDoxMjI1
:uid113	:laser-cutting	:UHJvamVjdDo0MTAz
:uid113	:laser-cutting	:UHJvamVjdDoxMjYx
:uid113	:laser-cutting	:UHJvamVjdDo5MzQ=
:uid113	:laser-cutting	:UHJvamVjdDo5Nzg=
:uid113	:laser-cutting	:UHJvamVjdDoxMjMx
:uid113	:laser-cutting	:UHJvamVjdDoxMjQ2
:uid113	:laser-cutting	:UHJvamVjdDoxMDAw

Figure 35: Query results US2

Figure 36 shows an example of suggested projects for a certain user on the WIF platform frontend. From here, the user can click any of the suggestions in order to explore them.

Browse Similar Projects

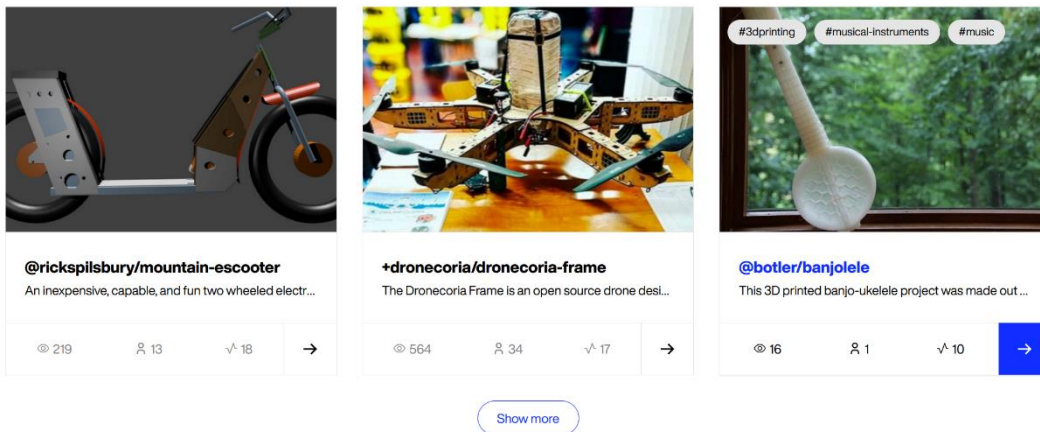


Figure 36: Project suggestions to a user of WIF platform

Figure 37 illustrates the process of adding new skills to a user profile. These new skills are defined by means of tags (defined using the # character). For example, the user shown in Figure 37 indicates that

the user has the *3D printing* and *Augmented Reality* skills, and the user is adding a new one called *#programming*. Wikifactory shows some suggestions of skills, such as *#arduino-programming*.

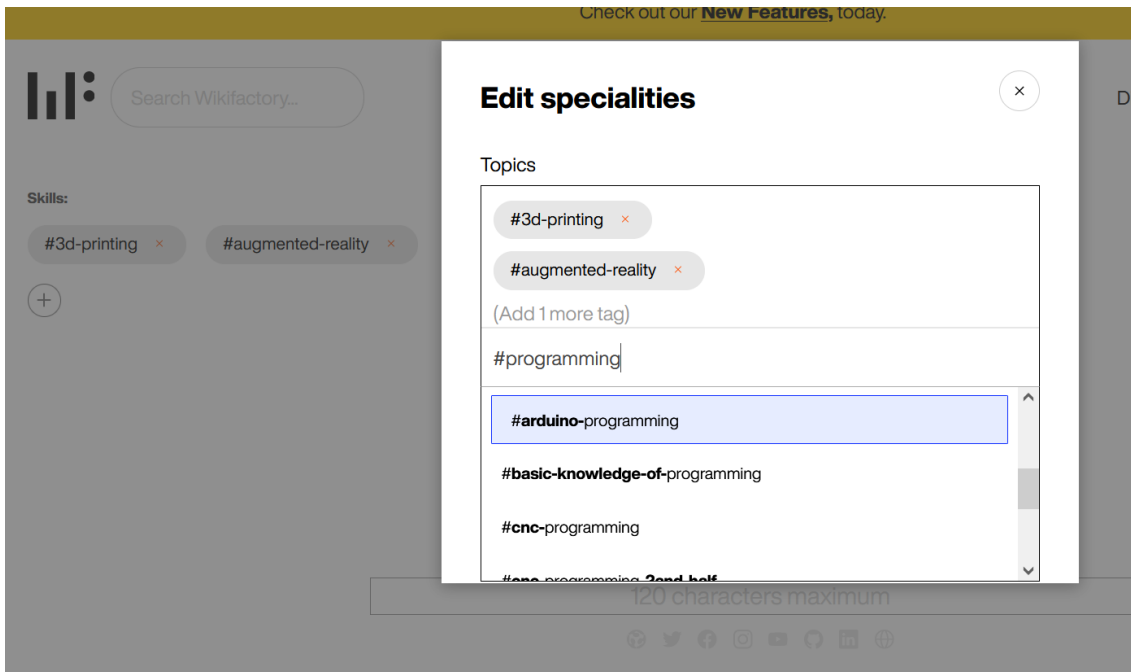


Figure 37: User profile skill enrichment layout on the WIF platform

Once the user has provided the skills, they are now visible from his profile, which is depicted in Figure 38.

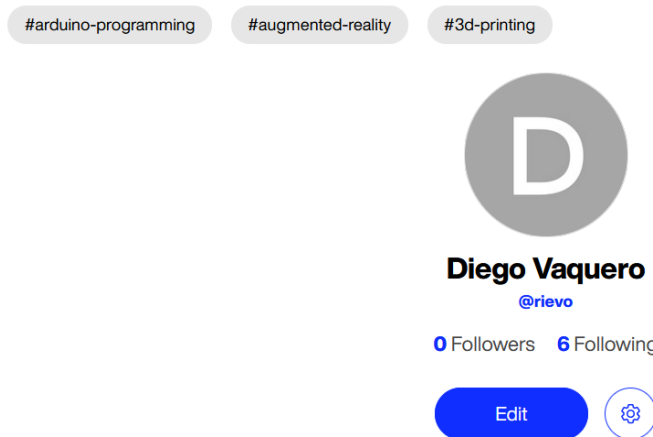


Figure 38: WIF user profile with skills tags

The same process of adding skills can be performed for projects, as Figure 39 shows:

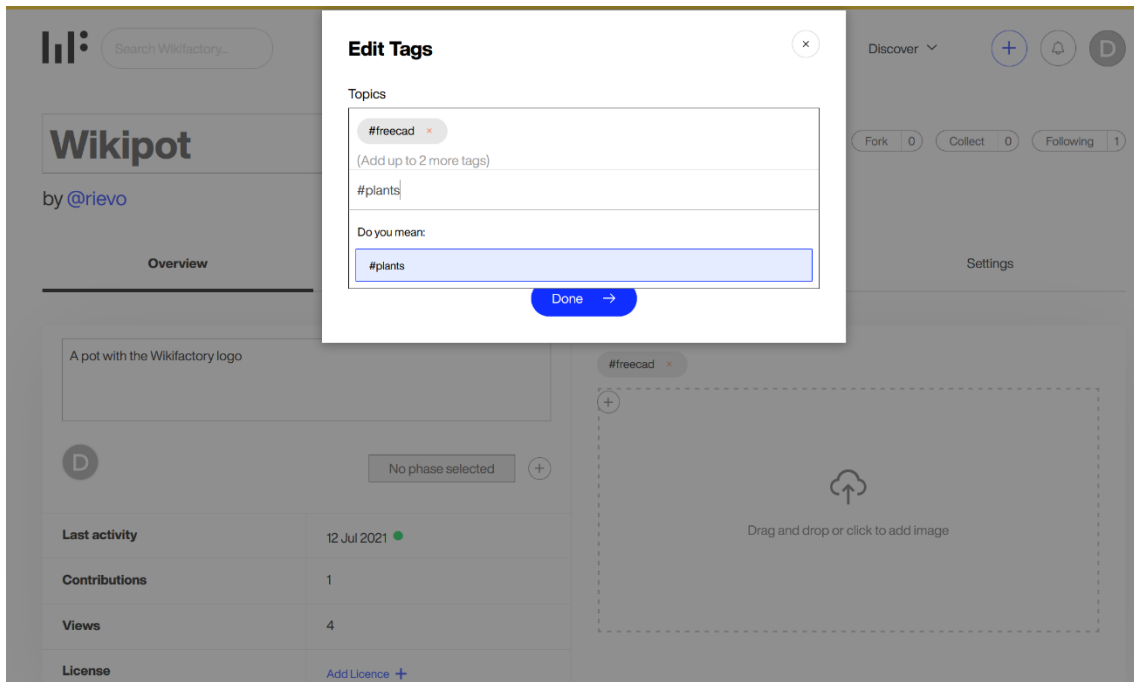


Figure 39: Adding skill tags to a project on WIF platform

Finally, once the project has skills associated, they are visible from the project page as well as in some other points, such as in the suggestions shown in Figure 36 or in the *Featured Projects* section shown in Figure 40.

Featured Projects

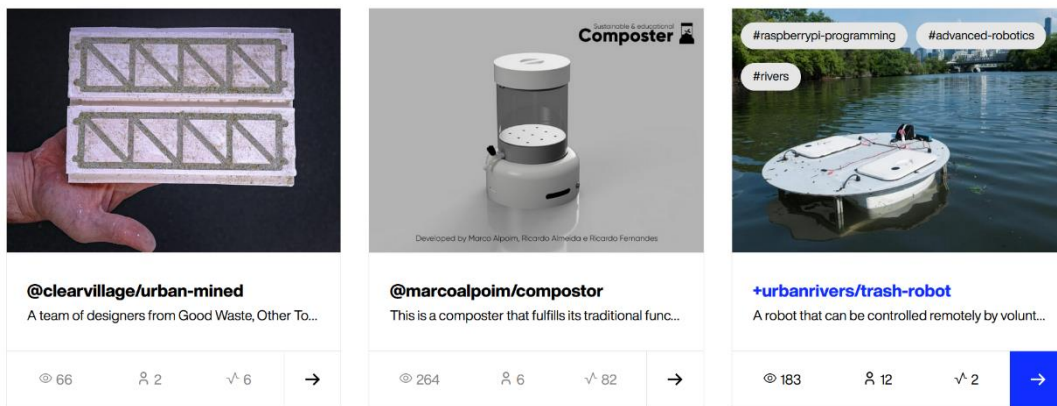


Figure 40: Featured project section on WIF platform

3.3. Step-by-step example for platform owner

The core ontology is open source and can be used by other platforms similar to WIF for carrying out skill based matchmaking. There are two different ways platforms can use and integrate the demonstrator, they are:

- i) Direct use – To reuse the ontology directly
- ii) Customized use - To customize and use the ontology based on the platforms need

To connect the ontology to the platform data the next step is to carry out instantiation, followed by developing suitable queries to retrieve matchmaking information from the ontology. Each of these steps are explained in the following sub-sections.

Reference the ontologies or use for own purpose

i) Direct use

To use the ontology for own data connection, e. g. for any other platform owner, it is possible just to refer to its location on the repository as namespace, which is the first stable release accessible under this permalink

https://raw.githubusercontent.com/OPEN-NEXT/WP3_Skillmatching/v1.0.0/Skillmatching/data/on_Instances.owl

and use the vocabulary for instantiation. The ontology is public and directly accessible from the repository.

ii) Customized use

Custom ontology

The following section refers to the constitution of the OWL ontology files and to classes used in the code. The constitution of an OWL file is further described in the annex section 3.4.2. The class structure of the code and explanation of its functionalities is given in the annex section 3.2.1. In case of confusion, it is recommended to read them.

If it is desired to adapt or extend the ontology for individual use, e. g. if the data structure requires changes, the ontology can be downloaded from the repository and edited. For the ontology to be accessible, it is suggested to change the ontology namespaces in the ontology file (*on_OSHPDP_schema.owl* and *on_skills.owl*) from the repository namespace to the new ontology location (cf. Figure 41). This should also be considered for the import statement if it involves the respective ontologies (see bottom of Figure 41). Depending on which ontology is downloaded relating prefixes need to be changed as well (e. g. if the skills ontology is downloaded and adapted its prefix has to be changed in the OSH project ontology (OSHPDP ontology).

```
<?xml version="1.0"?>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
  xml:base="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPDP_schema.owl"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  ontologyIRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPDP_schema.owl"
  versionIRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPDP_schema.owl/0.1">
  <Prefix name="" IRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPDP_schema.owl#" />
  <Prefix name="dc" IRI="http://purl.org/dc/elements/1.1/" />
  <Prefix name="okh" IRI="http://purl.org/oseq/ontologies/osh-metadata/0.1/base" />
  <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
  <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
  <Prefix name="xml" IRI="http://www.w3.org/XML/1998/namespace" />
  <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
  <Prefix name="foaf" IRI="http://xmlns.com/foaf/0.1/" />
  <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
  <Prefix name="oshpdp" IRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHPDP_schema.owl#" />
  <Prefix name="skills" IRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_skills.owl#" />
  <Prefix name="dcterms" IRI="http://purl.org/dc/terms/" />
  <Prefix name="oldskill" IRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/ontology/skills.owl#" />
  <Prefix name="wikifactory" IRI="https://wikifactory.com/api/graphql/" />
  <Import>https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_skills.owl</Import>
```

Figure 41: Repository namespaces in the ontology code

So, for customized use, the following steps have to be considered/executed:

- 1) Download of the ontology to new folder structure (local or server)
- 2) Change of the namespace, prefixes and import statement in the ontology file accordingly to fit the new file's location(s).

Custom instantiation

The mapping process is quite trivial. So, if there is very complicated data, that has to be mapped with rules and exceptions, an expansion based on the current mapping approach or another mapping process is recommended.

- 1) Cloning or downloading the repository
- 2) Opening the mapping ontology file *on_OSHPDP.owl* in the *data* folder of the repository (e. g. with a text editor).
- 3) Adding custom data mapping:

To instantiate own JSON data, e. g. in case for a platform owner, that wants to connect own project data, a change of the respective annotation is necessary. The figures below show a class-mapping declaration (Figure 42) and definition (Figure 43) for the class *Project* in the ontology.

```
<Declaration>
  <AnnotationProperty IRI="#wif_project_2_cmap"/>
</Declaration>
```

Figure 42: Declaration of annotation property in an owl file

```
<AnnotationAssertion>
  <AnnotationProperty IRI="#wif_project_2_cmap"/>
  <IRI>#Project</IRI>
  <Literal>/data/projects/result/edges/~node/id</Literal>
</AnnotationAssertion>
```

Figure 43: Mapping annotation for Class "Project" with pointer string to json input file

For the given example, the mapping is called “*wif_project_2_cmap*”. The IRI section specifies which concept of the ontology is annotated (here the class *Project*) and the literal section includes a string of the JSON pointer, i. e. values for instances of the class *Project* can be found in the field “*/data/projects/result/edges/~node/id*” in the corresponding input file. The “*~*” marks an array in the JSON data. Such markers will be automatically resolved into all existing entries of the array in the JSON file.

For every new annotation property, a declaration axiom (Figure 42) and a definition (Figure 43) has to be added to the ontology file. That applies to classes, object properties and data properties in the ontology. Here for every concept “touched” by the instantiation, there has to be a mapping to JSON data. That means, for example to map a data property, there has to be a mapping annotation property to the data property itself but there also has to be a mapping annotation property to the relating domain class of the data property. To stay in the example: to map a Projects name, there has to be a mapping annotation property to the data property *Project_name* and a mapping annotation property to the corresponding domain class *Project* as well. the same logic is applied to object properties, but here are three mapping annotation properties are necessary, each one to the domain and range classes and one to the object property with a pointer to the specific individual.

If the input data has multiple possible pointers for a concept in the ontology, extra mapping annotation properties can be created. E. g. the shown annotation is the second class-mapping for the concept *Project*, because in the input data were additional project data that could be reached with other pointers.

Now it is time to prepare the main method in the *CreateInstances.java* class in the code for instantiation.

1. If the skill ontology or the OSHPDP ontology were changed, the corresponding string variables *skillIRI* or *mappingIRI* needs to be changed as well.


```
//variable for skill ontology ici
private static String skillIRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_skills.owl";
//variable for mapping ontology ici
private static String mappingIRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_OSHDPDP_schema.owl";
//variable for instance ontology ici
```

Figure 44: skillIRI and mappingIRI variables in CreateInstances.java

- The IRI of the instantiated ontology to be created has to be specified. If public access to the instances is desired, the later location should be taken as namespace (like it is done in Figure 45)

```
//variable for instance ontology ici
private static String instanceIRI="https://github.com/OPEN-NEXT/WP3_Skillmatching/raw/main/Skillmatching/data/on_Instances.owl";
```

Figure 45: instanceIRI variable in CreateInstances.java

- The directory variable needs to be changed according to where the cloned/downloaded project is located. Add the path right away until the data folder of the project. (So to say that just the files name and extension have to be added in later use)

```
//directory where the project is stored
private static String directory="C://Springboot-Repository//WP3_Skillmatching//Skillmatching//data//";
```

Figure 46: directory variable in CreateInstances.java

- For each JSON input file, a *JSONReader* object has to be instantiated and the location of the files needs to be set. At best they were already copied into the projects folder structure.

```
////////////////////////////////////
//
//      setup json reader
//
////////////////////////////////////

//create one JSONReader per data file: issues, projects, user
JSONReader readIssues=new JSONReader();
//set the file to read and open it
readIssues.setFile(directory+"sampledata_issues_anonym.json");
readIssues.open();

JSONReader readUser=new JSONReader();
//set the file to read and open it
readUser.setFile(directory+"sampledata_user_anonym.json");
readUser.open();

JSONReader readProjects=new JSONReader();
//set the file to read and open it
readProjects.setFile(directory+"sampledata_projects_anonym.json");
readProjects.open();
```

Figure 47: Setting up JSONReader objects for the JSON input in CreateInstances.java

- This step has to be done for every input source that should be instantiated:** The mappings that should be instantiated need to be set. The general case should be, that per mapping annotation there has to be instantiated a class mapping, object property mapping and data property mapping. Not all mappings from the ontology have to be instantiated if not necessary. The figure below shows the instantiation of each five class-, object property- and data property mappings that use the source *sampledata_issues_anonym.json* from the *JSONReader readIssues* (Figure 47).

```

////////////////////////////////////
//
//      Instantiate issue-mappings
//
////////////////////////////////////

//going through all 5 issue_mapping_annotations
System.out.println("Instantiate Issue-mappings");
for (int i=1; i<=5;i++) {
    /*Define what mapping annotations should be looked for (in case there are more mappings in the ontology).
    * Mappings can be named differently if the ontology is used to map several sources of different structure.*/
    mapping.setClassmapping("wif_issue_"+i+"_cmap");
    //mapping.setClassIdent("identifier");
    mapping.setObjectpropertymapping("wif_issue_"+i+"_opmap");
    mapping.setDatapropertymapping("wif_issue_"+i+"_dpmap");

    // Extract the pointers from the mapping annotations and iris of the respecting concepts as lists
    classannotations = mapping.getClassesAnnotations(); //Format: [[classmapping pointer][rdfsType][classIRI]]
    dataannotations = mapping.getDatapropertiesAnnotations(); //Format: datapropertyIRI|dataobjectmapping pointer
    objectannotations = mapping.getObjectpropertiesAnnotations(); //Format: objectproperty_IRI|objectpropertymapping pointer

    ntmapper.instantiateToNTClasses(classannotations, readIssues );
    ntmapper.instantiateToNTDataproperties(dataannotations, readIssues);
    ntmapper.instantiateToNTObjectproperties(objectannotations, readIssues);
}
System.out.println("\n\n");

```

Figure 48: Instantiation of mapping annotations in *CreateInstances.java*

In the given code of the demonstrator were three JSON input files. To instantiate all information of those, several mappings each were necessary, due to the repeated occurrence of the same classes in the same file. This is represented by the loop in Figure 48.

- For each mapping annotation to be instantiated an *ArrayList<ArrayList<String>>* needs to be added in the code (Figure 49). Those have then to be filled with the corresponding annotations using the *get..Annotations* methods of the *OntoModeler mapping* (cf. Figure 48).

```

54      //annotation arrays for later instantiation
55      ArrayList<ArrayList<String>> classannotations=null;
56      ArrayList<ArrayList<String>> dataannotations = null;
57      ArrayList<ArrayList<String>> objectannotations = null;

```

Figure 49: Lists to save the annotation properties in *CreateInstances.java*

- The NT-file output location and name need to be set (explained more detailed in annex section 3.2.1). The instances are firstly stored in NT-format and afterwards converted to RDF language.

```

184      //Set output string for file
185      String ntoutput=directory+"on_Instances.nt";
186      ntmapper.setNTOutputLocation(ntoutput);
187      ntmapper.toNTFile();

```

Figure 50: Set NT output in *CreateInstance.java*

- At the end the ontology location for saving in *CreateInstances.java* has to be set.

```

197      ntparse.addImport(skillIRI);
198      ntparse.setOutput(directory+"on_Instances.owl");
199      ntparse.parseNT(instanceIRI, "RDF/XML");
200
201      instance.saveOntology(directory+"on_Instances.owl");
202      } catch (OWLException e) {

```

Figure 51: Setting ontology saving location in *CreateInstances.java*

- The last step is to run the *CreateInstances.java* class

The reasoner to assert the inferences needs some time depending on the amount of data that should be instantiated. The instantiation process has ended, when the ontology saving statement (Figure 53) appears on the console.

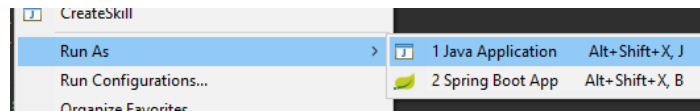


Figure 52: Running the CreateInstances.java class



Figure 53: Saving statement for the ontology on the console

Custom Querying

The query process for a platform owner works the same way as for contributors. The queries just vary based on intended results. An example for a platform owner is given below. The query answers the question, what projects can be suggested for users to participate and vice versa.

Table 3: Example query for platform owner

SPARQL Query		
<i>SELECT ?User ?Skill_Entity ?Project</i>		
<i>WHERE {?Skill_Entity a skills:Skill_Entity; oshpd:interest_of ?User; oshpd:tags ?Project.</i>		
<i>?Project a oshpd:Project.}</i>		
User	Skill_Entity	Project
:uid26	:laser-cutting	:UHJvamVjdDo5MzQ=
:uid3	:laser-cutting	:UHJvamVjdDo5MzQ=
:uid7	:laser-cutting	:UHJvamVjdDo5MzQ=
:uid18	:architecture-models	:UHJvamVjdDo5MzQ=
:uid7	:architecture-models	:UHJvamVjdDo5MzQ=
:uid11	:laser-cutting	:UHJvamVjdDo5Nzg=
:uid26	:laser-cutting	:UHJvamVjdDo5Nzg=
:uid3	:laser-cutting	:UHJvamVjdDo5Nzg=
:uid7	:laser-cutting	:UHJvamVjdDo5Nzg=
:uid7	:robotics	:UHJvamVjdDoxMDAw
:uid3	:robotics	:UHJvamVjdDoxMDAw
:uid4	:robotics	:UHJvamVjdDoxMDAw
:uid11	:robotics	:UHJvamVjdDoxMDAw
:uid27	:robotics	:UHJvamVjdDoxMDAw
:uid28	:robotics	:UHJvamVjdDoxMDAw
:uid18	:robotics	:UHJvamVjdDoxMDAw
:uid16	:robotics	:UHJvamVjdDoxMDAw
:uid26	:robotics	:UHJvamVjdDoxMDAw
:uid11	:laser-cutting	:UHJvamVjdDoxMDAw

Figure 54: Table 3 query results

3.4 Summary and outlook

An ontology to connect OSH project data was developed and added with a second ontology defining concepts of skills. The current development aim was to create a network that is able to give possibilities on assigning skills and interests to contributor profiles, tasks and projects and make them visible for queries. For this user stories were derived to guide the development in alignment with the implementation partner WIF and the other work packages project partners of OPEN_NEXT. The skills for the ontology were identified from WIF project data based on terms used in the ESCO hierarchy for skills. Based on this WIF was able to identify set of skills in a sample of project data. This data is connected over the semantic network and can be queried according to the introduced user flows. The OSH ontology’s concepts were added with mappings and a mapping code for JSON inputs is provided for instantiation. The mappings are read from the ontology and the relating JSON data are instantiated as the respective concept. After that reasoner inferences are asserted to the instances and saved as a separate ontology. This ontology serves for the queries for project-user skill matchmaking.

Building on this development, there are next steps aimed for the project goal and possible further follow-up activities after the project work. These are sketched out in bullet points at the end to record them for the future.

The demonstrator will be connected to the Wikifactory platform in a next step and validated through user interaction. For this it is envisioned to firstly provide users with a possibility to assign skills and interests to their profile by themselves indicating to the community and getting suggestions on this base. For a later use of the implementation, the now initial draft of skillset should be enriched and extended. The current skills provided are a first draft based on a sample Wikifactory project with terms of the ESCO hierarchy of skills, a European standard for skills, competences and occupations. How it helped identifying skills is explained more in detail in the annex section 3.1.2. Skill extraction and identification algorithm based on the data have to be comprehensively studied which, however, may leave the realm of ontology development.

In terms of the semantic network, a more comprehensive study on the relationship of individual concepts will help to formalize them for automatic assignment, either to connect individuals or to classify them as concepts.

Based on the aimed user validation it is being targeted to extend the user stories in the next step. On the one hand, this means expanding the current application case, but also finding further cases in the OSH landscape of the developed network.

Based on the current results, the following measures could be of interest as an extension of OPEN_NEXT.

- Automate the instantiation process for a more dynamic and efficient workflow. Currently, the instantiated project data is queried from the WIF GraphQL API and intermediately saved in JSON format. The data is then instantiated with the help of the demonstrator mapping function and provided as semantic files, which can be queried by end application. During the user interaction on a platform it would be better, if the instantiation and query takes places based on the user interaction. This prevents handling of redundant data that would slow down the query-answer processing. For example, if a user searches for possible interesting tasks in projects, there would be no need to also instantiate data from other users and their skills. For this, two steps are recommended:
 - a) Contextualization of queries based on the user interaction
 - b) The second step is about how to handle the contextualized queries. For this some kind of semantic enrichment of the existing scheme is necessary, either a direct integration or build upon that. Such a scheme would serve as a link between the GraphQL (or any other) scheme and the ontology. With the GraphQL query, the information would be passed

which semantic concepts are addressed in the query and must be instantiated accordingly. The mappings now are dependent on the structure of the GraphQL query and have to be adapted if the queries change. In order to keep the user operable, a complete instantiation of the project data is foreseen for the current approach. This approach would require a query rewriting module from SPARQL to GraphQL and vice versa, according to the semantic enrichment.

- Connecting the semantic network to the other OPEN_NEXT work packages like OSH metadata. That would require a study of existing interfacing concepts and certainly would need an adaption of the semantic network in some kind.
- Apply a Product lifecycle (PLC) approach on the skill data:
This approach aims to cluster the identified skills into some kind of PLC. It is difficult to define a PLC in OSH projects because of the varying methods of operation. Nevertheless, it would be interesting to classify the skills into higher-level activities if these activities can be linked to certain development phases, e. g. defining a skillset of an Ideation class. This general class would define, that there is some ideation in a development process, but it does not restrict when this activity occurs during development (like it would be in a classical PLC defined via phases). With this approach skillsets for similar projects could be suggested to project teams for consideration.

4. Collaborative Production

Collaborative production consists of two main stakeholders, those who want to manufacture and those who can manufacture. The one who wants to manufacture can be a company or a community member who is interested in manufacturing an OSH product. They would like to find a manufacturer locally for the product. This usually involves multiple steps such as searching for suitable manufacturers, exchanging emails, models, discussing pricing and finalizing the order. To make this process easier there are three steps that were identified and further developed. The first step was to identify production metadata which simplifies the exchange of information between the two stakeholders. This ensures that the important data is documented for anyone who wants to manufacture an OSH product. The second step, for those who want to manufacture, they can easily find production metadata in OSH documentation through machine learning. The third step, is to matchmake the two stakeholders, based on the production metadata. These three solutions are further detailed in the following sections.

4.1. Identifying production metadata

Company or community member, willing to replicate an OSH product they found on the Internet, need to assess if this product is manufacturable (at reasonable costs) with production tools available for them. This is a complex question and subject to the decision of experts skilled in the corresponding field of technology. The decision requires a deep study of the technical documentation, specifically CAD files. The problem scales with the level of complexity and diversity of technology fields touched by the design. Especially when 3D modelling data is only available in proprietary file formats, even quick analyses can become very time consuming and frustrating, hindering decentralised production and making design reuse too costly.

Standardised production metadata can represent essential properties of hardware components and shall be processable on any platform. Hence, they shall support willing parties in the assessment of manufacturability – independently whether CAD files are provided in proprietary or open formats – and enable further (automated) digital services based on that data. For example, if a part is found to be not manufacturable in a given context, the data shall support the identification of potential external production partners; this matchmaking can be automated. Figure 55 below illustrates this vision.

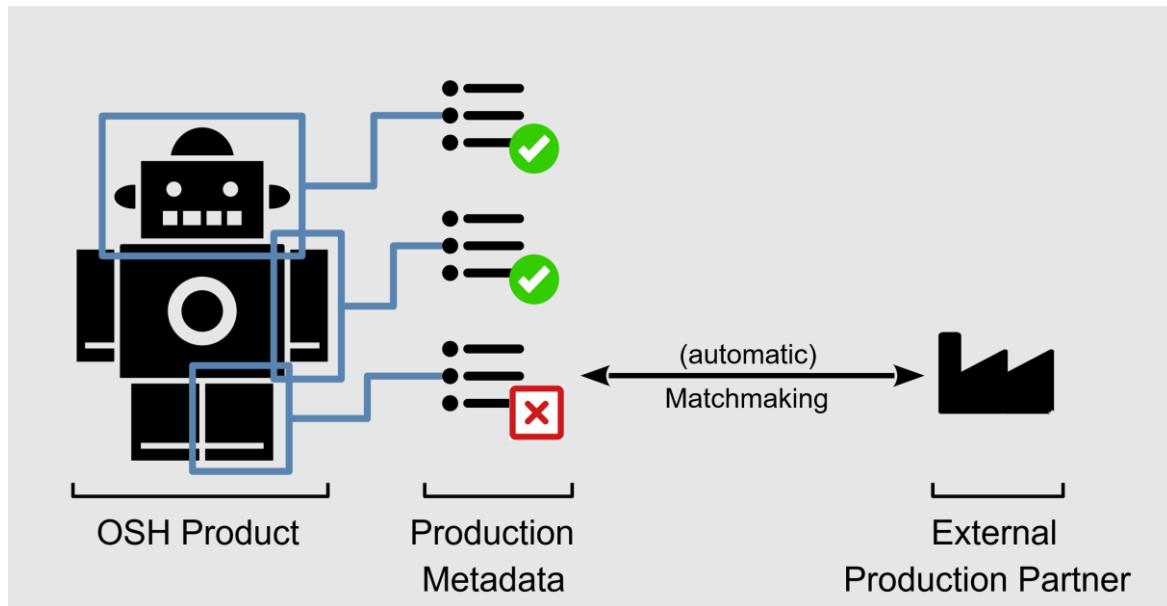


Figure 55: Production metadata per part allowing for a fast assessment of manufacturability and matchmaking with external production partners

Target groups of the specification are:

- user side: users, community and project team members looking for suitable (partial) technical solutions;
- implementation side: project team members, willing to support decentral production and design reuse by the provision of better data and platform owners, aiming to integrate useful information and features for their end users.

Specifically, the perspective of makers was taken for this version of the specification. This shall allow adoption at a low threshold within and beyond the OPEN_NEXT consortium and also integrates well into the ecosystem of existent specifications of the OSH community. Once the specification has proven its practical usefulness, it can be easily extended for further fields of technology and perspectives of other stakeholders (e.g. automotive industry) – independently of whether the represented hardware is open source or proprietary.

Such production metadata will not supersede in-depth analysis of technical documentation, but rather reduce those in-depth analysis to a minimum. This helps specifically in open technology research when comparing various OSH products potentially solving the same problem.

The production metadata specification has been elaborated in a cooperation between experts from the OPEN_NEXT consortium and the OSH community. The result is considered to be a stable draft and was created from the perspective of makers, but does not fulfil the requirements of a field study. For details see the notes on the development method (section 5.1) in the annexure.

The specification for production metadata describes how and in which format essential data shall be provided. As part of the metadata specification of Task 3.3 (of OPEN_NEXT project) the thereby defined production metadata allow for manual and automatic:

- filtering out OSH solutions unsuitable for a certain production environment,
- matchmaking with potential manufacturing partners.

As open source hardware, the specification is modular. Data fields are defined for identified technology fields of interest (e.g. 3D printing, CNC milling or PCB). For each data field a definition of a suitable data format, background information for intuitive understanding and an example entry is given. Definitions follow, as far as possible, official standards (as e.g. ISO 286 for the smallest tolerance class of a CNC-milled part).

When following the specification, production metadata is available in both human-readable form (on OSH platforms e.g. in plain text TOML files) and machine-readable form (in TTL/RDF format and via the Wikibase instance (Task 3.3) after being automatically processed by the crawler). Since it is a fundamental requirement of the specification that all data is published under a free/open license, the database is freely exploitable, also as Linked Open Data, for any use and willing party.

In this context, the user story U6/3: “Identify right partner for collaboration”, is primarily addressed. Also, U14/6 “Support rebuilds” is hereby supported, although not directly addressed.

4.1.1. Usage

The metadata can be used by any platform as a reference to define a template or a guide for its users. This in turn can be followed by companies, project members and collaborators to ensure complete and clear documentation, related to production metadata.

The production metadata identified is also used for developing a Wikibase instance and its metadata specification (part of task 3.3 in the project). The Wikibase instance³³ (a knowledge base) for OSH projects, is being developed, as an extension to another part of the OPEN_NEXT in the Work Package 3. This will host a distributed database with metadata about OSH projects from various OSH platforms. The data will be crawled from the different platforms using a crawler. The system relies on a metadata specification that a) provides the common ground for the data and b) ensures that all data is freely exploitable. Users can then search and filter open source hardware for specific production-related data.

The specification for production metadata has been published under a free/open license on GitHub, both in human-readable form (in Markdown³⁴ and machine-readable form³⁵ (in TTL/RDF, as part of the ontology of T3.3)).

When OSH projects follow the specification, the data will be automatically found by the crawler, uploaded to the free/open database of the Wikibase instance and hence becomes available for others (including other OSH platforms).

End users have contact with this specification by using features on applying OSH platforms which rely on the input from OSH developers willing to follow this specification – e.g. related filters and queries on Wikibase instance or manufacturer matchmaking on Wikifactory. It is not envisioned that end users handle raw data.

³³ <https://www.mediawiki.org/wiki/Wikibase/FAQ>

³⁴ <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OKH-LOSH.md>

³⁵ <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OKH-LOSH.ttl>

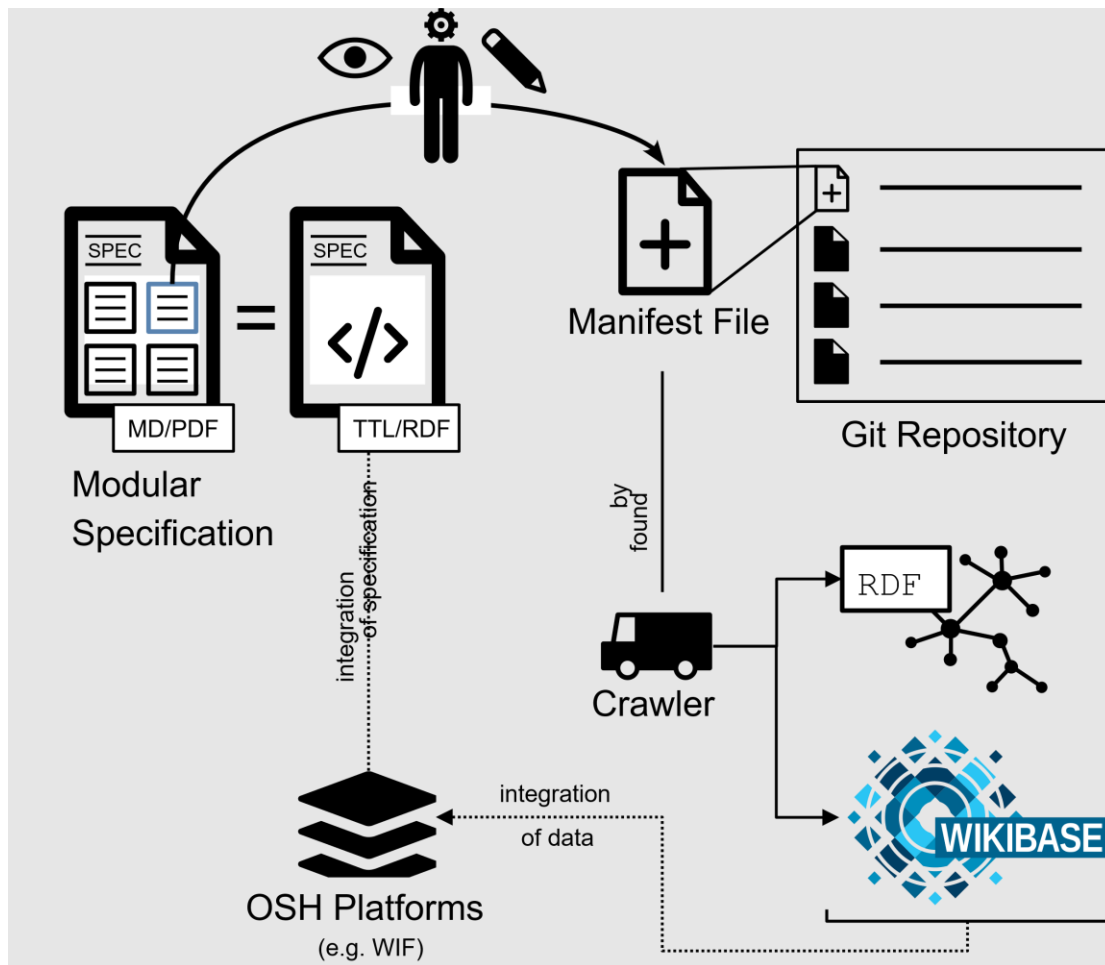


Figure 56: Provision of production metadata by publishing a plain text file („manifest file“) in a git repository

OSH developers can provide production metadata by following the metadata specification for open source hardware (Task 3.3). The specification is accessible for them either in human-readable form on GitHub (platform-independent) or in implemented form on the OSH online platform of choice (platform-specific). Metadata shall be provided for the corresponding fields of technology a specific design is touching (e.g. 3D printing an CNC milling) in the form defined in the specification. In the simplest form, OSH developers fill out a template of a plain text file and publish it in the GitHub repository of the OSH design. However, this may be quicker on other supported platforms (as Wikifactory or Appropedia) depending on how those platforms implemented the metadata specification.

Platform owners can follow the metadata specification for open source hardware (Task 3.3) which includes production metadata. For this, the machine-readable format of the specification (TTL/RDF) may provide the most convenient way to integrate the specification on the platform. Metadata must be provided in the specified formats via an API so that the crawler (T3.4) can access it. Thereby it is irrelevant whether such data has been manually input by users of the platform or automatically generated from algorithms of the platform.

However, it is still possible – although not recommended from the perspective of the project – to exploit specification and data for internal use only and exclude the crawler.

As the first applying platform owner (apart from the Wikibase instance itself), WIF decided to implement the specification in the Wikifactory platform to provide users with an automatic selection of potential (local) manufacturers for a given hardware solution, including all of its parts.

However, no usable demonstrator neither for the Wikibase instance nor for the implementation in the Wikifactory platform could be realised until the submission of this report. A presentable demonstrator of the Wikibase instance will be published with Deliverable 3.3.

4.1.2. Step-by-step example

Example: Clamp Ring of OHLOOM v1.0.0, a relatively simple, 3D-printable part as in Figure 57.

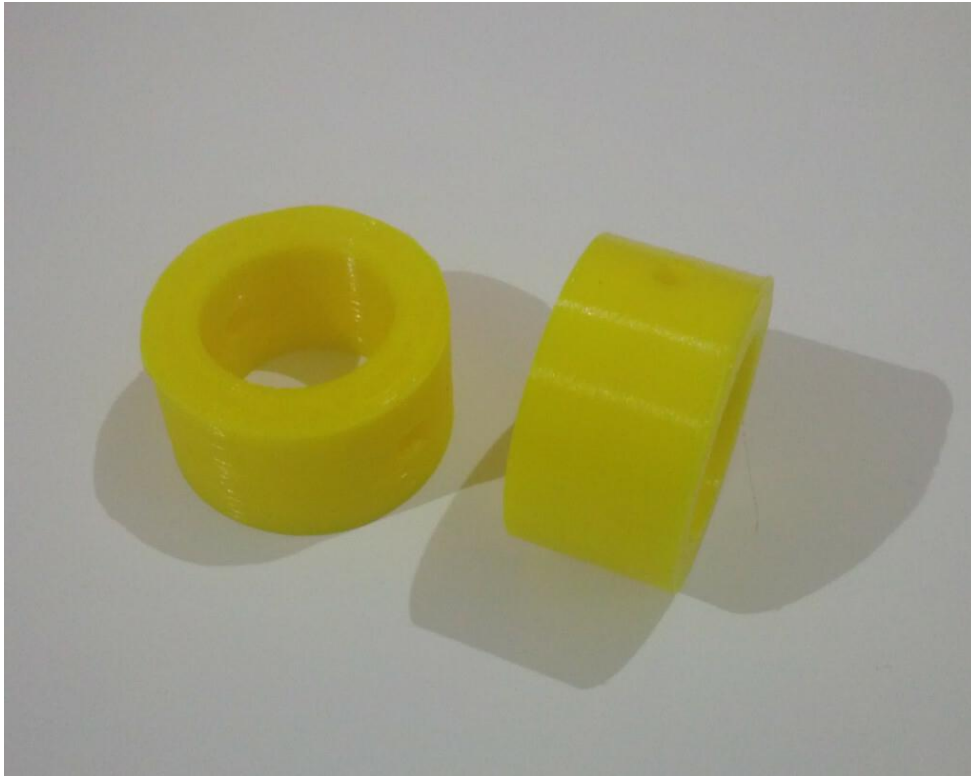


Figure 57: A picture of the clamp ring

A project team member, following the metadata specification for the use in a GitHub repository:

1. looks up the section for 3D-printing in the metadata specification: <>;
2. fills those metadata fields in the plain text file for metadata (so called “manifest file”);
3. saves (“commits”) the changes in the GitHub repository;
4. data is now accessible for anyone;
5. once the crawler found the file, data will be additionally accessible as Linked Open Data (RDF) and via the Wikibase instance

```
[[part]]
name = "Clamp Ring"
image = "/Documentation/Assembly_Guide/Parts_Print_2.jpg"
tsdc = "3DP"
source = "/3DParts/ClampRing/ClampRing.scad"
export = [
  "/3DParts/ClampRing/ClampRing.pdf",
  "/3DParts/ClampRing/ClampRing.stl"
]
printing-process = "FDM"
material = "ABS"
outer-dimensions-mm = "cylinder(h=30, r=28)"
mass-g = 30
infill = 30
raft-brim = true
supports = false
resolution-mm = 0.2
shell-thickness = 3
top-bottom-thickness = 3
```

Figure 58: Resulting production metadata for Clamp Ring

4.1.3. Summary & Outlook

In pre-validation workshops with pilots from the OPEN_NEXT consortium, workshop attendees found it exhausting to fill out plain text templates to follow the metadata specification (although features based on standardised production metadata generally have been found very useful) – developers deeply committed to open source values were the only exception. This was the case specifically for OSH projects that already had mature technical documentation.

To lower the threshold the team developed software using machine learning algorithms for automatic extraction of production metadata from text-based documentation. The software was tested with projects from the Appropedia platform; see section 4.2 for details.

As far as OSH projects provide 3D modelling data in the STEP format (ISO 10303), some production metadata can also be extracted directly from STEP files. However, due to the complexity of the library and technical issues (e.g. failing compilation under Linux systems – the team contributed to bug reports), development of software performing this extraction of metadata from STEP files have been defined to be out of scope for the OPEN_NEXT project. Nonetheless, it has been found to be a reasonable next step to support a wider and easier adoption of the specification.

4.2. Identifying production metadata in documentation through machine learning

While searching for projects to contribute, it can be hard to find the right project that fits the search requirements at first glance, because of the varying styles and structure of OSH projects on various platforms. The collaborators should often read the whole documentation to find out the necessary information.

To tackle this issue, a solution as an example for collaborative production is presented, which identifies selected production metadata in documentation through machine learning. The main aim is to extract common entities within a text or webpages using Named Entity Recognition (NER or NLP) method.

To do this the step was find entities in the documentation. An entity³⁶ can be any kind of data, a single person, place or object, about which can be differentiated from other data. In order to identify these entities, different kind of machine learning algorithms are searched.

Named Entity Recognition (NER) is used in natural language processing to detect the needed entities from text data into pre-defined characteristics. In this solution, the SpaCy library is used to train the deep learning model. The selection of entities was based on the main information in OSH projects on the previous section, through identifying production metadata. Therefore, the selected entities are manufacturing process, machine type, material and dimensions.

Main features are as follows:

- With a web application any kind of text can be given as input, and the output could be received with the classified entities.
- The application can be used for any plain text input or Mediawiki-based websites (while this was only tested on Appropedia)
- The algorithm uses NER of SpaCy to train the model with Deep Learning (NN) for the characteristics mentioned.

The steps for installation the solution from GitHub repository can be found in annexure under section 5.3 Installation Guide. The solution can be used by any community member or a platform owner to automatically find and highlight relevant production information in projects.

4.2.1. Step by step example

1. As a community member or contributor, you are looking for a project with specific process, machine type, material or dimensions, you have an interesting project where you want to find metadata (plain text or Mediawiki based websites).
2. To run the step by step example, the GitHub repository should be cloned and the application should be installed and executed. After the successful execution, the application window in Figure 59 appears (the installation steps can be found in the annex section 5.3)

³⁶ <https://afteracademy.com/blog/what-is-an-entity-entity-type-and-entity-set>

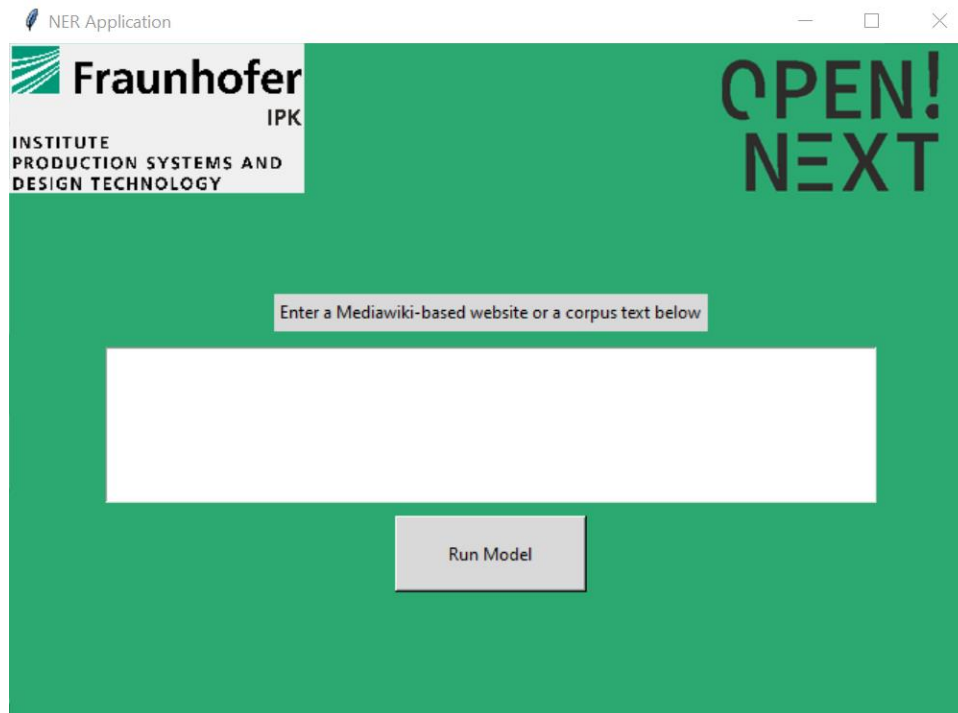


Figure 59: Front end of the production metadata identification application

The application accepts two types of input, text and Mediawiki-based websites. For both types of input as the step by step example is described below.

1. Providing the input as text entry from Wikifactory³⁷:

In this example, we will show you the results of the application from a project in Wikifactory as text input. First step is to copy the project data from the website of your choice and paste it in the application as in Figure 60.

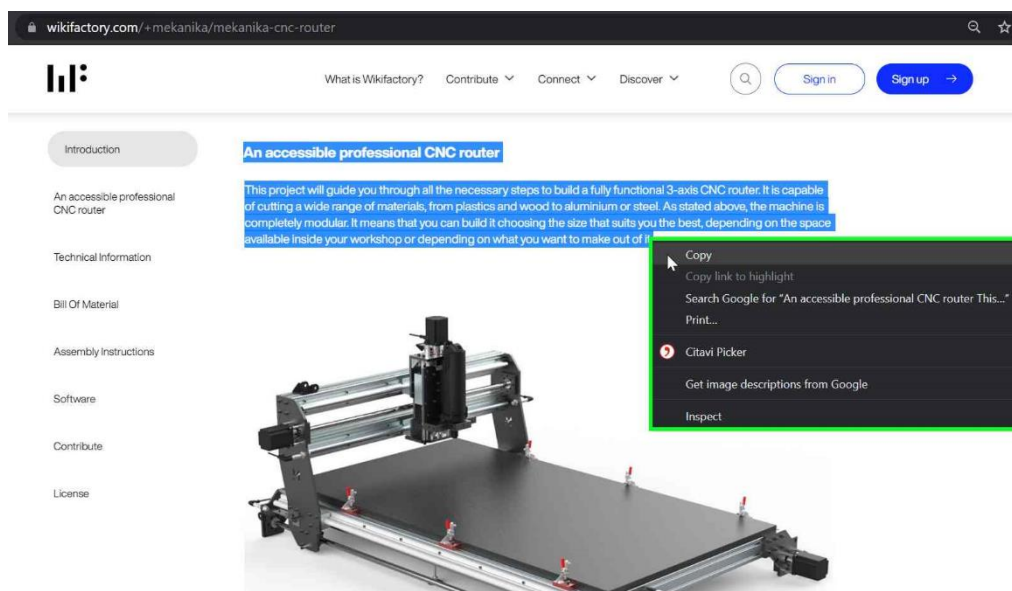


Figure 60: Copying entries from Wikifactory

³⁷ <https://wikifactory.com/+mekanika/mekanika-cnc-router>

After copying the text entry, we will paste it in the application and run the model as shown in Figure 61.

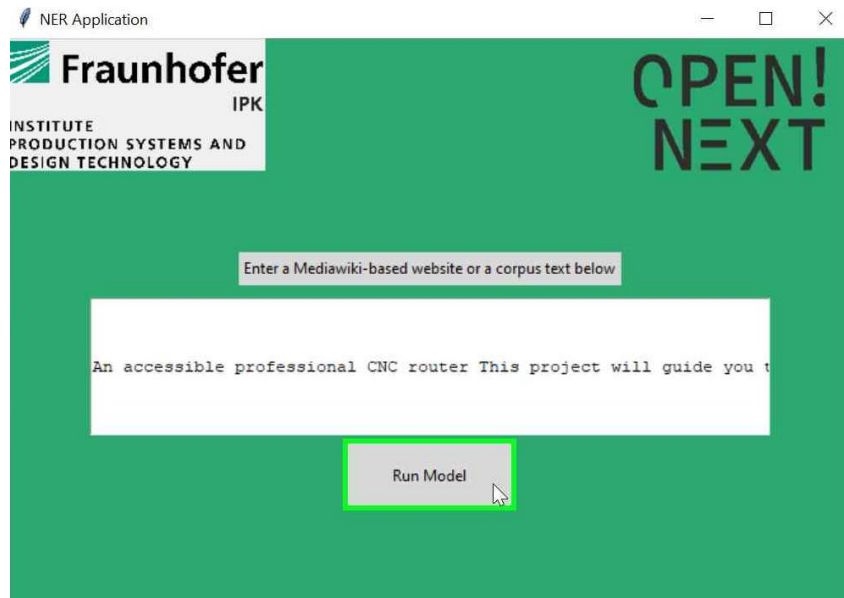


Figure 61: Pasting the entry as text in the application

The results of entry will pop up in HTML form as in Figure 62.

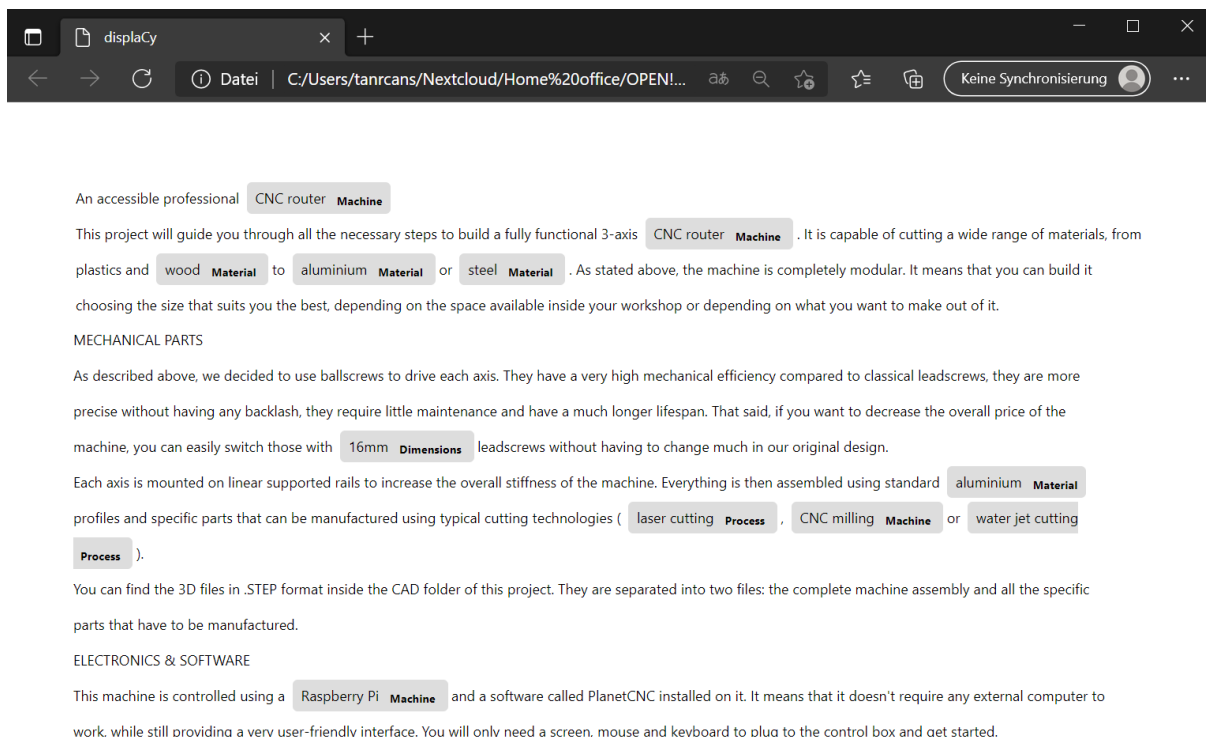


Figure 62: Results of text entry

As seen in the output of the algorithm indicated the process such as laser cutting, water jet cutting; machines as CNC router, materials like aluminium, wood and some dimensions.

2. The example with website: <https://www.appropedia.org/Recyclebot>³⁸

In this example, we will show you the results of the application from a Mediawiki-based website as link. For this purpose, we will use the Appropedia website and select any project we want to evaluate, and copy the URL as in Figure 63.

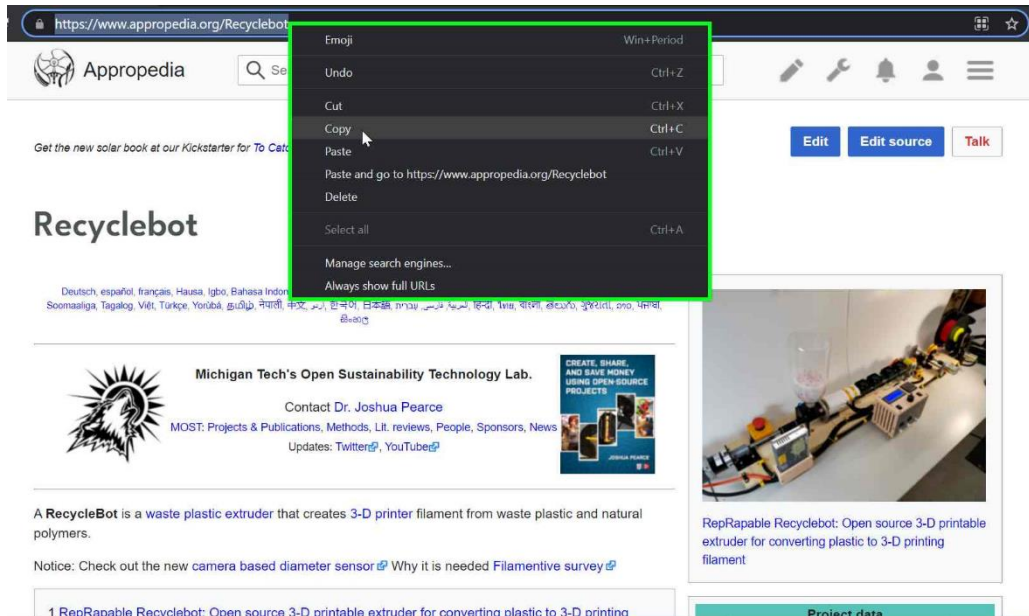


Figure 63: Copying the URL from an Appropedia website

After copying the URL, we will go to our application and paste the URL in the textbox and click on button “Run Model”.

³⁸ <https://www.appropedia.org/Recyclebot>

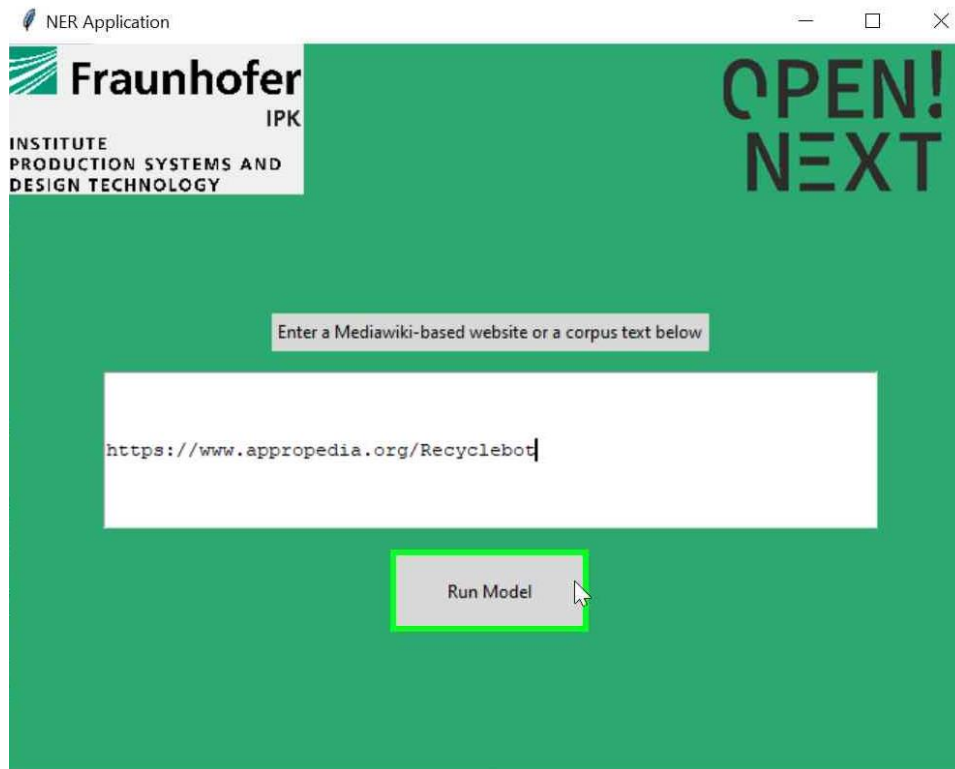


Figure 64: Pasting the URL and running the model

Results of the website are presented in Figure 65 where the algorithm captures all the information from website, and pops up in HTML form.

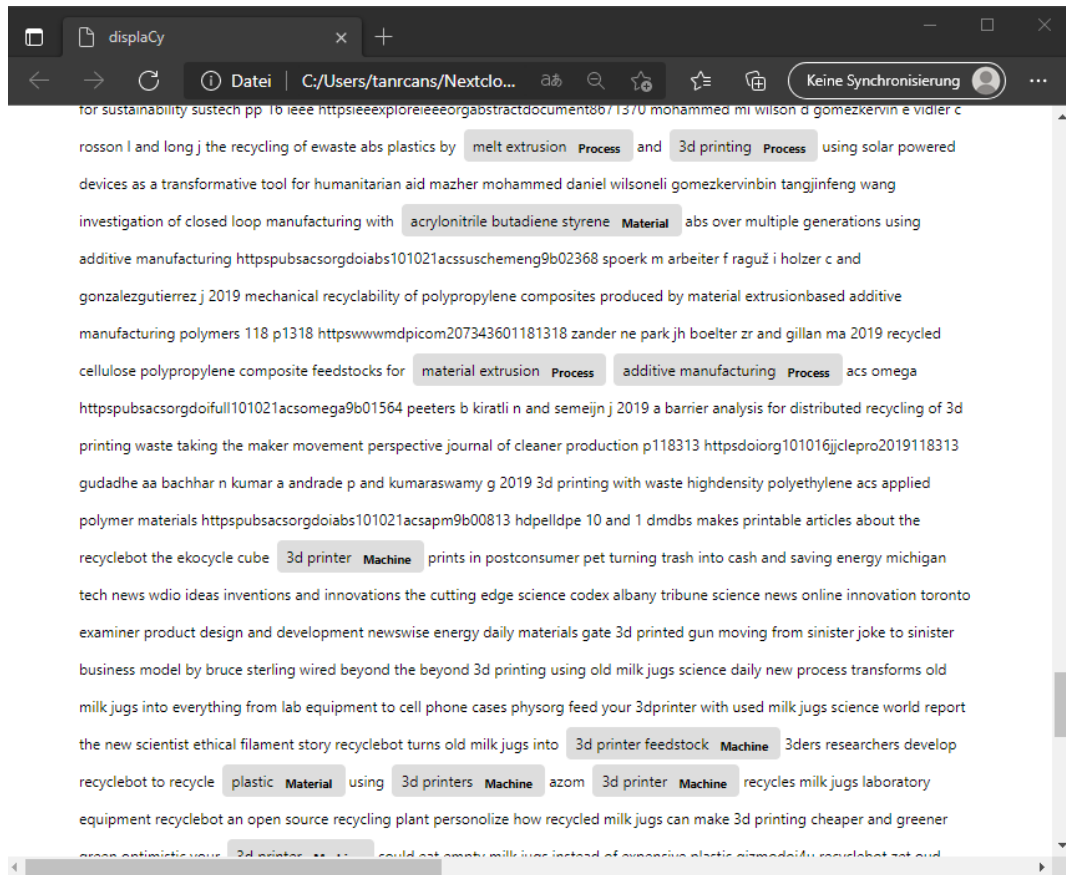


Figure 65: Results of the URL

4.2.2. Summary & Outlook

In this section, the named entity recognition algorithm has been explained and used in an application. This solution uses named entity recognition algorithm for extracting manufacturing process, machine type, material and dimensions of an open source hardware project, within a text or a website. The algorithm can be used in different use cases with corresponding training data.

As a next step, the captured information in each entity can be used for matchmaking in further steps for manufacturing. According to the process, machine type or material, prototyping partners such as Fablabs or Makerspaces with corresponding machine types or materials can be suggested to contributors.

In the first community-based approach to improve discoverability of OSH by representative metadata, namely the [Open Know-How](https://openknowhow.org/)³⁹ initiative, the Appropedia platform with 74% of all included OSH projects by far the most active contributor for OSH metadata (see a list of all projects following the OKH specification [here](https://github.com/OpenKnowHow/okh-search/blob/master/projects_okhs.csv)⁴⁰). Therefore, Appropedia was identified as a promising example case. Furthermore, Appropedia provides technical documentation mainly in form of Mediawiki-based articles, which makes them easily processible for machine learning algorithms.

Currently, Appropedia is manually identifying and specifying the metadata for its projects. They expressed interest to use or integrate the machine learning approach to automate metadata identification. This will also be explored as a next step.

³⁹ <https://openknowhow.org/>

⁴⁰ https://github.com/OpenKnowHow/okh-search/blob/master/projects_okhs.csv

4.3. Collaborative production and matchmaking tools on Wikifactory

A user needs survey and analysis conducted under WP3.1 highlighted two key challenges faced by open-source hardware projects during the physical prototyping and production phase (also highlighted in Section 1.1). First, the process of identifying the right manufacturer for the open-source hardware projects, and product designers in general, is still a challenge. Second, the collaborative process of setting the manufacturing specifications is often slow and error prone, mainly due to lack of appropriate and interoperable software. As the result of both factors, product developers often fall back to email to share the files while the manufacturers communicate the DFM (design for manufacturability) feedback by using annotated screenshots.

The aim of the *Collaborative Production* tools is to overcome those two factors by enabling rapid prototyping of parts and eventually entire products on the Wikifactory platform.

As a first step and by way of solving these communication challenges, the stand-alone CAD file-sharing solution called *CAD Rooms* has been developed, which offers product developers an easy way to view, share and collaborate with anyone, setting aside dedicated applications and requiring only a web-browser. *CAD Rooms* can manage over 30+ CAD file formats and the models they define can be uploaded, visualised, and interrogated. Additionally, the tool offers an assembly explorer, cross section tools, and most importantly, real-time chat for free-form feedback, as well as a 3D annotation tool for leaving precision feedback on parts and geometries.

The second step is to enable the manufacturing of the different parts of this project. In order to do so, the user must select the desired part on the 3D model and click on the “*Make*” button located in the top-right corner of the CAD Room.

Due to the fact that *CAD Rooms* leverages proprietary SDKs (Software Development Kits) for the visualisation of proprietary CAD file formats (such as Autodesk, SolidWorks and Siemens), a stand-alone application that can run without Wikifactory cannot be developed for this solution. Instead, the manufacturer match-making tools will be made accessible via a documented API so that 3rd party applications may interact.

4.3.1. Usage

The first step to open the *CAD Rooms* is to select any compatible file from the Wikifactory project page. Then, the user will see a similar scenario as the one presented in Figure 66:

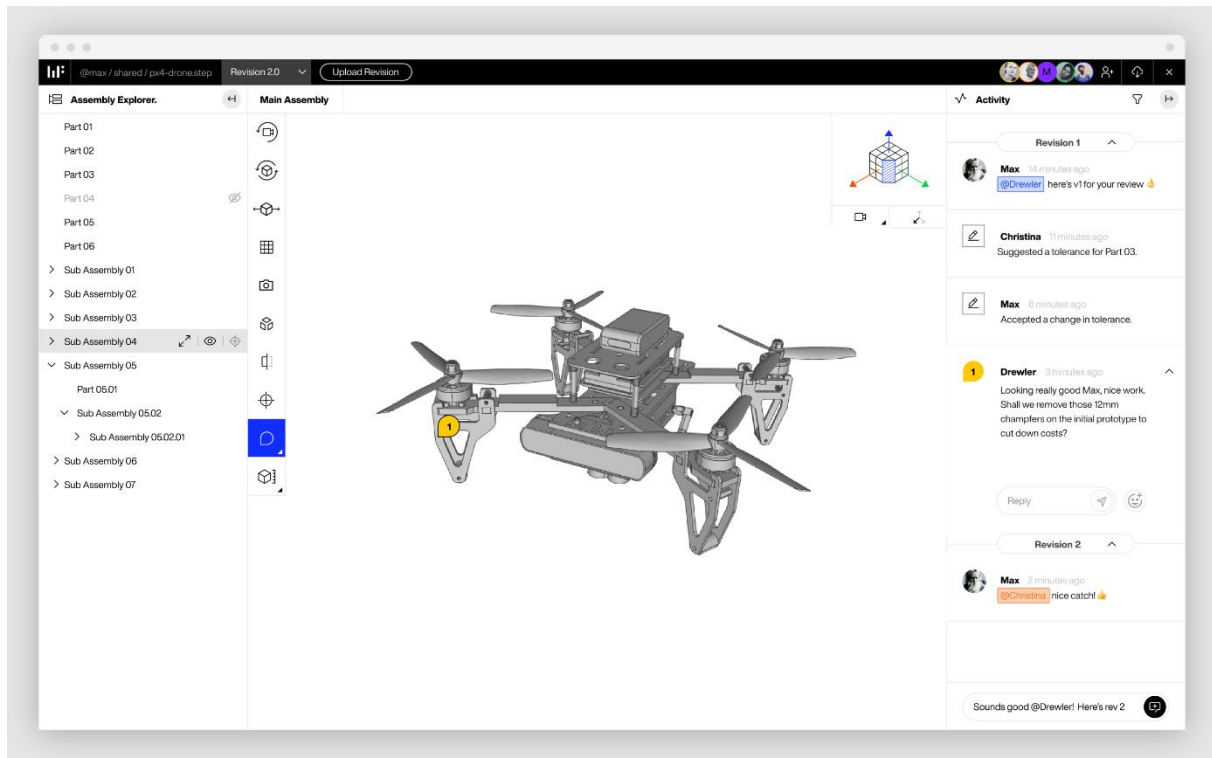


Figure 66: An example of the CAD Rooms tool

The central zone of Figure 66 is occupied by the content of the selected file, in this case a 3D model of a drone. On the left side of the figure appears the *assembly explorer*, which can be used to explore the different parts of the 3D model and see how they are related. On the right side of Figure 66 is the real-time chat. By using the chat, all the collaborators could share ideas about the model. Finally, in the 3D model itself there is a small yellow icon with the number 1 inside this. This icon represents an *annotation*, and they are used to share information specifically attached to some part of the model. For example, the text associated to the annotation can be seen on the real-time chat.

From there, the manufacturing process would begin and its details will be presented in the next section.

4.3.2. Step-by-step example

The user flow for being match-made with manufacturers gets initiated via a new “MAKE” button in the top right corner of CAD Rooms. Once clicked, the product developer can create an RFQ (request for quotation) by completing two steps. First, specifying how the part should be manufactured and second detailing their order. Figure 67 shows the interface for the selected part.

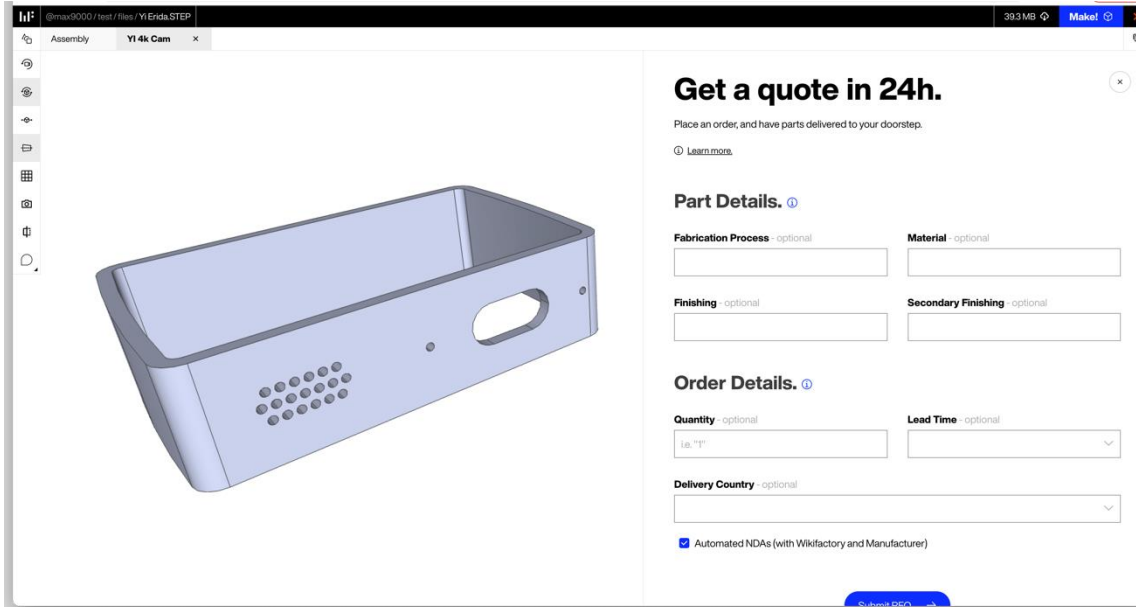
The image shows a web browser window displaying a form for requesting a quote. On the left, there is a 3D model of a rectangular tray with a handle and a grid of holes. The right side of the page contains a form with the following sections: 1. 'Get a quote in 24h.' with a subtext 'Place an order, and have parts delivered to your doorstep.' and a 'Learn more' link. 2. 'Part Details.' section with four input fields: 'Fabrication Process - optional', 'Material - optional', 'Finishing - optional', and 'Secondary Finishing - optional'. 3. 'Order Details.' section with three input fields: 'Quantity - optional' (with '10, "T"' as a placeholder), 'Lead Time - optional' (a dropdown menu), and 'Delivery Country - optional' (a dropdown menu). 4. A checkbox labeled 'Automated NDAs (with Wikifactory and Manufacturer)' which is checked. 5. A blue 'Submit RFQ' button at the bottom right.

Figure 67: Screenshot of the form used to start the manufacturing process

To specify how the part should be manufactured, the user must select a *Fabrication Process* and a *Material* as well as some optional finishing details. Those fields can be found on the right side of Figure 67. With the aim of making this process as easy as possible, it is only required for the user to fulfil the *Fabrication Process* and *Material* fields. Those fields have been implemented as smart filters that only show the relevant information. For example, when the product developer selects “3D Printing > DFM” as the *Fabrication Process* (DFM standing for Design for Manufacturing), only those materials compatible with DFM will be shown in the *Materials* dropdown.

With regards to detailing the order, it is only required to specify how many parts are required, how fast they are needed, and finally the delivery country.

Finally, the latest step consists of submitting the RFQ, using the button located below the form.

Once the RFQ request has been submitted, the platform shows matching manufacturers as well as notifies the project team to review the RFQ. Finally, they are invited to submit a final quote for the product developer. From this point, the product developer has the chance to review the *Quote* and optionally chat with the Manufacturer using the *CAD Rooms*, before placing an order, and having the parts delivered to their doorstep. For example, Figure 68 illustrates how the users see the list of manufacturers that could take the order as well as some details such as their location or the number of already delivered orders. Then, the user must choose one of them to start the process and click on the *Submit RFQ* button located on the bottom side of the Figure 68.

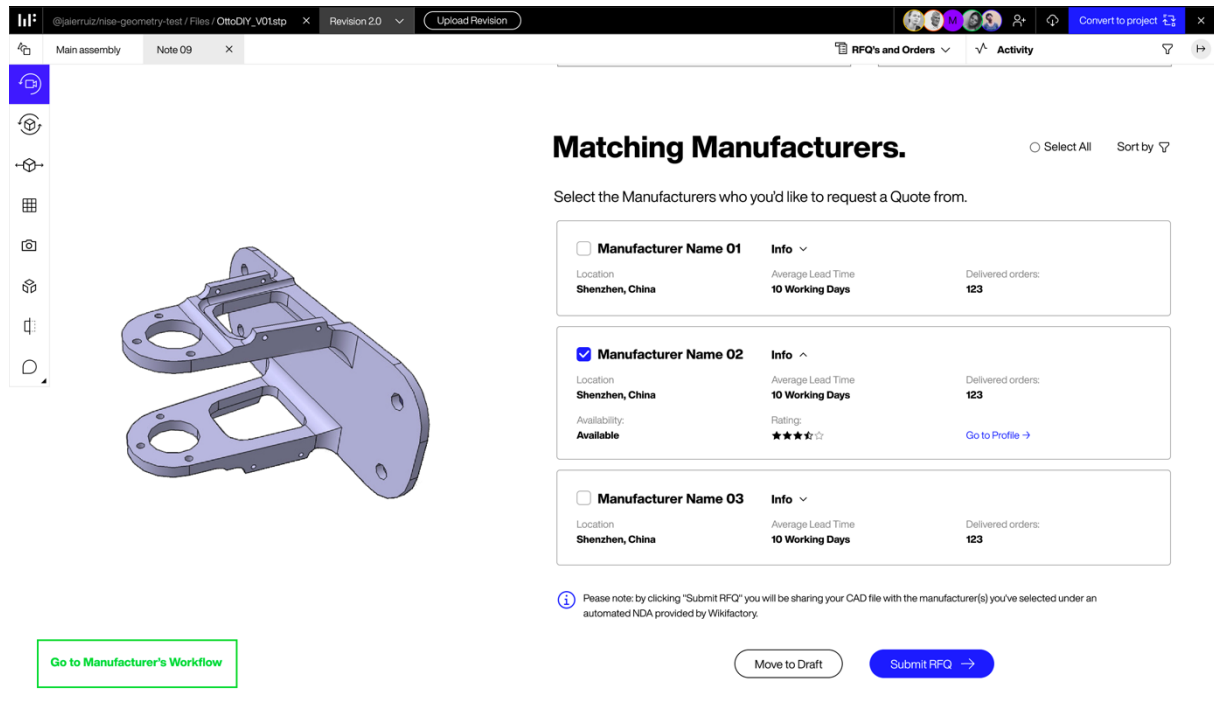


Figure 68 Inspecting the manufacturers that could make the selected part

4.3.3. Summary and outlook

The presented *CAD Rooms* tool acts as a bridge that connects product developers and manufacturers. By using the real-time chat and the 3D annotations, all the information for the manufacturing process can be grouped together, and there is no longer a need to keep a long chain of shared emails with different versions of the models. Now, having this functionality, the match-making user flow itself can be built on top. The user flows have been prototyped, tested with OPEN_NEXT participants, and a first version of the backend APIs have been implemented.

At this moment, a beta of the *share a file* functionality is currently working in Wikifactory. This functionality will speed up the process of manufacturing spare parts, since it is no longer required to create a specific project for the process. Thus, any user could share a link to a part, that could be consumed by a different community member. Not only that, but in the near future Wikifactory will allow the process of manufacturing whole models, not only parts.

Finally, with regards to the process of detailing the RFQ order, a set of smart filters has been mentioned before. These smart filters can be extended to other options to better detail the process and offer all the possibilities offered by the manufacturers. This initial set of filters will be expanded after releasing the first version of the full flow. For example, when the user selects the *3D Printing* process, Wikifactory will dynamically add related fields such *Layer height* or *Infill*.

5. Documentation and Guidelines

Product development is often complex; In OSH product development there are additional layers of tasks. OSH products and projects need to be documented so that they are accessible to and understood by a diverse group of interested community and its collaborators. The methodologies used to develop and document also vary depending on the product and the project community. To ease process of documentation a few documentation templates and guidelines were developed. These can be categorized into 4 main categories namely: project/product overview template, technology readiness levels for OSH, Licencing OSH guide, integrating templates in WIF. Each of these are further detailed in the following sections. The target groups here are community members and companies working with OSH projects.

5.1. Project/Product overview template

It is often recommended to document continuously during product development, however when an interested community member wants to get an idea about the project itself, it can be hard to find an overview underneath the complete project documentation. Hence, an up to date project overview improves the visibility and readability of the project to the community. Hence, inspired by a project template⁴¹ on Wikifactory, a new version⁴² of the same was developed to cater to OSH and C3.

The new template is in the form of a Markdown document so that it can easily be copied/duplicated and uploaded onto any online platform. The template has the following sections:

- About your project
- Current status
- The problem
- Product features/ functions
- Project viability
- The team
- The Bill of materials
- The manufacturing/production/assembly
- About the design
- Electronics and programming
- About the prototype
- Potential improvements

⁴¹ <https://wikifactory.com/+wikifactory/project-example-template/file/README.md>

⁴² [https://github.com/OPEN-NEXT/WP3-Documentation-Guidelines-for-OSH-](https://github.com/OPEN-NEXT/WP3-Documentation-Guidelines-for-OSH-Projects/blob/main/Documentation%20%26%20Guidelines/Project%20overview%20documentation%20Template.md)

[Projects/blob/main/Documentation%20%26%20Guidelines/Project%20overview%20documentation%20Template.md](https://github.com/OPEN-NEXT/WP3-Documentation-Guidelines-for-OSH-Projects/blob/main/Documentation%20%26%20Guidelines/Project%20overview%20documentation%20Template.md)

Each of the above sections, are presented with a set of questions, which helps in documenting essential information by answering them. A snippet of the same can be seen in the Figure 69. Further deep links are also provided to learn about certain topics such as about the licenses, technology readiness levels as shown in the figure under sections about your project and current status respectively.

ABOUT YOUR PROJECT

- What is the general idea of your project?
- For whom is your project designed or made for?
- What prior work inspires you to develop your project?
- What is your goal?
- Don't forget to select the license suitable for your project as well. A suggested read can be found [here](#).

CURRENT STATUS

- What is the current stage of your project?
 - use [OTRL](#) for the state of your technology
 - and [ODRL](#) for the state of your documentation
- What are the next steps?
- How can the community help? - provide links to the issue board/any suitable page.
- Who are you looking for? In terms of skills needed

THE PROBLEM

- What is the problem or need that your Hardware or Product Design is for?
- How is it better than the alternative? Is it cheaper, more accessible, more appropriately designed?
- What is the unique value proposition?

Figure 69: Snippet of project overview template

5.2. Technology-readiness levels for OSH

For effective design reuse and practical application of locally produced open source hardware, prior maturity assessment of the technical solution is needed. As the assessment of manufacturability (see chapter 4.1), this is a complex task and requires an in-depth study of the documentation, which specifically becomes time-consuming and frustrating in the case of complex hardware, poor documentation and proprietary file formats. Likewise, it is hard to search and compare OSH products for their level of maturity.

Since project team members already know the OSH product and its technical documentation in depth, a classification of its maturity is a fairly easy and quick task, compared to the efforts a user, new to the product, would need to take. The technology-readiness levels for OSH shall provide an intuitive, common language for users of the OSH product (as illustrated in Figure 70). Thus, the model shall also be available as Linked Open Data.

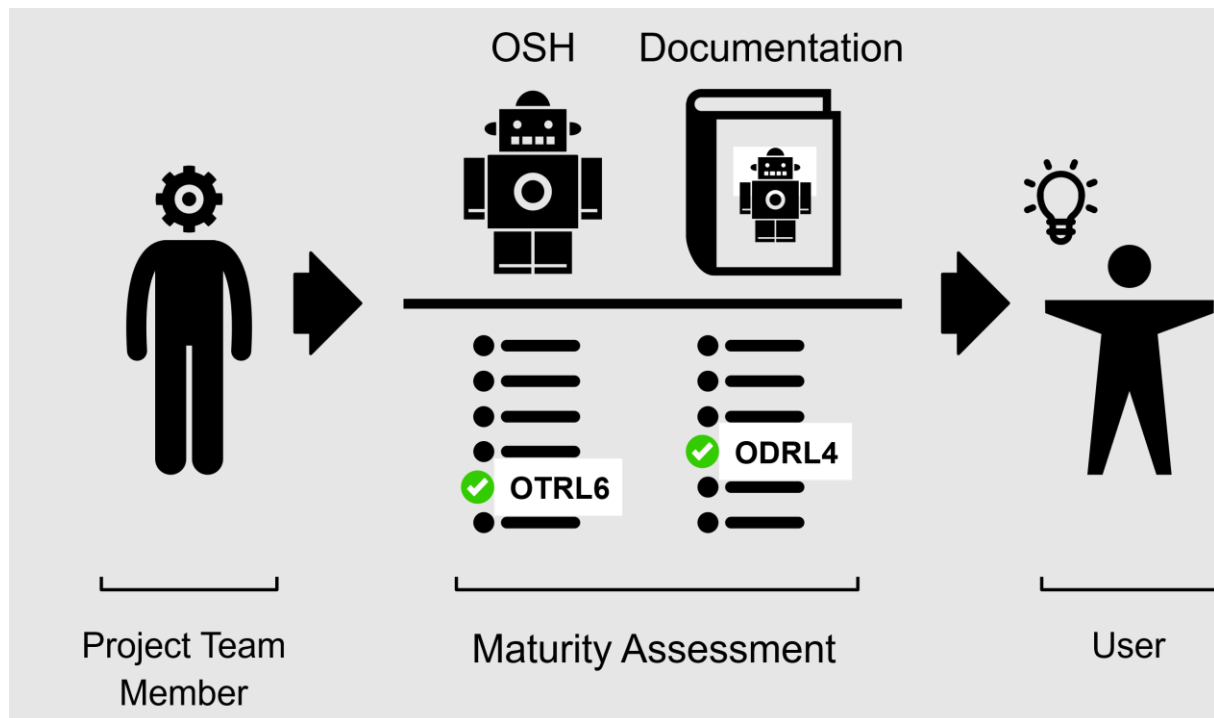


Figure 70: Providing users with a fast assessment of maturity of technology & documentation for a certain OSH product

Following the decentralized nature of open source product development, classification of OTRL and ODRL shall be realizable for any project team member skilled in the corresponding field of technology. Hence, the maturity levels must be formulated in intuitive manner so that project team members can use the model without further training.

In this context, the user stories (D3.1): U7/1: “Identify the status of a project” and U8/1: “Differentiate the level of professionalism are addressed here”. For further details please see the development methods and design notes (section 9.1) in the annexure.

Main Features:

As technical solutions in OSH become accessible by their documentation (and their licenses) the current draft includes two scales: The Open Technology Readiness Levels (OTRL), classifying the maturity of the technology, and the Open Documentation Readiness Levels (ODRL), classifying the maturity of the documentation.

The OTRL are based on the [EU-H2020 Annex G](#)⁴³, hence come with 9 levels; the ODRL are based on the requirements of (DIN SPEC 3105-1) and findings of T2.3 and consist of 5 levels (plus Level 0 when documentation is unavailable or not released under a free/open license).

Since technology- and (Ding et al. 2009) documentation-readiness of an OSH product is an essential information for design reuse, the concept has been integrated into the metadata specification and ontology of T3.3. Likewise, OTRL and ODRL are available in human-readable form (as a Markdown document) and machine-readable form (as TTL/RDF file) under a free/open license on GitHub.

⁴³ https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf

5.2.1. Usage

The current release of the classification model has been published under a free/open license on GitHub, both in human-readable form (in [Markdown](#)⁴⁴ and [machine-readable form](#)⁴⁵ (in TTL/RDF, as part of the ontology of Task 3.3)).

Since primarily developed for the metadata specification (T3.3) project team members should use the model to fill out corresponding metadata fields so that the information becomes automatically available via the Wikibase instance. However, project team members can also use the model independently. The same applies for platform owners.

OTRL and ODRL levels are represented by an identifier. A project team member looks up the best matching levels in the tables of the model (see Table 4 and Table 5 below) for a certain OSH product and uses their identifiers to document his/her assessment. In the simplest case, those identifiers are put in the plain text file for metadata (the “manifest file”) and published in the GitHub repository of the project. The file is then automatically found and processed by the crawler (D3.4), processed data is uploaded automatically to the Wikibase instance (D3.3).

The column “Detail” outlines the project or documentation stage of the corresponding level, the column “Characteristic” provides intuitive indicators for each level. Each level is described in more detail in the Markdown document of the model, in case more details are needed. The first column contains the ID of the level, which is equal to its key in the ontology (T3.3).

Table 4: Technology Readiness Levels for the context of OSH (OTRL)

OTRL ID	Detail	Characteristic
OTRL1	basic principles observed	rough idea
OTRL2	technology concept formulated	concept ready
OTRL3	experimental proof of concept	concept is validated in tests
OTRL4	technology validated in lab	early prototype
OTRL5	technology validated in relevant environment	mature prototype
OTRL6	technology demonstrated in relevant environment	ready-to-use product
OTRL7	system prototype demonstration in operational environment	ready-to-use product (for critical or complex applications)
OTRL8	system complete and qualified	finished product (could be sold in the EU)
OTRL9	actual system proven in operational environment	established product

⁴⁴ <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OTRL.md>

⁴⁵ <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OTRL.ttl>

Table 5: Documentation Readiness Levels for the context of OSH (ODRL)

ODRL ID	Detail	Characteristic
ODRL0	no documentation available	or noncompliant license
ODRL1	basic information available	know that the OSH is, but not more
ODRL2	drafty documentation	documentation in progress, yet patchy
ODRL3	early release	lots of reconciliation required to build & operate the OSH
ODRL4	relevant release	few reconciliations required to build & operate the OSH
ODRL5	mature release	no reconciliation required to build & operate the OSH

5.2.2. Step-by-step example

Practical case: [OHLOOM v.1.0.0](#)⁴⁶

The Open Hardware Loom (OHLOOM, version 1.0.0 shown in Figure 71) is a desktop smallware loom mostly made of wood, 3D-printed parts and standard components, published by Open Source Ecology Germany e.V. (non-profit) in 2021.

⁴⁶ <https://gitlab.com/OSERGermany/ohloom/>

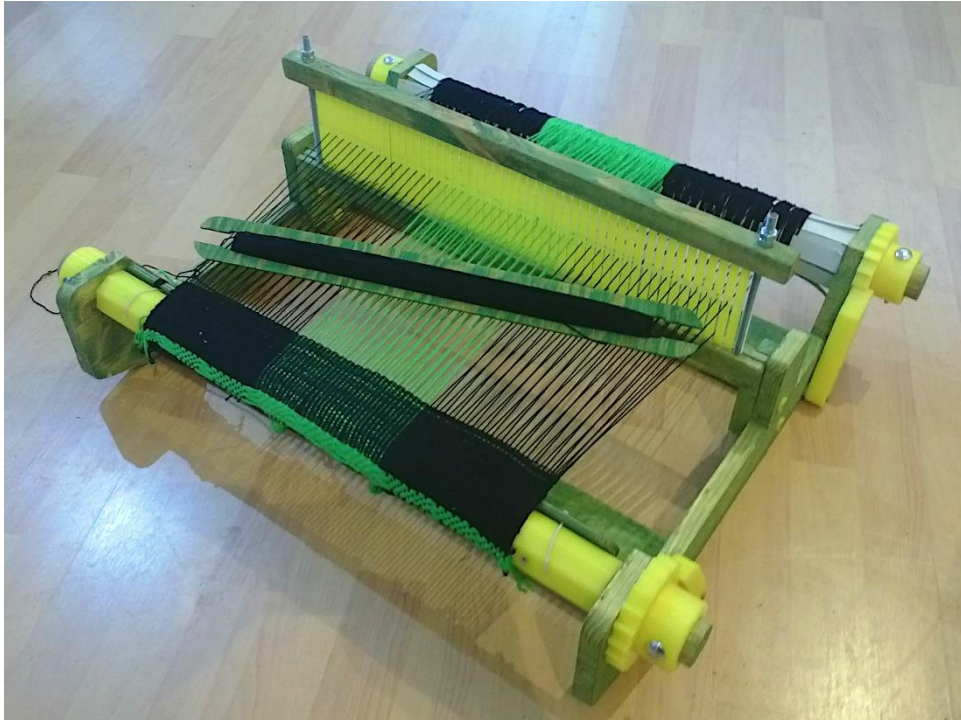


Figure 71: View on an assembled OHLOOM v1.0.0 in use

1. A project team member of the OHLOOM project reads the markdown document of the classification model (accessible here⁴⁷)
2. For OTRL: The OHLOOM has been built and run in the envisioned environment, but not yet by users skilled in the field of technology. It has room for improvement and was not deemed as a final version. Hence it is considered a “mature prototype” but not a “ready-to-use product”.
3. For ODRL: The OHLOOM has been replicated by independent users with no contact with the project team. Furthermore, external contributors have developed the documentation further without reconciliation with the project team. The target group clearly acts autonomously, hence the documentation reached a “mature release” and qualifies as Open Source Hardware according to (DIN SPEC 3105-1). In fact, the project is currently reviewed by the OSH community to be officially attested following (DIN SPEC 3105-2).
4. The resulting inputs for the plain text metadata file (the “manifest file”) in the GitLab repository are:
 - technology-readiness-level = “OTLR5”
 - documentation-readiness-level = “ODLR5”

5.2.3. Summary & Outlook

OTRL and ODRL can provide users very quickly with essential information about the maturity of the design and its documentation so that time consuming study of the technical documentation will not be necessary in most cases when selecting OSH products by their maturity.

⁴⁷ <https://github.com/OPEN-NEXT/OKH-LOSH/blob/master/OTRL.md>

In collaboration with the OSH community, the team continues optimizing the classification system, primarily to facilitate its wide adoption.

WIF expressed interest to integrate the last release of the classification model to provide users with better filter options when exploring OSH on the Wikifactory platform.

Since primarily developed for the Wikibase instance (T3.3), OTRL and ODRL are part of the instance’s ontology and form a crucial element of the metadata specification (D3.3). By the release of D3.3 project team members will be able to classify project stages after auditable criteria and make this information available for others e.g. via the Wikibase instance.

The team collaborates with the community around DIN SPEC 3105-1 and the Open Know-How Specification. By the final release, the team will contribute its findings to those standardisation efforts in order to support a wider dissemination of the model.

5.3. Licensing open source hardware guide (TL;DR)

Many OSH projects struggle with questions around legal issues. In particular, intellectual property rights (IPR), especially copyright, should be basic knowledge when handling with OSH since it is a free/open license that grants anyone with the 4 rights of open source; the rights to study, modify, make and distribute (DIN SPEC 3105-1). The OSH community still struggles with projects using no or inappropriate licenses for their hardware designs.

Licensing can be made relatively simple when essentials about IPR are known. Instead of relying on legal consulting or extensive guidelines, a short, intuitive document could help communicating those essentials to recipients that would not access this information otherwise. The “tl;dr | IPR” document addresses this gap (as illustrated in Figure 72).

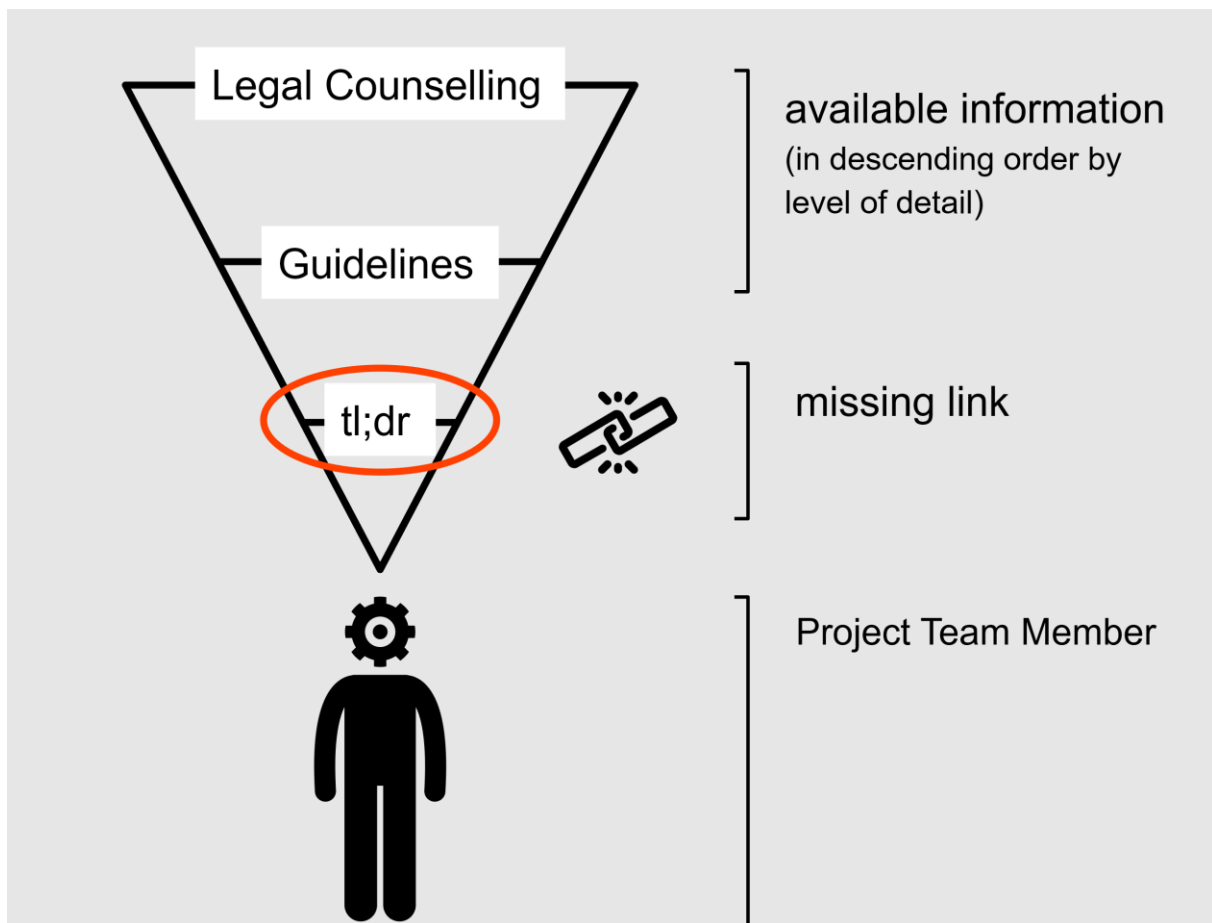


Figure 72: Low-threshold documents (e.g. in form of a tl;dr) may provide the missing link to connect project team members with essential (legal) information

In this context, the user stories (D3.1): U12/11 and U14/5 are addressed here.

For further details please see the notes on the development method (section 10.1) in the annexure.

Main Features: The document relies on the “[OSH Guideline | Legal Issues](#)”⁴⁸ distributed by Open Source Ecology Germany e.V. (non-profit). Following the [tl;dr concept](#)⁴⁹, it condenses essentials about

⁴⁸ <https://gitlab.com/OSGermany/osh-guideline-legal-issues>

⁴⁹ https://gitlab.com/OSGermany/ohloom/-/raw/master/Documentation/User_Guide/User_Guide.jpg

IPR relevant for the use and development of OSH on one page using less than 500 words and pictograms. It furthermore gives license recommendations for most common use cases so that often no further research is needed to apply the new knowledge and license an OSH product with an appropriate license.

5.3.1. Usage

The document consists of two parts:

- page 1 for IP law essentials,
- page 2 for licensing OSH.

Page 1 aims to provide recipients with the minimum necessary overview in IP law and outlines the necessity and usefulness of an appropriate open source license. Page 2 builds upon that knowledge, explains the most relevant licensing schemes and gives concrete recommendations.

The document is primarily distributed in digital form via the GitHub repository with PDF exports for each release, which makes unambiguous referencing in online discussions easy. The design is also feasible for print e.g. as posters in makerspaces.

5.3.2. Summary & Outlook

The document received overall positive feedback from the OPEN_NEXT consortium; however, room for improvement has been pointed out – this has been documented and will be integrated by the team. Once done, the final document will be integrated into Open Source Ecology Germany’s GitLab repository in order to contribute back to their efforts and so the document will be further distributed and maintained by their community.

For WP4/5 workshops with the pilots on legal issues have been planned; this document will serve as essential material for them. For the workshop series, it is also considered to create more documents of this style for other fields of interest of pilots and makerspaces so that modular material can circulate beyond the OPEN_NEXT project.

The collaboration the legal issues working group also led to a rework of license recommendations on the Wikifactory platform.

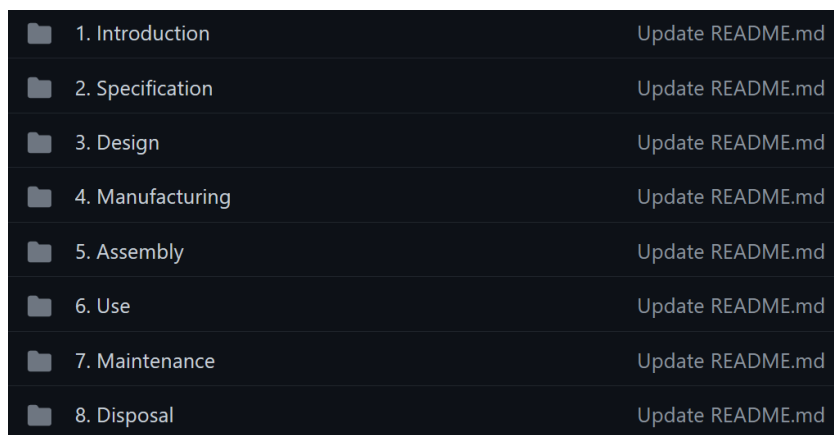
5.4. Integrating templates in Wikifactory

The main objective of this section is to provide the users with an easy way to integrate templates in their open-source hardware projects. Those templates may be focused on different topics and present different structures. Thus, the users will be able to choose the one that better suits their project.

As a use case, we propose the integration of this documentation and guidelines in the Wikifactory platform. Besides that, the documentation guidelines used to illustrate the process are those ones designed by the Grenoble INP, which can be found in this GitHub repository: https://github.com/OPEN-NEXT/wp2.3_Guideline-for-documentation-of-OSH-design-reuse⁵⁰.

5.4.1. Usage

The *Documentation* folder defined within the repository gathers all the different categories to be considered when documenting the project. In particular, the categories are shown in Figure 73.



1. Introduction	Update README.md
2. Specification	Update README.md
3. Design	Update README.md
4. Manufacturing	Update README.md
5. Assembly	Update README.md
6. Use	Update README.md
7. Maintenance	Update README.md
8. Disposal	Update README.md

Figure 73: Documentation categories

As can be seen in Figure 73, the guidelines for the documentation go through all the stages of the project. Starting from a general *introduction* to the project, passing through the *specification* and *design* stages. After that, there are guidelines with regards to the process of *manufacturing* the project, as well as for the *assembly* stages. Finally, there are guidelines to illustrate how to *use* the project as well as how to *maintain* it. There is a last step, associated to the *disposal* step once the project is no longer used. This categorization is still a work in progress, so the order and content of the ones shown may be altered.

5.4.2. Step-by-step example

In order to integrate the guidelines in a new project in the Wikifactory platform, there are two possibilities. On the first one, the user would import the GitHub repository using the import-export functionalities presented in Section 2.1 and by following the instructions listed in Section 2.2.

Once imported, the templates and guidelines will be available for the user in the Wikifactory platform, as Figure 74 depicts:

⁵⁰ https://github.com/OPEN-NEXT/wp2.3_Guideline-for-documentation-of-OSH-design-reuse

Drive (latest)

[New folder](#) [Upload file](#)







Name	Last Contribution	Last Modified / Actions
 Case studies	Import files (#bbbd31d)	6 minutes ago
 Documentation	Import files (#bbbd31d)	6 minutes ago
 Metadata	Import files (#bbbd31d)	6 minutes ago
 Sources	Import files (#bbbd31d)	6 minutes ago
 LICENSE	Import files (#bbbd31d)	6 minutes ago
 README.md	Import files (#bbbd31d)	6 minutes ago

Figure 74 Screenshot of the documentation and guidelines imported in a Wikifactory project

As shown in Figure 74, there is number of pre-defined folders and documents that the user can follow to document the project. In particular, the most important one is the Documentation folder, which contains the templates themselves.

The other possibility for integrating the documentation and guidelines in a new project, consists of *forking* a template project already existing in Wikifactory. The only required step facing the user, consists of going to templates project, and click on the *fork* button.

By doing that, the user will have an own copy of the template project, which can be updated and tailored according to the users' own purposes.

Facing the future, there are plans to facilitate even more the integration of the documentation and guidelines in the projects. Thus, when the users create a new project (note that this is a different option that importing the GitHub repository or *forking* the templates project), they will be able to select the desired templates from a set of possible ones. Not only that, but they will be able to adapt them according to the context of the project. Thus, unnecessary guideline components could be removed in this selection step, so they will not be present later in the project.

6. Summary and Outlook

Under the four categories of interoperability, collaborative production, community management and documentation and guidelines, various solutions were developed and presented in this document. The main outcomes of the deliverable are the:

- First stable version of the Import-Export tool, which supports open and flexible exchange of data and files across platforms.
- Development of a concept, ontology and a demonstrator to enable skill based matchmaking on collaborative platforms. This solution supports in enhancing collaborator and project information to enable an improved company-community collaboration
- Development of a project overview template and enabling easy integration of templates for companies on platform such as Wikifactory.
- Development of a guide to assist in selecting of licenses for projects dealing particularly with open source hardware
- Development of technology readiness levels for projects based on the maturity of documentation and project, to enable project team to select the suitable level. In addition, provide information about the maturity to an interested community/ member.
- Development of production metadata, to identify basic information needed for collaboration with producers. This in turn, helps enhancing the completeness of the project documentation when followed.
- Development of an automated production metadata detector using machine learning. This solution shows the potential to find relevant information without having to read the complete documentation. This saves time for collaborators to find interesting projects.
- Development of collaborative production tools on Wikifactory such as CAD rooms and matchmaking tools. The CAD rooms extends their existing 3D viewing tool, to enable seamless annotation and collaboration opportunities. In addition, the matchmaking tool to find interested manufacturers, further enables companies to find each other in the OSH community locally and around the world.

The solutions derived and their respective benefits along the different phases/activities of a project is illustrated in Figure 75.

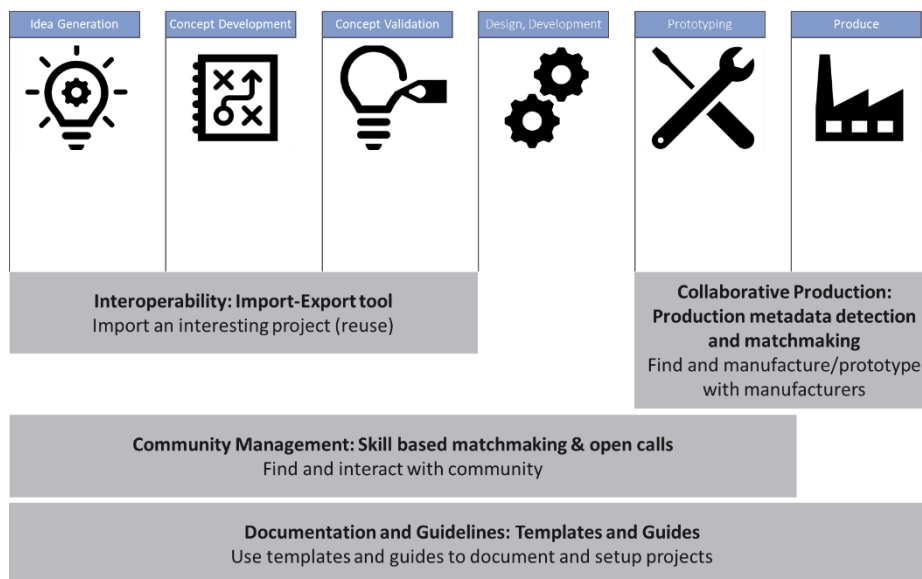


Figure 75: Solutions and their benefits across project phases

Each of the solutions are presented individually but they can also be connected on platforms such as Wikifactory to provide useful ICT support for open source hardware communities and companies. The ICT features, tools and demonstrators are developed to support an envisioned story of a Platform Demonstrator. An example story is presented here:

A company is using Wikifactory platform (OSH platform) and is in the beginning of a project, they find a project that they would like to further work on. They would like to fork this project and start their own development. The project is currently located on GitHub; however, the company would like to work further on Wikifactory. So, they use the **Import-Export tool** to import the project files from GitHub. After importing the files, they would like to start a community for their project. This is when they browse through the profiles of makers on WIF based on the **skills** required. In addition, they also make an **open call** to find interested collaborators. As the project progresses in development the project is further detailed with its requirements with **skill tags** so that eager contributors can find the project. Here is where the **documentation templates** play an important part. The project is checked and cross-referenced with the templates to achieve complete and clear documentation. The company is worried about the licensing, that is when they refer to the **licensing guideline** to find a licence suitable for their repository and for their company/ community policy. The company finds an interesting markdown 3D printing **template**⁵¹, which they easily import and **integrate** it in WIF. It is also essential that the company indicates the maturity of the project and the documentation, where they refer to the **OTRL and ODRL** guideline and assigned the suitable one to the project. These steps enhanced the projects visibility and discoverability to the potential contributors. Last but not the least, the project is ready to be prototyped, the company used the **production matchmaking tool** and found interested Fablabs to work with. After working on the prototype, they use the **production matchmaking tool** again and found producers in two locations for their products.

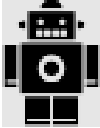





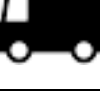


The above solutions were presented to the OPEN_NEXT SMEs and Fablabs, their feedback was taken into consideration and developments based on the feedback are under progress. The solutions will undergo pre-validation with external companies as the next step. The feedback from the pre-validation will be taken into consideration and further developed. In addition, for each of the solutions, in their summary and outlook section the further development ideas are also presented. The final validations will be carried out on stable versions of solution demonstrators.








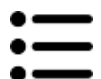


⁵¹<https://github.com/OPEN-NEXT/WP3-Documentation-Guidelines-for-OSH-Projects/blob/main/Documentation%20&%20Guidelines/3D%20Printing%20guideline.md>

7. Licenses

Category	Solution name	License	DOI
Interoperability	Import-Export tool	MIT License	https://doi.org/10.5281/zenodo.5336838
Community Management	Skill based matchmaking	GNU General Public License v3.0	https://doi.org/10.5281/zenodo.5336800
Collaborative Production	Identifying production metadata	GNU General Public License v3.0	https://doi.org/10.5281/zenodo.5336681
	Identifying production metadata in documentation through machine learning	GNU General Public License v3.0	https://doi.org/10.5281/zenodo.5336459
	Collaborative production and matchmaking tools on Wikifactory	Not applicable	Not applicable
Documentation and Guidelines	Project/Product overview template	GNU General Public License v3.0	https://doi.org/10.5281/zenodo.5336764
	Technology readiness levels for OSH	GNU General Public License v3.0	https://doi.org/10.5281/zenodo.5336681
	Licensing open source hardware guide	Creative Commons Attribution-Share alike 4.0 International Public License	https://doi.org/10.5281/zenodo.5156249
	Integrating templates in Wikifactory	Not applicable	Not applicable

8. Attributions

Image	Name	Author	License	Source
	Robot	Jean-Philippe Cabaroc	CC-BY 3.0	https://thenounproject.com/search/?q=22552&i=22552
	Book	Deemak Daksina	CC-BY 3.0	https://thenounproject.com/search/?q=1230262&i=1230262
	Industry	Shashank singh	CC-BY 3.0	https://thenounproject.com/search/?q=2094212&i=2094212
	Edit	Awesome	CC-BY 3.0	https://thenounproject.com/search/?q=958191&i=958191
	Network	Alina Alvarez	Public Domain	https://thenounproject.com/search/?q=50035&i=50035
	Platform	Adrien Coquet	CC-BY 3.0	https://thenounproject.com/search/?q=platform&i=3097564
	Transporter	Ralf Schmitzer	CC-BY 3.0	https://thenounproject.com/search/?q=transporter&i=745250
	Wikibase logo	H. Snater	GPLv3	https://en.wikipedia.org/wiki/Wikibase#/media/File:Wikibase_logo.svg
	Document	Arjuazka	CC-BY 3.0	https://thenounproject.com/search/?q=document&i=1830658

	Code	ProSymbols	CC-BY 3.0	https://thenounproject.com/search/?q=587999&i=587999
	Checkbox	ArtWorkLeaf	CC-BY 3.0	https://thenounproject.com/search/?q=1055725&i=1055725
	Checkbox	Thomas Le Bas	CC-BY 3.0	https://thenounproject.com/search/?q=2320266&i=2320266
	Eye	Vicons Design	CC-BY 3.0	https://thenounproject.com/search/?q=48975&i=48975
	File	Flamingo	CC-BY 3.0	https://thenounproject.com/search/?q=1407669&i=1407669
	File	Icon Lauk	CC-BY 3.0	https://thenounproject.com/search/?q=2154950&i=2154950
	Idea	Alkhalifi_design	CC-BY 3.0	https://thenounproject.com/search/?q=4116032&i=4116032
	List	Jonata hangga will putra march stanly	CC-BY 3.0	https://thenounproject.com/search/?q=4113813&i=4113813
	Broken Link	Creative outlet	CC-BY 3.0	https://thenounproject.com/search/?q=592865&i=592865
	Engineer	Gan Khoon Lay	CC-BY 3.0	https://thenounproject.com/search/?q=856401&i=856401

9. References

- Antoniou, Grigoris; van Harmelen, Frank (2009): Web Ontology Language: OWL. In: Steffen Staab und Rudi Studer (Hg.): Handbook on Ontologies. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 91–110.
- Dai, Jason Xinghang; Boujut, Jean-François; Pourroy, Franck; Marin, Philippe (2020): Issues and challenges of knowledge management in online open source hardware communities. In: *Des. Sci.* 6. DOI: 10.1017/dsj.2020.18.
- DIN SPEC 3105-1: DIN SPEC 3105-1:2020-07; Open Source Hardware - Teil 1: Requirements for technical documentation
- DIN SPEC 3105-2: DIN SPEC 3105-2:2020-07 Open Source Hardware - Part 2: Community-based assessment;
- Ding, Ying; Jacob, Elin K.; Zhang, Zhixiong; Foo, Schubert; Yan, Erjia; George, Nicolas L.; Guo, Lijiang (2009): Perspectives on social tagging. In: *J. Am. Soc. Inf. Sci.* 60 (12), S. 2388–2401. DOI: 10.1002/asi.21190.
- Guarino, Nicola; Oberle, Daniel; Staab, Steffen (2009): What Is an Ontology? In: Steffen Staab und Rudi Studer (Hg.): Handbook on Ontologies. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 1–17.
- Larrucea, Xabier; Santamaria, Izaskun; Colomo-Palacios, Ricardo; Ebert, Christof (2018): Microservices. In: *IEEE Softw.* 35 (3), S. 96–100. DOI: 10.1109/MS.2018.2141030.
- Li, Zhuoxuan; Seering, Warren (2019): Does Open Source Hardware Have a Sustainable Business Model? An Analysis of Value Creation and Capture Mechanisms in Open Source Hardware Companies. In: *Proc. Int. Conf. Eng. Des.* 1 (1), S. 2239–2248. DOI: 10.1017/dsi.2019.230.
- Méndez, Sergio J. Rodriguez; Haller, Armin; Omran, Pouya G.; Wright, Jesse; Taylor, Kerry (2020): J2RM: an Ontology-based JSON-to-RDF Mapping Tool. In: Kerry Taylor, Rafael Goncalves, Freddy Lecue und Jun Yan (Hg.): Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020), Bd. 2721. ISWC (Demos/Industry). globally online, 1-6 November, S. 368–373. Available online under <http://ceur-ws.org/Vol-2721/paper593.pdf>.
- Pan, Jeff Z.; Vetere, Guido; Gomez-Perez, Jose Manuel; Wu, Honghan (2017): Exploiting Linked Data and Knowledge Graphs in Large Organisations. Cham: Springer International Publishing.
- Pezoa, Felipe; Reutter, Juan L.; Suarez, Fernando; Ugarte, Martín; Vrgoč, Domagoj (2016): Foundations of JSON Schema. In: Jacqueline Bourdeau, Jim A. Hendler, Roger Nkambou Nkambou, Ian Horrocks und Ben Y. Zhao (Hg.): Proceedings of the 25th International Conference on World Wide Web. WWW '16: 25th International World Wide Web Conference. Montréal Québec Canada, 11.04.2016-15.04.2016. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, S. 263–273.