



Innovating Sales and Planning of Complex Industrial Products
Exploiting Artificial Intelligence

Deliverable 4.4

Knowledge Acquisition Component-Implementation and Test

Deliverable type:	Software
Deliverable reference number:	ITEA 20054 D4.4
Related Work Package:	WP4
Due date:	2024-10-31
Actual submission date:	2025-01-02
Responsible organisation:	Dakik Software
Editor:	Bilge Özdemir
Dissemination level:	Public
Revision:	Final Version 1.0

Abstract:	This deliverable provides the documentation of subcomponents of the Knowledge Acquisition Component (KAC). Subcomponent documentation includes information such as purpose, function, input and output data, and UI.
Keywords:	Knowledge Acquisition Component, intelligent price building, assisted knowledge acquisition, rule-based reasoning, data mining, machine learning, product data, knowledge management.

Table_head	Name 1 (partner)	Name 2 (partner)	Approval date (1 / 2)
Approval at WP level	ERMETAL	NATIF	04/12/2024
Veto Review by All Partners			02/01/2025

Editor

Bilge Özdemir (DAKIK)

Contributors

Alex Ivliev (TUD)

Mehtap Öklü (DAKIK)

Mert Daloğlu (DAKIK)

Bilge Özdemir (DAKIK)

Mario Thron (IFAK)

Yazmin Andrea Pabon Guerrero (UC3M)

Executive Summary

Deliverable 4.4 focuses on the implementation and testing of the Knowledge Acquisition Component (KAC), a core module of the InnoSale project designed to enhance the planning and sales processes for complex industrial products by leveraging artificial intelligence. This document provides comprehensive documentation of the subcomponents that form the KAC, detailing their purposes, functionalities, inputs, outputs, and user interfaces.

Key subcomponents covered include:

- **Fuzzy Logic Rules Editor** – A tool for managing fuzzy logic rules through a user-friendly interface.
- **Nemo Web App** – Facilitates inference-based reasoning on incomplete data using domain-specific rules.
- **Ontology Meter Viewer** – Offers insights into ontology structure and relationships.
- **FactoryLoadProvider Configurator** – Enables efficient planning and management of factory loads.
- **3D Part Visualizer Interface** – Supports collaborative editing, visualization, and operation management of 3D part models.

These subcomponents, implemented using state-of-the-art technologies, integrate seamlessly into the broader InnoSale platform. Their development aligns with the project's goals to simplify knowledge acquisition, enhance decision-making, and improve operational efficiency. The results of implementation and testing confirm the KAC's capability to support efficient data and knowledge management processes in industrial environments.

Table of Content

1	Introduction	1
2	Knowledge Acquisition Subcomponent Implementation & Test	1
2.1	Fuzzy Logic Rules Editor [IFAK]	1
2.1.1	Overview	1
2.1.2	Technical Details	1
2.1.3	Installation and Setup	1
2.1.4	Usage Examples	2
2.1.5	API Documentation	3
2.2	Nemo Web App [TU Dresden]	3
2.2.1	Overview	3
2.2.2	Technical Details	4
2.2.3	Installation and Setup	4
2.2.4	Usage Examples	5
2.2.5	API Documentation	6
2.3	Ontology Meter Viewer Component [Panel]	7
2.3.1	Overview	7
2.3.2	Technical Details	7
2.3.3	Installation and Setup	7
2.3.4	Usage Examples	8
2.3.5	API Documentation	9
2.4	FactoryLoadProvider Configurator [DEMAG, IFAK]	10
2.4.1	Overview	10
2.4.2	Technical Details	10
2.4.3	Installation and Setup	10
2.4.4	Usage Examples	11
2.4.5	API Documentation	12
2.5	3D Part Visualizer Interface [DAKIK, ERMETAL]	12
2.5.1	Overview	12
2.5.2	Technical Details	13
2.5.3	Installation and Setup	13
2.5.4	Usage Examples	13
2.5.5	API Documentation	31
3	Conclusion	31
4	Abbreviations	32

Tables

Figure 1: Fuzzy Logic Rules Editor	3
Figure 2 Interface of the Nemo Web Application.....	6
Figure 3 Tracing in the Nemo Web App.....	6
Figure 4 Ontology Meter viewer interfaz	9
Figure 5: FactoryLoadProvider Configurator	12
Figure 6 Parts List	15
Figure 7 Create New User Screen	15
Figure 8 System Logs Screen	16
Figure 9 My Account Screen	16
Figure 10 My Tasks Screen	17
Figure 11 Loading Screen.....	17
Figure 12 GIU Startup Screen	18
Figure 13 View Helper	19
Figure 14 Surface Selection	19
Figure 15 Add Operations.....	20
Figure 16 Operations / Single Operation Menu	20
Figure 17 Highlight.....	21
Figure 18 Example of Operations Attributes	21
Figure 19 Bounding Box.....	22
Figure 20 Operation Vectors	22
Figure 21 Operation Vectors, Angle-Based Colouring	23
Figure 22 Operation Vectors (Holes)	24
Figure 23 Example of R/L Symmetry.....	25
Figure 24 General Operations	25
Figure 25 Part Selection Page	26
Figure 26 GUI Holes	27
Figure 28 Unfolding	27
Figure 28 Form Calculation.....	29
Figure 29 Similar Part Finder Page.....	31

1 Introduction

The Knowledge Acquisition Component (KAC) is a critical part of the InnoSale project, enabling authorized users to input and update product data and knowledge efficiently. This documentation provides detailed information about the various subcomponents that make up the KAC, offering insights into their functionalities, technical implementations, and usage guidelines.

The primary purpose of this documentation is to serve as a comprehensive reference for developers, integrators, and users involved in the InnoSale project. It aims to facilitate a deeper understanding of the KAC's subcomponents, enabling effective utilization, maintenance, and potential extensions or customizations.

By providing thorough explanations of each subcomponent's architecture, design considerations, and integration points, this documentation empowers stakeholders to leverage the full potential of the KAC. Additionally, it offers guidance on installation, configuration, and deployment processes, ensuring a smooth and efficient setup across different environments.

Overall, this subcomponent documentation plays a crucial role in supporting the successful implementation and adoption of the Knowledge Acquisition Component within the InnoSale project, enabling seamless integration of new products and efficient management of product data and knowledge.

2 Knowledge Acquisition Subcomponent Implementation & Test

2.1 Fuzzy Logic Rules Editor [IFAK]

2.1.1 Overview

The Rules Editor page is an element of a multi-page Web-application. It provides a tool for managing fuzzy logic rules within the application. It features a user-friendly text editor interface that allows you to enter or modify existing rules. Once you have made your changes, you can save the updated rules through an API call, ensuring that the backend logic is immediately updated to reflect these modifications.

2.1.2 Technical Details

The application uses the Streamlit GUI framework. It has access to the API of the Fuzzy Logic Engine developed in D3.5 and also uses it. Locally, the Fuzzy Logic rules are stored as a text file.

2.1.3 Installation and Setup

To run the Fuzzy Logic Rules Editor locally, ensure you have Python 3.10+ installed. Then proceed following steps:

a. Clone the repository:

Execute following commands:

```
unzip innosale_fuzzylogic_frontend-main.zip
cd fuzzy_pricing_app
```

b. Create and activate a virtual environment:

Create a virtual environment with the `--copies` flag (to copy files rather than use symbolic links):

```
python3 -m venv .venv --copies
```

Activate the virtual environment:

- On macOS/Linux:

```
source .venv/bin/activate
```

- On Windows:

```
.venv\Scripts\activate
```

c. Install the dependencies:

Execute this command:

```
pip install -e .
```

d. Run the application:

Then you can run the Application

```
fuzzy_pricing_app
```

This will start the Streamlit app, which can be accessed at <http://localhost:8501> in your browser.

A completely different Installation routine is by **using Docker**. For that, you need to perform **step a** as in the previous installation procedure. For containerized deployment, a Dockerfile is included.

i. Build the Docker image:

You need to build a docker image based on the provided code and Dockerfile:

```
docker build -t fuzzy_pricing_app .
```

ii. Run the Docker container:

Then you can instantiate the Docker container:

```
docker run -p 8501:8501 fuzzy_pricing_app
```

Finally, the app will be accessible at <http://localhost:8501>.

2.1.4 Usage Examples

Figure 1 provides an overview of the application window. The main UI control is the editor pane, where you can edit the Fuzzy Logic rules. The syntax is according to the IEC61131-7 Fuzzy Control Language. It contains parts for fuzzyfication and defuzzyfication of input or respectively output variables and finally for the logic rules.

New variables can be introduced by using the “Pricing Factors” page, and execution of the pricing is done by using the “Price Calculation” pages of this application. Both pages are described in D4.2. When pressing the “Save Rules” button, then the component triggers the “set_rules” endpoint of the Fuzzy Logic Engine (which has been developed in D3.5).

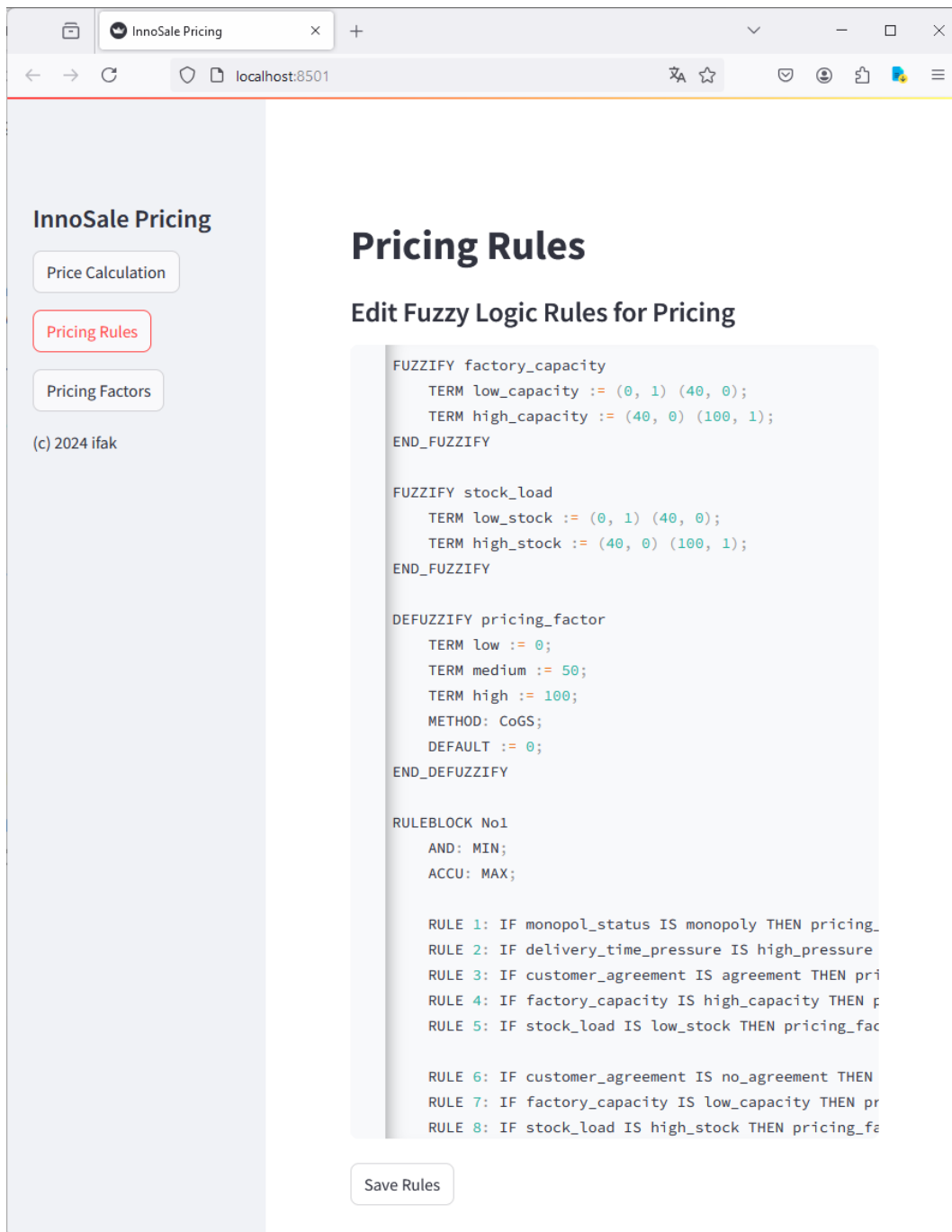


Figure 1: Fuzzy Logic Rules Editor

2.1.5 API Documentation

This application has no API by its own, but it uses the API of the Fuzzy Control Language Engine (FCLE). This component is developed in D3.5 and the API is documented there.

2.2 Nemo Web App [TU Dresden]

2.2.1 Overview

Customer requests often arrive with incomplete details, relying on context or common knowledge to convey some of the required information implicitly. To facilitate the automatic processing of such requests, it is crucial to make all the available information explicit. Within

the InnoSale Platform, this task is accomplished by the Datalog engine Nemo (a detailed description can be found in D3.6). It relies on rules that encode domain-specific knowledge, allowing it to make inferences on incomplete information. Such inference rules need to be supplied by human experts, who have deep knowledge of the application domain, but are not necessarily familiar with the intricacies of using an inference engine. The Nemo Web App aims to reduce the barrier by providing access to Nemo via a web interface.

The Nemo Web App provides:

- A text editor for entering rules, which is capable of syntax highlighting and offers basic refactoring functions
- A way to reference local and remote input files
- A way to store and share written Datalog programs
- Compute the inferences (by running Nemo locally within the browser)
- Explanations of the inferred facts, displayed in a visual way

The application can be accessed here: <https://tools.iccl.inf.tu-dresden.de/nemo/>.

2.2.2 Technical Details

The Nemo Web App is developed with TypeScript as the primary programming language. It is built as a Single Page Application using Vite. For rendering the user interface, Nemo Web relies on React.js, which is also used for managing the application's state. The text editor is powered by Monaco Editor. To display complex errors and identify different syntactical elements for syntax highlighting, Nemo implements the Language Server Protocol.

The Datalog program is evaluated by Nemo, which is written in Rust and exposed to the frontend through a WebAssembly (WASM) API. Hence, reasoning is performed within the web browser, entirely on the user's machine. No user data is uploaded to a server.

The source-code is openly available on GitHub: <https://github.com/knownsys/nemo-web/>.

2.2.3 Installation and Setup

The web application is available here: <https://tools.iccl.inf.tu-dresden.de/nemo/>.

Since its components are openly available, it is also possible to build the application from source. This requires installing Rust's package manager cargo, the JavaScript package manager npm, and the wasm-pack crate. An up-to-date version of the installation steps described next, can be found in the README files of the respective repositories:

1. Build Nemo and the language server:

```
git clone https://github.com/knownsys/nemo
cd nemo
cargo build -r

cd nemo-language-server
cargo build -r
```

2. Build the WASM library:

```
cd ../nemo-wasm

wasm-pack build --out-dir nemoWASMBundler -t bundler --weak-refs --release
wasm-pack build --out-dir nemoWASMWeb -t web --weak-refs --release
```

3. Build the VS Code extension

```
# In root directory
git clone https://github.com/knowsys/nemo-vscode-extension
cd nemo-vscode-extension

cp -r <PATH_NEMO>/nemo-wasm/nemoWASMWeb .

npm i
npm run package
```

4. Build web application


```
# In root directory
git clone https://github.com/knowsys/nemo-web
cd nemo-web

mkdir nemoVSIX
cp <PATH_VSCODE_EXTENSION>/nemo-<VERSION>.vsix nemoVSIX/nemo.vsix
cp -r <PATH_NEMO>/nemo-wasm/nemoWASMBundler .

npm i
npm run dev
NODE_OPTIONS=--max_old_space_size=4096 npm run build
```


2.2.4 Usage Examples

The user enters a logic program written in the Nemo dialect of Datalog into the online editor shown in Figure 2 Interface of the Nemo Web Application. Alternatively, existing programs may be loaded from disk or shared via link. Syntax highlighting and semantic-aware refactoring provided by the editor simplify the rule-writing process. Additional data can be imported from online sources using import statements or local files can be added by clicking “Add local file as input”. To start the reasoning process, the user clicks “Run program”. The inferred facts are displayed in tables and can be downloaded in various formats, including CSV and RDF formats.


Nemo
Graph Rule Engine

v0.5.2-dev Nemo on Github Web Interface on Github Docs

Give feedback!



Code editor
Examples
Open file
Save file

```

51 % This is needed for computing the distances in meters.
52 %
53 % This calculation is an approximation based on an a stackoverflow ans
54 % https://stackoverflow.com/questions/639695/how-to-convert-latitude-c
55
56 coordinates_m(?Xm, ?Ym) :- coordinates(?longitude, ?latitude), PI(?PI)
57 ?Xm = ?longitude * ?earth * COS(?latitude / 180.0 * ?PI) / 360.0,
58 ?Ym = ?latitude * (?earth / 360.0).
59 project_location_m(?project, ?Xm, ?Ym) :- project_location(?project, ?
60 ?Xm = ?longitude * ?earth * COS(?latitude / 180.0 * ?PI) / 360.0,
61 ?Ym = ?latitude * (?earth / 360.0).
62
63 % An environmental impact assessment is only needed,
64 % if the planned wind turbine is taller than 50m.
65 environmental_impact(?project) :- project_height_m(?project, ?height),
66
67 % Calculate for each project the distance to all residence buildings
68 distance(?project, ?distance) :-
69   project_location_m(?project, ?Xp, ?Yp),
70   coordinates_m(?Xr, ?Yr),
71   ?distance = SQRT(POW(?Xp - ?Xr, 2.0) + POW(?Yp - ?Yr, 2.0)).
72
73 % A project is rejected if there is a residence building closer than 2
74 project_reject(?project) :- distance(?project, ?distance), ?distance <
75
76 % A project is rejected if an environmental impact assessment is neede
77 % and an endangered species was found.
78 project_reject(?project) :-
79   environmental_impact(?project),
80   project_species(?project, ?species),
81   endangered(?species).

```

Program execution

Run program
Add local file as input

Input files (local):

endangered.csv

322.0 B

×

locations.csv

105.6 KB

×

Results:

Derived 621 facts in 0.1 seconds (11+25+8 ms)

CIRC_EARTH (1)
PI (1)
TRUE (1)
closest_house (3)

coordinates (122)
coordinates_m (122)
distance (366)

endangered (21)
environmental_impact (2)
locations (781)

project (3)
project_approved (1)
project_height_m (3)

project_location (3)
project_location_m (3)
project_reject (2)

project_species (4)

project_reject (2 rows)
Save all rows as CSV

1 "Wind Turbine A"

2 "Wind Turbine B"

Figure 2 Interface of the Nemo Web Application

Optionally, an explanation as for why a particular fact was derived may be requested. Such explanations are displayed visually as proof trees, traces, that show what rules and facts lead to the derivation of which information. The interface further allows collapsing and expanding nodes for better clarity. An example trace is displayed in **Figure 3**.

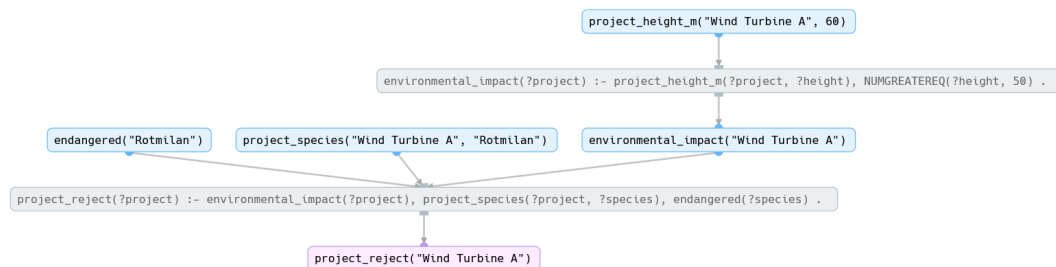


Figure 3 Tracing in the Nemo Web App

2.2.5 API Documentation

The Nemo Web Assembly API provides the following functions:

Name	Function
reason	Given a Nemo logic program and a set of input files, compute the logical consequences of the logical rules
getResult	Get the result of the computation triggered via the reason function
savePredicate	Save the result of a specific table to disk
traceFact	Given a fact, compute a proof tree that explains its origin

2.3 Ontology Meter Viewer Component [Panel]

2.3.1 Overview

The Ontology Meter Viewer Component is designed to help experts understand the structure of the ontology and, thus, identify areas for improvement and optimization. This component shows ontology information:

- Number of Concepts: The total number of concepts in the ontology.
- Number of Properties: The count of properties in the ontology (object properties, data properties, etc).
- Number of Instances: How many individual instances (members) have been defined for each concept?
- Hierarchy Depth: The maximum depth of the class hierarchy in the ontology.
- Number of Relationships: How many relationships or connections are defined between the concepts in the ontology?
- Degree of Connectivity: The number of relationships that leave or reach a specific concept.
- Ontology Density: The proportion of defined relationships concerning the total number of concepts.
- Axiom Complexity: How many axioms or logical rules have been defined in the ontology?
- File Size: The size in bytes or kilobytes of the file containing the ontology.

2.3.2 Technical Details

As part of the InnoSale project, a web application is provided. To run this application locally:

Prerequisites

- Node.js (v16+)
- Angular CLI
- Python 3.9+
- pip (Python package manager)
- Virtual environment tool (venv or conda)

a) Clone the repository

```
unzip innosale_ontology_meter_viewer-main.zip  
cd innosale_ontology_meter_viewer
```

Backend:

a) Navigate to backend directory

```
cd ../backend
```

b) Create virtual environment

```
python -m venv venv source venv/bin/activate
```

c) Install dependencies

```
pip install -r requirements.txt
```

d) Run FastAPI server

```
uvicorn main:app --reload
```

FrontEnd:

a) Navigate to backend directory

```
cd ../backend
```

b) Install dependencies

```
npm install
```

c) Run Server

```
ng serve
```

This will start the application with the frontend running on `http://localhost:4200` and the backend API server running on `http://localhost:8000`. You can access the Ontology Meter Viewer by opening a web browser and navigating to the frontend URL. The Swagger UI for the API documentation can be accessed at `http://localhost:8000/docs`, allowing you to explore and test the available API endpoints directly from your browser.

2.3.3 Installation and Setup

As part of the InnoSale project, a web application is provided.

2.3.4 Usage Examples

Figure 4 provides an overview of the application window. The Upload Ontology button allows you to upload an ontology file with a `.owl` or `.rdf` extension.

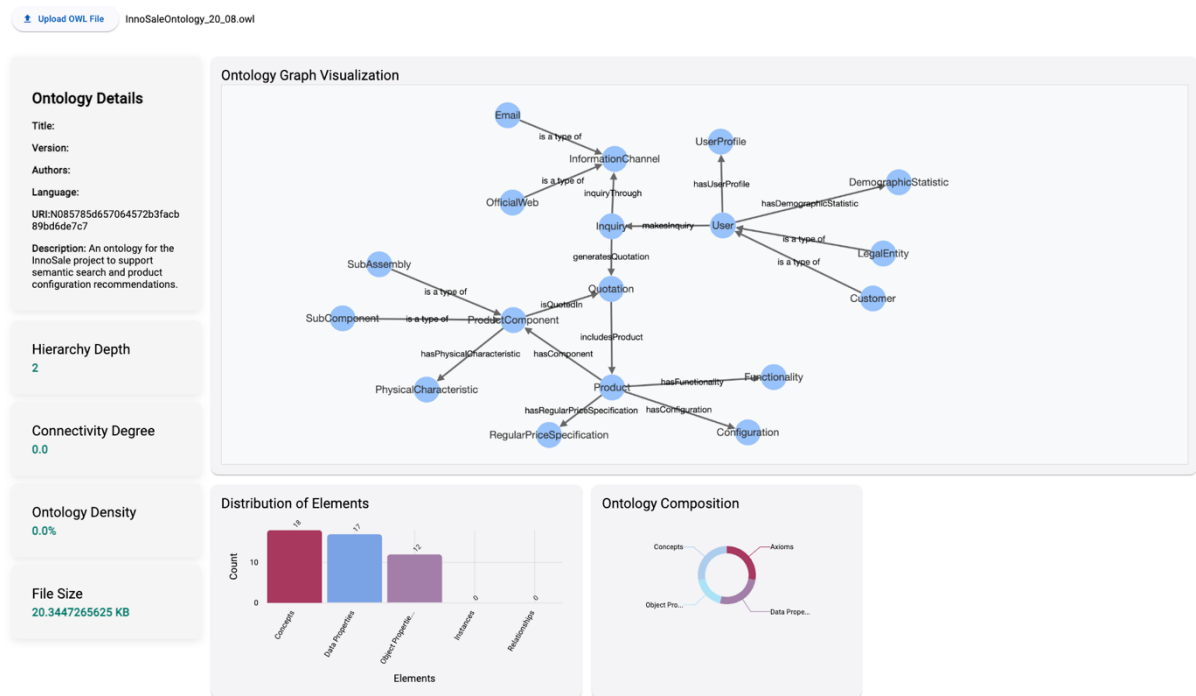


Figure 4 Ontology Meter viewer interface

Once the file is loaded, the tool displays a list of key metrics such as hierarchy depth, the number of relationships, and ontology density. It also provides a visual representation of the ontology in the form of a graph.

This interactive graph allows users to explore the ontology in more detail. When a user clicks on a specific ontology concept, the tool displays relevant information like the name of the concept, its data properties, and other important details. This functionality makes it easier for users to understand the relationships and structure of the ontology, enabling an intuitive and detailed exploration of its components.

2.3.5 API Documentation

The following table provides an overview of the available API endpoints, their respective functions, and a brief description of their purpose:

Name	Function
/ontology/upload	Uploads a new ontology file (in OWL, RDF, TTL, or N3 formats) and replaces the current instance for further analysis and exploration.
/analyze-ontology/	Analyzes the uploaded ontology, returning general metrics and graph elements representing the ontology's structure.
/ontology/concepts	Retrieves a list of all concepts (classes) defined within the ontology.
/ontology/concepts/{concept_id}	Retrieves detailed information about a specific concept, including its superclass hierarchy and properties.

/ontology/search	Searches for concepts that match a specified keyword, useful for exploring the ontology by term or partial matches.
/ontology/concepts/{concept_id}/properties	Lists all properties (both object and data properties) associated with a specific concept, providing domain and range information.
/ontology/relationships	Returns a list of all relationships between concepts, showcasing how different concepts are interconnected.
/ontology/metrics	Provides general metrics of the ontology, including counts of concepts, properties, instances, and other key attributes that summarize the ontology's scope.

2.4 FactoryLoadProvider Configurator [DEMAG, IFAK]

2.4.1 Overview

FactoryLoadProvider Configurator is a multi-page Streamlit application for configuring the factory load of a machine manufacturer. It allows users to define factory load as percentages (%) mapped to specific time slots, such as days or weeks. The configured data is used to populate and manage a REST-API endpoint providing real-time access to the factory load status.

2.4.2 Technical Details

The application uses the Streamlit GUI framework and integrates with the Factory Load Provider API to manage and retrieve factory load data. Locally, the factory load data is stored as a CSV file, named `factory_load.csv`.

2.4.3 Installation and Setup

To run this application locally, ensure you have Python 3.10+ installed. Then proceed following steps:

a. Clone the repository:

```
unzip innosale_factoryload_frontend-main.zip
cd innosale_factoryload_frontend
```

b. Create and activate a virtual environment:

Create a virtual environment with the `--copies` flag (to copy files rather than use symbolic links):

```
python -m venv venv --copies
```

Activate the virtual environment:

- On macOS/Linux:

```
source venv/bin/activate
```

- On Windows:

```
venv\Scripts\activate
```

c. Install the dependencies:


```
pip install -e .
```

d. Run the application:

```
streamlit run --server.port=8502 src/floadfe/main.py
```

This will start the Streamlit app, which can be accessed at <http://localhost:8502> in your browser.

Note: On our setup, the console window showed by error, that the server is running on port 8088. The app was still accessible via <http://localhost:8502>.

A complete different Installation routine is by **using Docker**. For that, you need to perform step “a” as in the previous installation procedure. For containerized deployment, a Dockerfile is included.

i. Build the Docker image:

You need to build a docker image based on the provided code and Dockerfile:

```
docker build -t factoryload_frontend .
```

ii. Run the Docker container:

Then you can instantiate the Docker container:

```
docker run -p 8502:8502 factoryload_frontend
```

Finally, the app will be accessible at <http://localhost:8502>.

2.4.4 Usage Examples

In the example usage of the Factory Load Management application, the user accesses the web interface via a browser. The page is divided into two sections:

- **Left Section (Weeks 1-8):** The left section shows sliders for adjusting the load values for the first 8 weeks, starting with the current week. Each week is represented by a slider labeled with the respective week in the format "Year/Week" (e.g., "2024/45"). The user can modify the load for each week by dragging the sliders, with values ranging from 0 to 250.
- **Right Section (Weeks 9-16):** Similarly, the right section displays sliders for weeks 9 to 16. Users can adjust the load for these weeks in the same way as the left section.

The application automatically checks the CSV file for the necessary 16 weeks of data. If some weeks are missing, they are added with a default load of 100%. Users can modify these values directly through the sliders.

Once the user has made the necessary adjustments to the load values, they can click the Save button, which updates the CSV file with the new data. This ensures that the load changes are persistent and can be tracked for future reference.

Figure 5 visually show the left and right sections with sliders for each of the 16 weeks, offering a clear and interactive interface for managing factory load data efficiently.

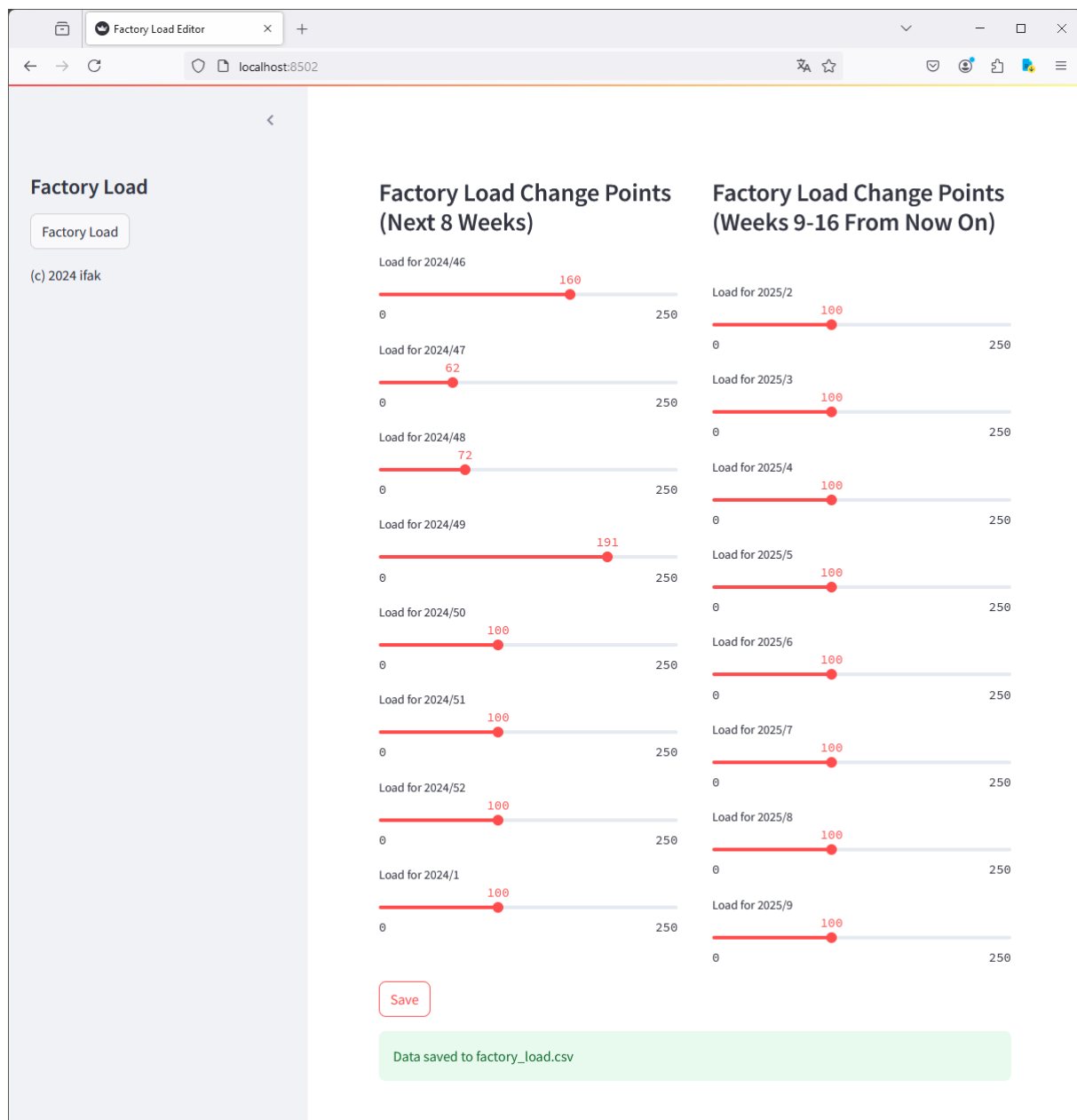


Figure 5: FactoryLoadProvider Configurator

2.4.5 API Documentation

This application has no API by its own, but it uses the API of the Factory Load Provider. This component is developed in D4.6 and the API is documented there.

2.5 3D Part Visualizer Interface [DAKIK, ERMETAL]

2.5.1 Overview

The 3D Part Visualizer Interface is a collaborative tool that allows users to visually inspect, annotate, and perform operations on 3D part models in real-time. Its primary purpose is to facilitate effective communication and knowledge sharing between sales engineers, product experts, and customers during the complex industrial product sales process.

By providing a shared 3D visualization environment, the interface enables seamless collaboration where multiple stakeholders can remotely view the same 3D part model

simultaneously. This visual aid is particularly useful when discussing complex product configurations, identifying areas that require modifications, or communicating specific requirements to manufacturing teams.

Some key features include:

- Loading and visualizing 3D parts (STL files)
- Surface selection on 3D parts
- Adding and editing operations (cutting, drilling, etc.) on selected surfaces
- Visualizing symmetry of parts
- Detecting and highlighting holes in parts
- Collaborative editing with multiple users
- Tracking user actions and operations performed
- Intuitive GUI for controlling 3D view and performing operations

2.5.2 Technical Details

The **3D Part Visualizer Interface** enables multiple users to collaborate on 3D models and track every operation performed. The 3D environment is powered by **ThreeJS** for rendering and manipulating 3D objects in the browser, while the GUI is managed by **Tweakpane** for intuitive user controls. These components communicate through **ReactJS**, which manages state and ensures real-time updates as users modify parts or perform operations.

Data flow between the frontend and backend is handled by a **Python Flask API**, which manages server-side operations like fetching and sending data related to 3D parts. The API retrieves data from a **MongoDB** that stores information on parts, operations, and user activities.

This architecture ensures real-time collaboration, efficient tracking, and scalability for future features.

2.5.3 Installation and Setup

As part of the InnoSale project, a web application is being developed with a React.js user interface and a REST API, using Python and Flask, backed by a MySQL database. A setup script has been initiated to simplify installation and data migration on a new server. This script automates deploying the system, saving 800, 3D files from Ermetal, creating the database and tables, and writing the data. Configuration settings will be added as needed to further streamline the process.

2.5.4 Usage Examples

Dashboard

InnoSale

DASHBOARD

 Ecommerce

 Dashboard

PAGES

 Create New User


 System Logs

 My Account

 My Tasks

 Offers

 All Parts

 All Operations

NLP

 Audio Files

 Summarization

 Semantic Search

THREE JS

 ThreeJS Scene

PREDICT

 Price Prediction

The interfaces developed provided comprehensive pages that allowed members of various departments of Ermetal to efficiently manage users, user actions, offers, parts and operations within the system. These interfaces allowed each department to easily access and control relevant data related to their own tasks. To meet these needs, administrators define new users to the system using the **Create New User** page. They can also track the transactions performed by all users via the **System Logs** page. All users, including administrators, use the **My Account** page to edit their own account information. The **My Tasks** page is used to assign tasks to any user and track them. Extensive work was done to create user-friendly interfaces for management functions such as **Offers**, **All Operations and All Parts**. In addition, the **Audio Files** page was developed to extract information from previous meeting records. Here, track of the offers given in the past in audio files can be followed and the **Summarization** page was created to get a summary of the relevant meeting content and search within it. The **Semantic Search** interface was developed to search within the summary. A ThreeJS Scene page was created to visualize and manipulate parts, a **Similarity** page was created to calculate the similarities of these parts, and a **Price Prediction** page was created to calculate Labor Hours and Form Cost on parts.

This setup allowed for streamlined workflows and collaboration across departments.

The part list contains detailed information about metal parts in the system. Each attribute of a part is displayed in a separate column, including the part's geometric data, file path, and manufacturing date. After selecting a part, the "View File" button allows users to examine a visual representation of the chosen part in a 3D environment for closer inspection.

Page

Parts List

ADD

VIEW FILE

EDIT

DELETE

DETAIL

COLUMNS

FILTERS

DENSITY

EXPORT

ID

Teklif Id

Teklif No

Teklif Talep Rev No

Sac Kalınlık

Sac Cinsi

Net X

Net Y

Net XY Division

Kontur Boyu

Açınım Yüzey Alanı

Sac Ts Max

Sac Uzama

Sertlik

2

60

001-2020

0

1.2

XES

95

71

1.33803

321

5915

350

37

YUMUŞA

3

61

001-2020

0

1.2

XES

157

63

2.49206

395

8203

350

37

YUMUŞA

4

5

001-2020

0

2

XES

393

334

1.17665

1328

111166

350

37

YUMUŞA

5

7

001-2020

0

2

XES

591

240

2.4625

2012

94758

350

37

YUMUŞA

6

62

001-2020

0

0.8

XES

640

260

2.44075

1819

115875

350

37

YUMUŞA

Rows per page:

5

1-5 of 882

Figure 6 Parts List**Create New User**

This page was developed for administrators to create new users on the InnoSale ecosystem.

Pages

Create New User




Figure 7 Create New User Screen**System Logs**

This page was developed for administrators to check all users actions on InnoSale ecosystem.

Page

System Logs

Start Date
10/01/2024 12:00 AM

End Date
11/01/2024 12:00 AM

CLEAR

GET LOGS

Page

Type

User Type

OfferID

Description (contains)

Date	Offer ID	Page	Type	By	Description
Fri, 04 Oct 2024 14:11:49 GMT	10	Similar Part Finder	Calculation	Test Doe	Similar part found for 10: geometry_based
Fri, 04 Oct 2024 14:11:19 GMT		Login	Login	Test Doe	User logged in
Fri, 04 Oct 2024 14:11:19 GMT		Login	Logout	Test Doe	User logged out
Fri, 04 Oct 2024 14:11:19 GMT		Login	Logout	Test Doe	User logged out
Thu, 03 Oct 2024 16:48:09 GMT		Login	Login	Test Doe	User logged in

Rows per page: 5 1-5 of 2640


Figure 8 System Logs Screen

My Account

This page was developed for all users to edit their account info on the InnoSale ecosystem.

Pages

My Account



Name
Test

Surname
Doe

Username
admin

Mail
admin@innosale.com

New Password

Valid New Password

Current Password

RESET CHANGES

APPLY CHANGES

CHOOSE FILE

UPLOAD

Figure 9 My Account Screen

My Tasks

This page was developed for all users to track their tasks on the InnoSale ecosystem.

Pages

My Tasks

Filter Tasks

☒ Assigned to me ☐ Assigned by me

REFRESH

ASSIGN NEW TASK







Columns	Filters	Density	Export				
Date	Assigned By	Assigned To	Offer ID	Done Text	To-Do Text	Accomplished	
Mon, 14 Aug 2023 15:55:05 GMT	 Test Doe	 Test Doe	10	from admin which tekliif_id = 10	10 processing	 Completed	
Mon, 14 Aug 2023 15:36:20 GMT	 Test Doe	 Test Doe	2	from admin which tekliif_id = 2	to admin which tekliif_id = 2 aaa...	 Not completed	
Rows per page: 5 1-2 of 2 < >							

Figure 10 My Tasks Screen

Loading Screen

Loading screen has been developed to make the ThreeJS page more secure and visually appealing while everything loads.



Figure 11 Loading Screen

GUI Startup Screen

Under this screen, various operations can be performed on the 3D scene via the GUI.



Figure 12 GIU Startup Screen

File: The STL files used in the project are stored in a specific location on the server, which were previously uploaded. The list of files from this location is fetched from the backend server and displayed in this controller. The user can select the 3D part they wish to work on from this list.

After File Upload: Once a part is selected, it can be loaded into the system either with or without thickness. In this application, all operations are performed on parts without thickness. Therefore, a surface of the selected part must be chosen, and operations can then continue on that surface.

At this stage, certain calculations are performed to enable the surface selection process. The pseudo-code for these calculations is shown below:

```
point_to_triangle_map = {}
points_neighbors_map = {}

triangle_normals_list = []
triangle_points_list = []

for triangle in triangles:

    triangle_index = get_triangle_index(triangle)
    triangle_normal = get_triangle_normal(triangle)

    triangle_points_list.push([])
    triangle_normals_list.push(triangle_normal)

    for point in triangle:

        point_index = get_point_index(point)
        triangle_points_list[triangle_index].push(point_index)

        if point_index in point_to_triangle_map:
            point_to_triangle_map[point_index].push(triangle_index)
        else:
            point_to_triangle_map[point_index] = [triangle_index]

    for index, point_index in triangle_points_list[triangle_index]:

        for sub_index, sub_point_index in triangle_points_list[triangle_index]:

            if index != sub_index:
                if point_index in points_neighbors_map:
                    points_neighbors_map[point_index].push(sub_point_index)
                else:
                    points_neighbors_map[point_index] = [sub_point_index]
```

Controls and Speed:

Orbit Control: The user can view the part from different angles by using the mouse. The perspective can be changed by left-clicking, right-clicking, middle-clicking, and scrolling, allowing the user to navigate the 3D scene easily.

View Helper: The user can also use the view helper located in the top right corner to look at the object from specific angles.



Figure 13 View Helper

Keys and Speed: If the user wants to move the camera freely, they can use the WASD keys for directional movement. To rotate the view, the QE keys can be used. While using these keys, the camera moves at a set speed, which can be adjusted using the 'Speed' controller.

Lock Status: When using the Orbit Controls and QE keys, movement occurs along the XYZ axes. Any of these axes can be locked, preventing movement along that axis. Once locked, no movement will take place on the selected axis.

Surface Selection: The user selects a surface and confirms the selection with the 'Confirm' button in the GUI. After confirmation, the user can continue performing operations on the chosen surface.

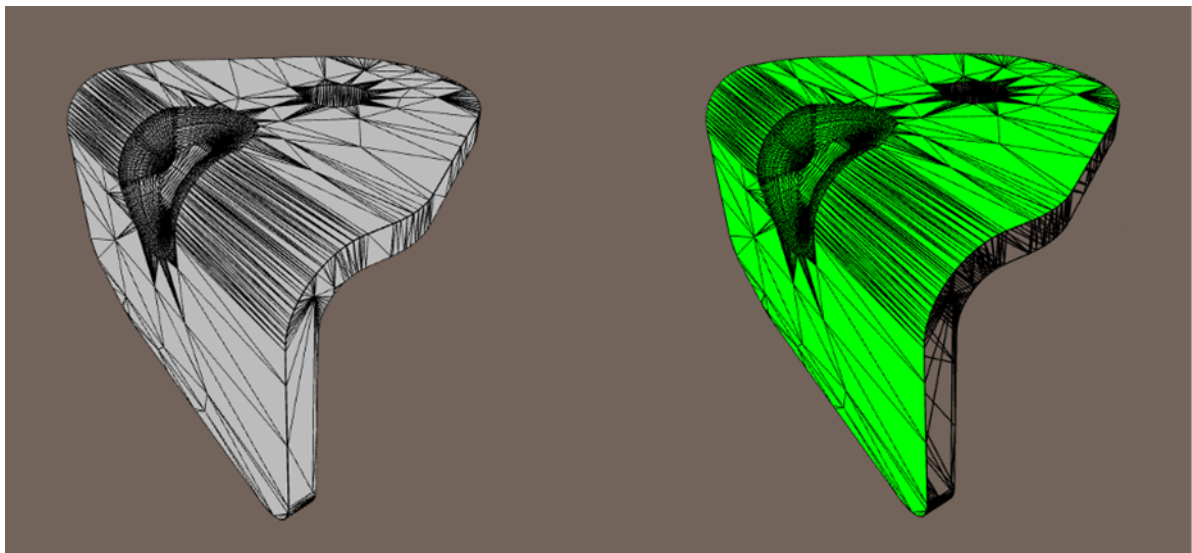


Figure 14 Surface Selection

The pseudo-code for surface selection is as follows:

```
used_triangles_list = [False] * len(triangles_list)
selected_triangle = get_selected_triangle()

queue = [selected_triangle]
surface_triangles = [selected_triangle]

while len(queue) > 0:
```

```

triangle_index = queue.pop()
triangle_normal = triangle_normals_list[triangle_index]
triangle_points = triangle_points_list[triangle_index]

for point_index in triangle_points:
    neighbor_points = points_neighbors_map[point_index]
    neighbor_triangles = [
        point_to_triangle_map[point] for point in neighbor_points
    ]
    for neighbor_index in neighbor_triangles:
        if used_triangles_list[neighbor_index] == False:
            neighbor_normal = triangle_normals_list[neighbor_index]
            normal_angle = angle(triangle_normal, neighbor_normal)

            if normal_angle < degree(5):
                queue.push(neighbor_index)
                surface_triangles.push(neighbor_index)
                used_triangles_list[neighbor_index] = True

```

Part Operations

Users can add operations to the parts through the GUI.

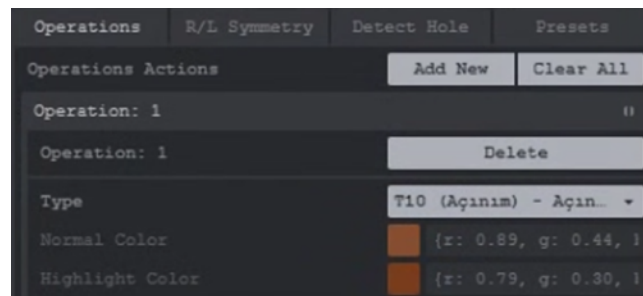


Figure 15 Add Operations

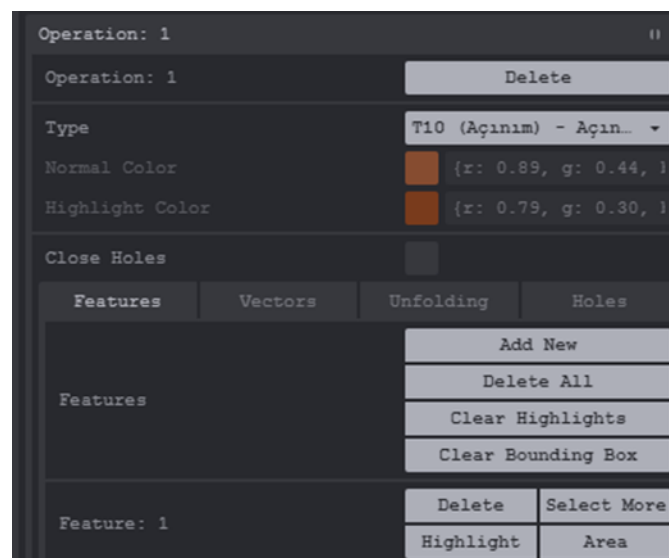


Figure 16 Operations / Single Operation Menu

Operation Type: This section specifies the type of operation.

Normal and Highlight Color: This section defines the normal and highlighted working colors.

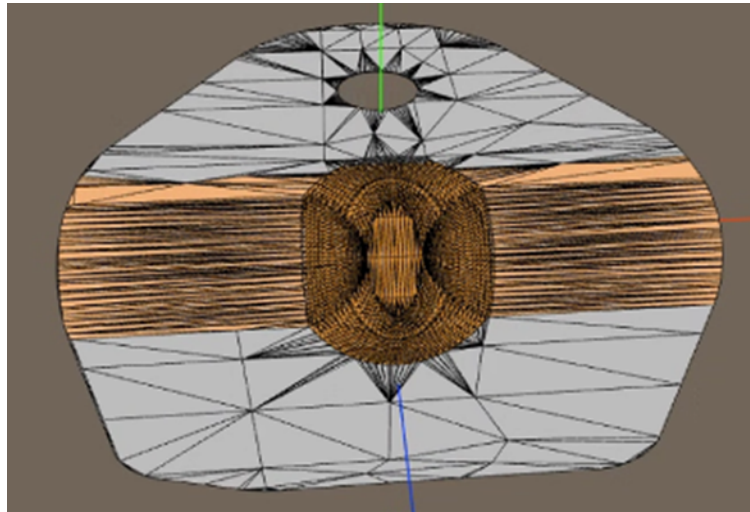


Figure 17 Highlight

Attributes: Users can assign attributes to operations. Once an attribute is assigned, certain triangles on the part are painted.

Features	Vectors	
Features	Add New	
	Delete All	
	Clear Highlights	
	Clear Bounding Box	
Feature: 1	Delete	Select More
	Highlight	Area
Feature: 2	Delete	Select More
	Highlight	Area

Figure 18 Example of Operations Attributes

Bounding Box: When users click the 'Area' button, the bounding box for the assigned attributes becomes visible.

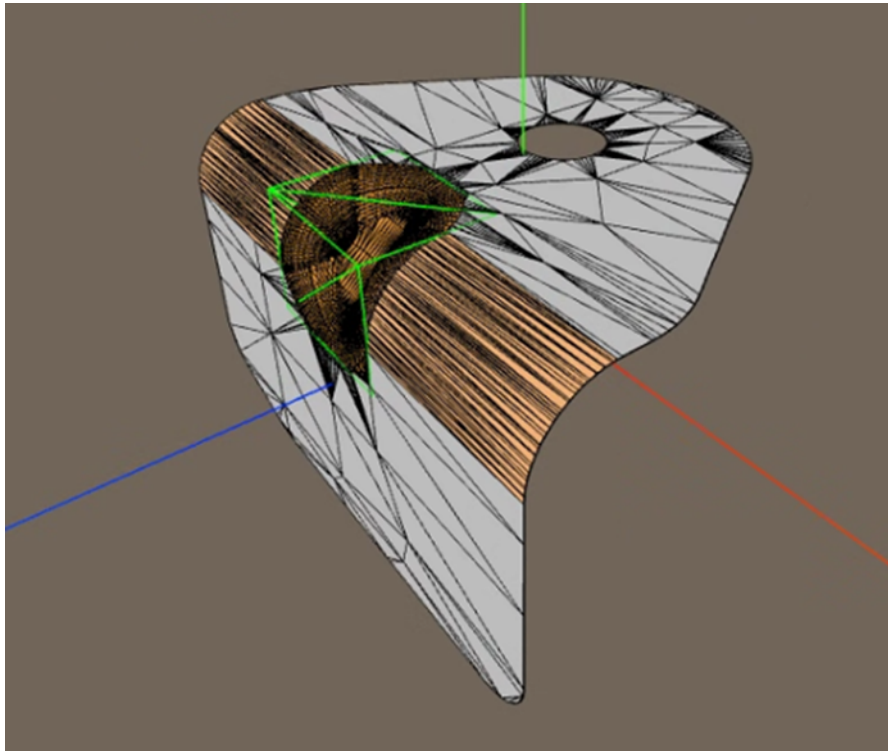


Figure 19 Bounding Box

Vectors:

Normal Vector: The user sets the normal vector for the operation. A flat white plane is positioned towards the part based on this vector. To adjust normal vectors, users can utilize the 'Normal Vector', 'Reset', 'From Mesh Selection', and 'From Another Operation' controls.

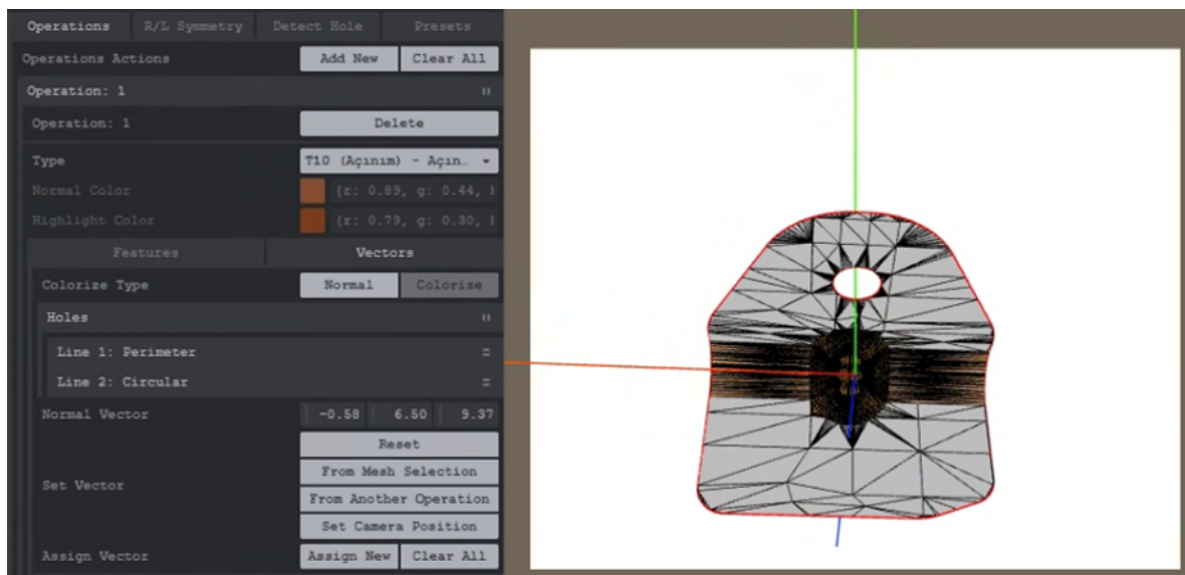


Figure 20 Operation Vectors

Vector Assignment: The user can assign vectors on the background white plane. Each vector has a distinct color and code.

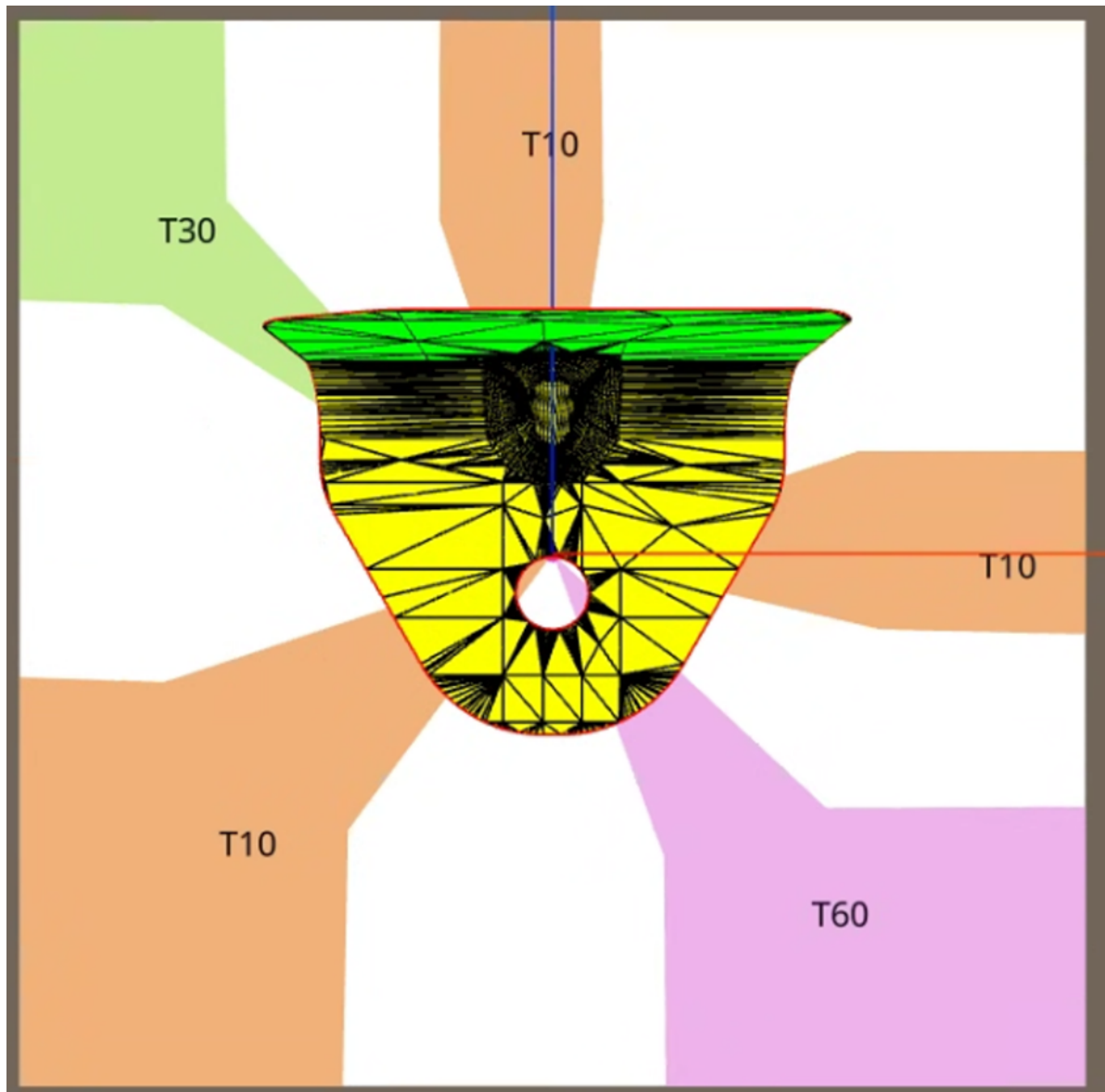


Figure 21 Operation Vectors, Angle-Based Colouring

Angle-Based Coloring: Users can toggle a feature that colors triangles based on the angle between the triangle's normal vector and the operation's normal vector. Triangles with a 90degree difference are green, those with angles less than 90 degrees are yellow, and those with angles greater than 90 degrees are red.

Holes: When the user sets the normal vector, holes in the part are highlighted with a red outline and displayed through the GUI. The parameters for these holes are available in the GUI.

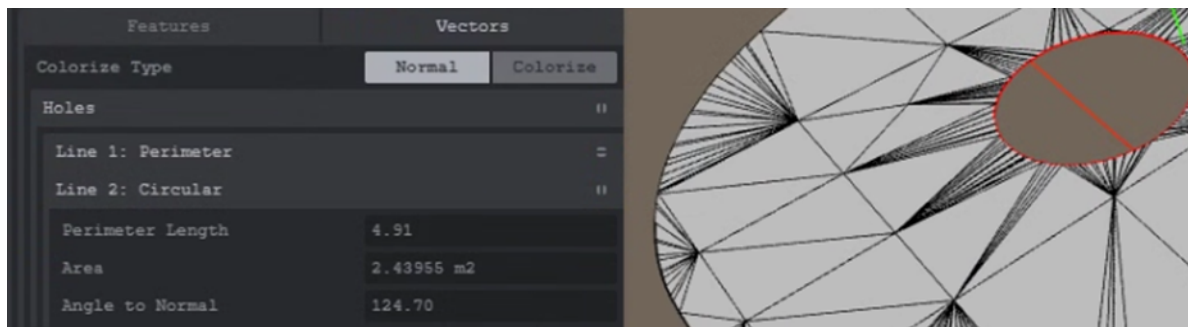


Figure 22 Operation Vectors (Holes)

The pseudo-code for these calculations is provided below:

```
while any_hole_not_detected():
    start_point, end_point = find_hole_start_line()

    queue = [end_point]
    hole = [[start_point, end_point]]

    while len(queue) > 0:
        point = queue.pop()
        point_start = pointer_list[point]
        point_end = pointer_list[point + 1]

        for neighbor_point in get_neighbor_points(point_end):
            if is_line_on_edge(point_end, neighbor_point):
                hole.push([point_end, neighbor_point])
                queue.push(neighbor_point)

    all_holes.push(hole)
```

R/L Symmetry: Through this tab, users can view and reposition the symmetrical version of a part. They can also use the previously mentioned view controller to observe the part from various angles.

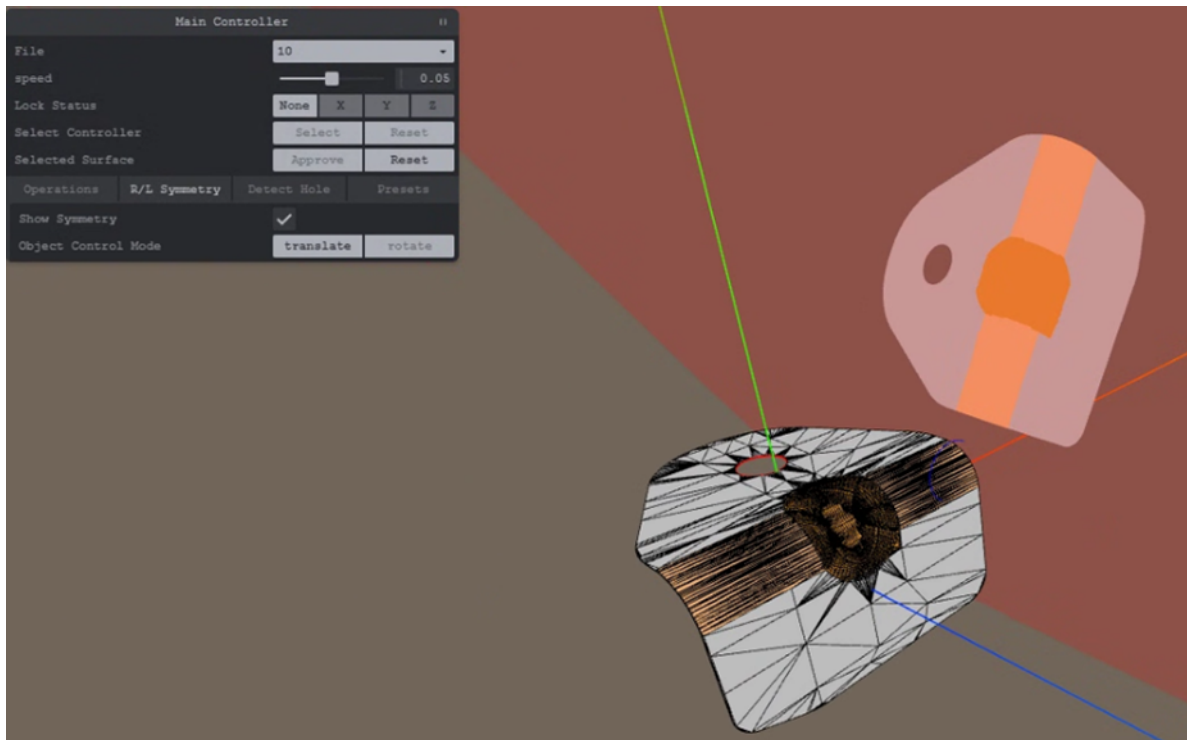


Figure 23 Example of R/L Symmetry

Camera: If the user loses or moves the part too far away, they can reset the camera position using the "Reset Camera" button.



Figure 24 General Operations

Part Selection Page

The part selection page allows users to easily find and choose parts while creating or applying presets. After selecting a part, a preview is displayed on the right side. If any presets are available for that part, users can select them, or they can create a new preset if needed. Once a preset is created or selected, users can click the navigation button to proceed to the main ThreeJS page.

ThreeJS

Select a part to work on it
Figure 25 Part Selection Page**Hole Closing Algorithm / Page / GUI**

Detecting holes in parts has been a significant issue for Ermetal in this project. Knowing which operation will create a hole, as well as the area, perimeter, and normal angles of the holes, makes it easier for Ermetal users to process the part. To address this issue, a hole closing algorithm has been developed and optimized. The pseudo-code for the optimized hole detection algorithm is as follows:

```

var point = queue.shift()
var point_start = pointer_list[point]
var point_end = pointer_list[point + 1]

for (let i = point_start; i < point_end; i++) {
  if (point_neighbor_type_list[i] === 1 && point_neighbor_used[i] === 0) {
    point_neighbor_used[i] = 1
    var neighbor_point = point_index_neighbors_list[i]
    var neighbor_point_start = pointer_list[neighbor_point]
    var neighbor_point_end = pointer_list[neighbor_point + 1]

    for (let j = neighbor_point_start; j < neighbor_point_end; j++) {
      if (point_index_neighbors_list[j] === point) {
        point_neighbor_used[j] = 1
      }
    }
    hole.push([point, neighbor_point])
    all_holes_points_list[all_holes_points_list.length - 1].add(point)
    all_holes_points_list[all_holes_points_list.length - 1].add(neighbor_point)
    queue.push(neighbor_point)
    break
  }
}
}

```

Each hole has its own controller, shown in the blue area below, where users can view details and toggle the hole on or off. Additionally, the "Zoom In" button allows users to inspect the hole more closely. If the hole is not real (e.g., caused by a mesh error), users can remove it by clicking the "Not a Hole" button. Optimized the hole detection algorithm to run faster after completion.

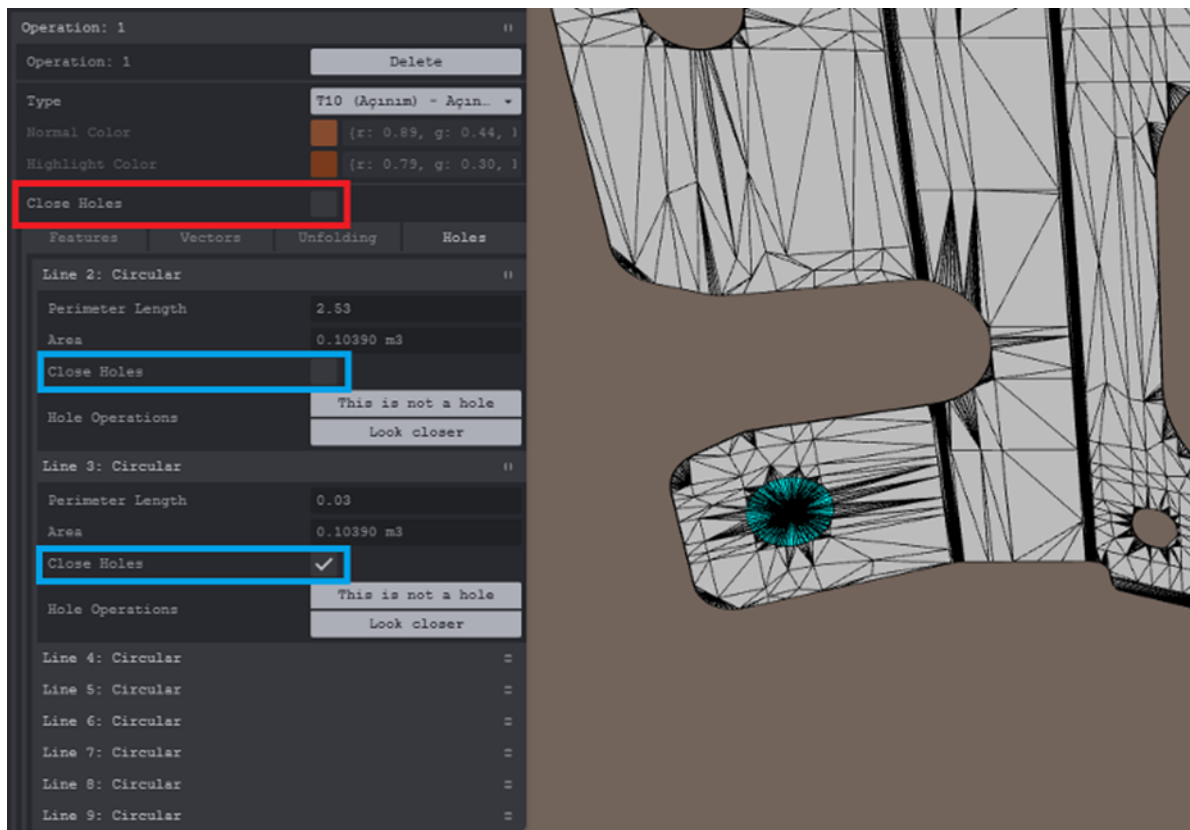


Figure 26 GUI Holes

Unfolding Algorithm / Page / GUI

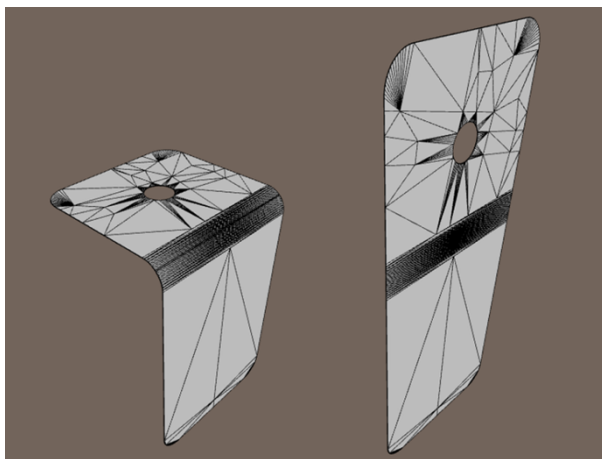
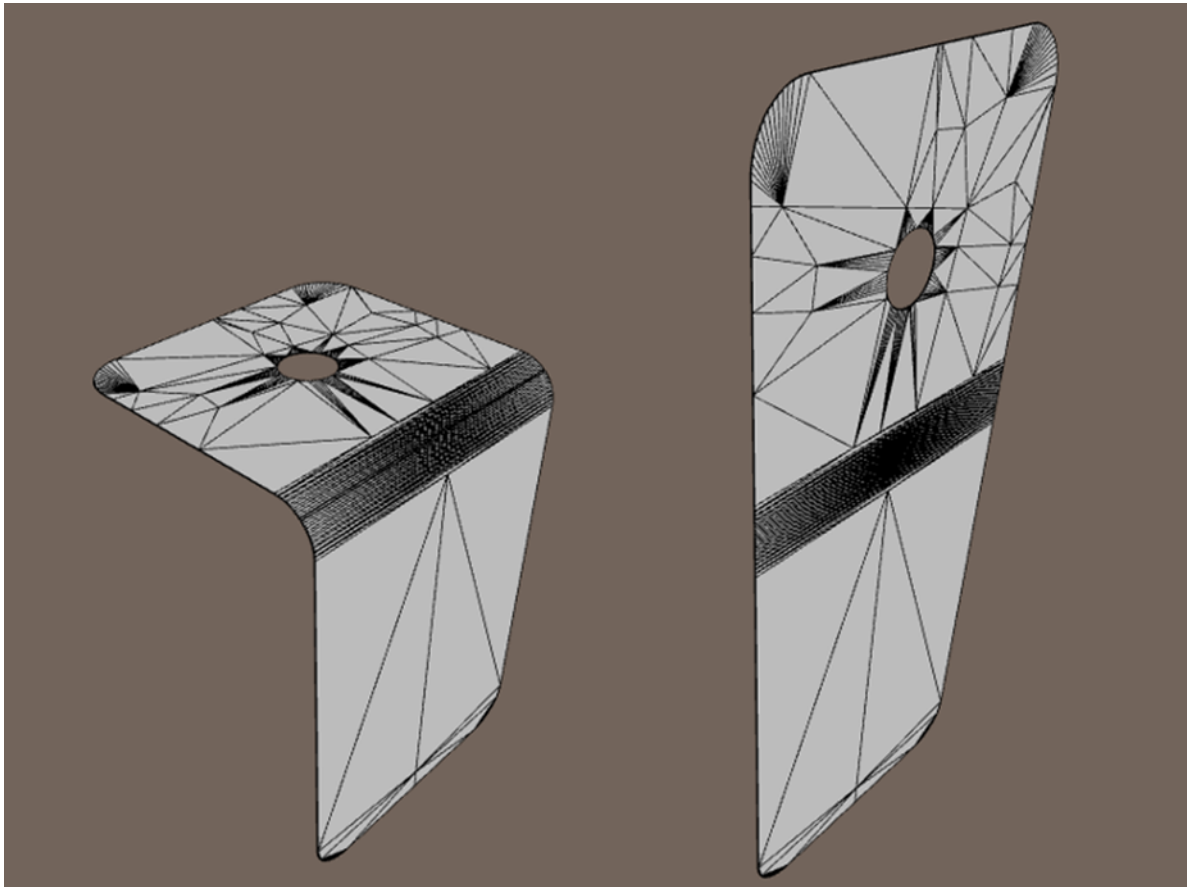


Figure 27 Unfolding



Another important topic for Ermetal has been the unfolding process, which determines how much sheet metal is needed for a part. In the initial unfolding algorithm, simple parts could be processed efficiently. However, as the number of triangles in a part increased, the unfolding time also increased. To address this, the algorithm was optimized to eliminate time constraints. This optimization allows the algorithm to handle more complex parts efficiently, regardless of the number of triangles.

The pseudocode for the optimized unfolding algorithm is as follows:

```
while (queue.length > 0) {
  const surface_index_A = queue.shift()
  const nb_list = this.unfold_data.surface_neighbors[surface_index_A]

  nb_list.forEach(nb_index => {
    if (nb_index !== index_1 && nb_index !== surface_index_A && !connected_surface_indexes.includes(nb_index)) {
      queue.push(nb_index)^
      connected_surface_indexes.push(nb_index) }}}}
```

Form Calculation

After the forms used by Ermetal were defined in the system, a form calculation system was developed that calculates which form the parts are suitable for based on the area they cover as a result of each operation and presents this visually. The formula given by Ermetal was used during the calculation. The calculations and their results are displayed on the Tweakpane GUI. An example calculation is presented below. And below the image there is a pseudo code of calculation.

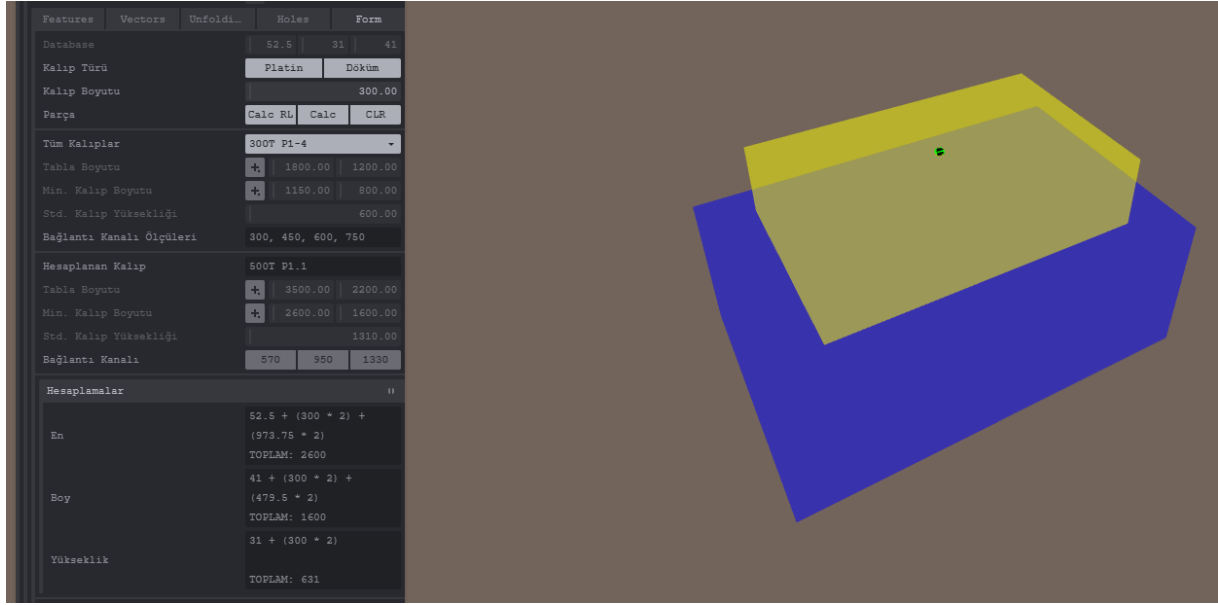


Figure 28 Form Calculation

```
def form_calculation(only_clean):
    if only_clean:
        remove_previous_elements()
        refresh_UI()
        return

    remove_object("form_boundingbox")
    remove_object("form_minimumform")
    remove_object("form_tabla")
    remove_object("form_form")
    remove_object("form_plane1")
    remove_object("form_plane2")

    form_size = get_form_size()
    xDist = calculate_x_distance(form_size)
    yDist = calculate_y_distance(form_size)
    zDist = calculate_z_distance(form_size)

    compatible_forms = find_compatible_forms(xDist, yDist, zDist)

    if not compatible_forms:
        show_alert("No compatible form found for this part.")
        return

    selected_form = select_minimum_index_form(compatible_forms)
    update_UI_with_form(selected_form)

    xDist, xAdd = adjust_x_distance(selected_form, xDist)
    zDist, zAdd = adjust_z_distance(selected_form, zDist)

    add_form_details_to_UI(selected_form)
    add_connection_channel_options_to_UI(selected_form)

    bounding_box = create_bounding_box(xDist, yDist, zDist)
```

```
minimum_form = create_minimum_form(bounding_box)
form = create_form(bounding_box)
plate = create_plate(form)

add_to_scene(bounding_box, minimum_form, form, plate)
refresh_UI()
```

Similarity Algorithm / Page / GUI

To create a new part offer, Ermetal references the most similar parts to the one being proposed, making similarity a key aspect. The **Similarity Page** has been designed with a more compact and user-friendly layout. Users can toggle the visibility of part triangles using the "Show/Hide Triangles" button. Additionally, operation filtering allows users to display only parts that include the selected operations.

Part ID
10

Calculation Type
☒ Feature ☐ Geometry

☐ Use Filters

−

Sac Kalınlığı

3

+

3

Net X

203

Net Y

66

Kontur

53

Yüzey Alanı

2901

Operations

AÇINIM KESME

FORM VERME&BÜKME

CALCULATE

SEE ALL SIMILAR PARTS

ID	Benzerlik Oranı	Sac Kalınlığı	NetX	NetY	Kontur	Yüzey Alanı
1569	%99.9546	2	62	51	205	2761
2709	%99.9406	3	85	46	248	3734
1770	%99.9273	3	92	82	296	5583
1188	%99.9143	2	87	54	299	4519
1651	%99.9098	2	93	45	253	3081

1 row selected
1-5 of 872

☒ Hide Triangles

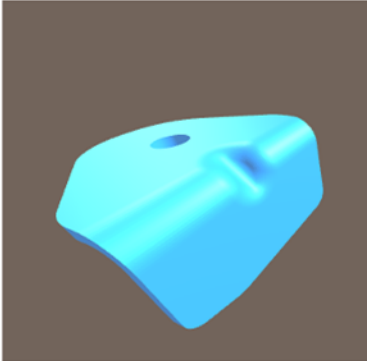
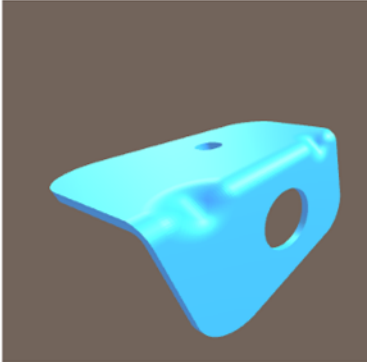



Figure 29 Similar Part Finder Page

2.5.5 API Documentation

ThreeJS Controls ThreeJS Actions

GET	/get_all_part_processes	Gets All Part Processes	🔒	✓
POST	/get_one_part_processes	Gets One Part Processes	🔒	✓
POST	/create_part_preset_to_db	Creates Part Preset to Database	🔒	✓
POST	/load_part_preset_from_db	Loads Part Preset from Database	🔒	✓
POST	/save_part_preset_to_db	Saves Part Preset to Database	🔒	✓
POST	/delete_part_preset_from_db	Deletes Part Preset from Database	🔒	✓

3 Conclusion

The implementation and testing of the Knowledge Acquisition Component mark a significant milestone in the InnoSale project. By addressing the complexities of sales and planning for industrial products, the KAC demonstrates its potential to transform traditional workflows into streamlined, AI-driven processes.

The comprehensive documentation provided in this deliverable ensures that stakeholders, including developers and integrators, can effectively utilize, maintain, and extend the functionalities of the KAC. Each subcomponent contributes uniquely to the overall system,

offering tools for rule-based reasoning, ontology visualization, factory load management, and 3D model collaboration.

The successful testing phase highlights the robustness and reliability of these tools in real-world scenarios. Moving forward, the integration of the KAC within industrial operations promises enhanced efficiency, informed decision-making, and improved adaptability to evolving market demands. This achievement underlines the pivotal role of AI-driven knowledge systems in driving innovation across industrial domains.

4 Abbreviations

WASM	Web Assembly
CSV	Comma Separated Values
RDF	Resource Description Framework
FCLE	Fuzzy Control Language Engine
GUI	Graphical User Interface
IEC61131-7	International Electrotechnical Commission Standard for Fuzzy Control Programming
KAC	Knowledge Acquisition Component