



# FINAL DAI-DSS PROTOTYPE, DOCUMENTATION AND TEST REPORT

D4.3

<b>Editor Name</b>	Marlene Mayr (BOC)
<b>Submission Date</b>	February 28, 2025
<b>Version</b>	1.0
<b>State</b>	FINAL
<b>Confidentially Level</b>	PU



Co-funded by the Horizon Europe  
Framework Programme of the European Union

# EXECUTIVE SUMMARY

This deliverable, “D4.3 – Final DAI-DSS Prototype, Documentation and Test Report”, provides a comprehensive overview of the final implementation of the DAI-DSS. Building on the foundations established in “D4.1 – DAI-DSS Architecture and Initial Documentation and Test Report” and “D4.2 – Initial DAI-DSS Prototype”, this document details the integration, orchestration, and deployment of the components for AI-based decision-making within industrial applications. The prototype is designed to enhance operational efficiency and support decision-making for various scenarios. In this last version, the DAI-DSS Prototype extends its applicability across multiple industrial use cases, including workforce allocation, production planning, machine maintenance, and validation of documents. The demonstration materials can be accessed via the following link:

<https://innovationshop.fairwork-project.eu/>

By leveraging a modular and scalable architecture, the system facilitates the interaction between AI services, UIs, and structured data repositories. The implementation of DAI-DSS consists of several integrated building blocks:

The **DAI-DSS User Interface** collects multiple UI components for the different scenarios and AI services to enable stakeholders to visualize data, interact with decision-making tools, and monitor industrial workflows.

The **DAI-DSS Orchestrator** component serves as the central coordination engine, managing workflows, microservices, and AI-driven recommendations to ensure the system operation. It includes different approaches that range from centralized to decentralized prototypes.

The **DAI-DSS Configurator** consists of a tool designed to enhance decision support systems through configuration and integration frameworks. It consists of the Configuration Framework, which assists in creating decision models and strategies, and the Configuration Integration Framework, which generates system configurations. It allows for microservices and workflow configuration, featuring an interface with a wizard for UI components combination.

The **DAI-DSS Knowledge Base** is highlighted as a central data repository, storing user properties, sensor data, and processed data. It plays a key role in the system's data flow, integrating with the Configurator and using REST API for data retrieval.

The **DAI-DSS AI Enrichment** incorporates various decision-making techniques and AI services, including neural networks, decision trees, constraint programming, Multi-Agent Systems (MAS), Large Language Model (LLM), and retrieval-augmented generation (RAG), to provide tailored recommendations and automation support.

The final DAI-DSS Prototype delivers several advancements over previous iterations by 1) supporting AI-driven decisions that aim to enhance decision-making and information access with different AI techniques while including reflections on AI and data reliability, 2) ensuring scalability and adaptability of the system by its component-based architecture that allows for integration with various applications and expansion into new domains, 3) advancing data utilization and processing with efficient storage, retrieval, and processing of industrial data, in the Knowledge Base and Vector Databases and 4) proposing a flexible approach to enable the extension with new prototypes.

The DAI-DSS marks a step forward in AI-powered decision support for industrial environments. Its modular and scalable architecture provides a foundation for future AI enhancements, data integration, and broader enterprise adoption. In particular, the results and prototypes aim to be used as starting point for use cases in the area of robots in manufacturing settings for example supporting decisions in maintenance or optimal robot-task and line allocation. Furthermore, findings and implementations documented in this deliverable contribute to advancing intelligent, and effective decision-support solutions in industrial ecosystems, and aim to contribute to future European AI research and reference architectures.

## PROJECT CONTEXT

<b>Workpackage</b>	WP4: Development of DAI-DSS
<b>Task</b>	T4.1: Architecture, Documentation and Testing T4.2: Development of DAI-DSS Orchestrator T4.3: Development of DAI-DSS Configurator T4.4: Development of DAI-DSS Knowledge Base T4.5: Enrichment of DAI-DSS with AI Algorithms
<b>Dependencies</b>	WP2, WP3, WP5

## Contributors and Reviewers

<b>Contributors</b>	<b>Reviewers</b>
Herwig Zeiner, Lucas Paletta, Julia Tschuden, Michael Schneeberger (JR) Gustavo Vieira, Rui Fernandes (MORE) Johanna Lauwigi, Alexander Nasuta (RWTH) Damiano Falcioni, Marlene Mayr (BOC) Christian Muck (OMiLAB) Rishyank Chevuri (JOTNE)	Hans Zhou (RWTH) Rishyank Chevuri (JOTNE) Damiano Falcioni (BOC)

**Approved by:** Robert Woitsch [BOC], as FAIRWork coordinator

## Version History




<b>Version</b>	<b>Date</b>	<b>Authors</b>	<b>Sections Affected</b>
1.0	February 28, 2025	FAIRWork Consortium	ALL

# Copyright Statement – Restricted Content

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

This is a restricted deliverable that is provided to the community under the license Attribution-No Derivative Works 3.0 Unported defined by creative commons <http://creativecommons.org>

You are free:

	to share within the restricted community — to copy, distribute and transmit the work within the restricted community
<b>Under the following conditions:</b>	
	Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
	No Derivative Works — You may not alter, transform, or build upon this work.

**With the understanding that:**

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. This is a human-readable summary of the Legal Code available online at:

<http://creativecommons.org/licenses/by-nd/3.0/>

# TABLE OF CONTENT

- 1 Introduction ..... 10
  - 1.1 Purpose of the Document ..... 10
  - 1.2 Document Structure ..... 10
  - 1.3 Change History ..... 10
- 2 FAIRWork Use Cases..... 12
  - 2.1 Assist Decisions about Fair Worker Allocation..... 13
  - 2.2 Assist Decisions about Production Planning..... 13
  - 2.3 Assist Decisions for Truck Loading ..... 14
  - 2.4 Improve Information Access to Support Maintenance..... 14
  - 2.5 Enhance Documentation, Validation and Information Access..... 15
    - 2.5.1 Improve Reliability of “Documentation about Quality Check” ..... 15
    - 2.5.2 Support Validation of Calibration Documents ..... 15
    - 2.5.3 Improve Information Access to Cleanroom Compliance Requirements..... 16
- 3 Building Blocks and their Integration..... 17
  - 3.1 Integrating User Interfaces..... 18
    - 3.1.1 Order Overview UI Component ..... 19
    - 3.1.2 Worker Overview UI Component ..... 19
    - 3.1.3 Allocation Proposal through AI Resource Allocation Service UI Component..... 20
    - 3.1.4 Production Planning Service UI Component..... 22
    - 3.1.5 Truck Loading UI Component..... 23
    - 3.1.6 Machine Maintenance UI Component..... 24
    - 3.1.7 Document Transformation UI Component ..... 25
    - 3.1.8 Document Compliance UI Component ..... 26
    - 3.1.9 Calibration Document Validation UI Component ..... 28
    - 3.1.10 Outlook ..... 30
  - 3.2 Orchestration of Microservices..... 30
    - 3.2.1 Workflow-based Orchestration ..... 30
    - 3.2.2 Multi-Agent Orchestration ..... 36
    - 3.2.3 Outlook ..... 41
  - 3.3 Integrating Configuration..... 41
    - 3.3.1 Multi-Agent Orchestrator Configuration ..... 41
    - 3.3.2 Configuration Framework ..... 42
    - 3.3.3 Configuration Integration Framework..... 44
    - 3.3.4 Outlook ..... 47

3.4	Integrating the Knowledge Base .....	47
3.4.1	Reliability .....	51
3.4.2	Outlook .....	54
3.5	Integrating AI-Services.....	54
3.5.1	Support the Understanding of Decisions through Conceptual Modelling.....	54
3.5.2	Decision Support through Decision Tree .....	64
3.5.3	Resource Allocation using Neural Networks.....	67
3.5.4	Resource Allocation using Linear Sum Assignment Solver .....	68
3.5.5	Production Planning Service with a Hybrid Approach.....	70
3.5.6	Resource Allocation MAS-based .....	72
3.5.7	Truck Loading Service .....	79
3.5.8	Support Machine Maintenance using RAG and LLM.....	80
3.5.9	Document Transformation using LLM.....	83
3.5.10	Support Compliance for Clean Room using RAG and LLM.....	86
3.5.11	Calibration Certification Service.....	88
3.6	Real World Data Providers.....	89
3.6.1	Intelligent Sensor Boxes .....	89
3.7	Summary of the DAI-DSS Building Blocks.....	92
4	Prototype Deployment .....	95
4.1	Deployment of Configuration Integration Environment, Workflow Engine and User Interfaces .....	97
4.2	AI-services Deployment .....	99
4.2.1	Decision Service Sever 1 and 2.....	99
4.2.2	AWS-deployed AI-services .....	101
4.3	Cost Factors for LLM Deployment .....	101
4.4	Knowledge Base Deployment.....	102
5	Extending DAI-DSS for New Use Case Scenarios .....	103
5.1	Extending Workflows and User Interfaces .....	103
5.2	Extending Decision Services through Conceptual Modelling .....	103
5.3	Extending DAI-DSS Capabilities through AI-Enrichment Services.....	104
5.4	Extending Real-World Data Provisioning .....	105
5.5	Extending the Knowledge Base .....	105
6	Summary, Conclusion and Outlook .....	106
7	References .....	108

# LIST OF FIGURES

Figure 1: Overview of DAI-DSS High-Level Architecture.....	17
Figure 2: Data Flow in DAI-DSS Framework.....	18
Figure 3: Order Overview UI Component.....	19
Figure 4: Worker Overview and Manual Assignment UI Component.....	20
Figure 5: Multi-Agent Allocation Service.....	21
Figure 6: LinSumSolver Allocation Service.....	21
Figure 7: Rule-based Allocation Service.....	22
Figure 8: Production Planning Service.....	22
Figure 9: 2D Verison of Truck Loading UI Component.....	23
Figure 10: 3D Version of Truck Loading UI Component.....	23
Figure 11: Machine Maintenance Indexing UI.....	24
Figure 12: Machine Maintenance Query UI.....	25
Figure 13: Document Transformation UI.....	26
Figure 14: Document Transformation Model Result.....	26
Figure 15: Document Compliance Indexing UI.....	27
Figure 16: Document Compliance Information Extraction Process.....	27
Figure 17: Document Compliance Query UI.....	28
Figure 18: Calibration Validation UI Output.....	29
Figure 19: Workflow Engine Control Page.....	30
Figure 20: Test Call of a Microservice Operation for retrieving Data from the Knowledge Base in the Microservice Controller.....	31
Figure 21: Example of a Workflow Definition for retrieving Information from the Knowledge Base.....	32
Figure 22: Workflow Workbench.....	32
Figure 23: Workflow Execution Details.....	33
Figure 24: Example of an API Call for triggering a Workflow in Postman.....	34
Figure 25: Simplified Workflow Definition.....	34
Figure 26: Execution Path of Workflow "dataset_crf_workloadbalance_kb".....	36
Figure 27: Available Endpoints to Registered Users.....	37
Figure 28: Orchestrator Endpoints.....	37
Figure 29: Login Endpoint Usage.....	38
Figure 30: Successful Login Response.....	38
Figure 31: Login Endpoint Data Models.....	38
Figure 32: Microservices Allocation Results Endpoint Usage.....	39
Figure 33: Generic Parameters Structure to be used in the GET Request.....	39
Figure 34: Output Model.....	39
Figure 35: Production Planning Microservice Data Model.....	40
Figure 36: MAS Orchestration of Microservices.....	41
Figure 37: Required Authorization Header.....	41
Figure 38: POST Request Parameters.....	42
Figure 39: Accessing Login Page of Modelling Tool.....	43
Figure 40: Overview Experiment for Decision Service Configuration.....	43
Figure 41: OLIVE instance showing the Configuration Environment.....	45
Figure 42: Example of Microservice Definition.....	46
Figure 43: Configuration Steps of creating a Web Application.....	47
Figure 44: Sequence Diagram for DAI-DSS Interactions Part 1.....	49
Figure 45: Sequence Diagram for DAI-DSS Interactions Part 2.....	50

Figure 46: Reference Data Library Definition .....	51
Figure 47: Breakdown Structure of the CRF Use Case .....	52
Figure 48: Node System Properties – Metadata.....	52
Figure 49: Document System Properties - Metadata.....	53
Figure 50: FLEX Data stored in the Knowledge Base .....	54
Figure 51: OLIVE User Interface for Resource Allocation Experiment .....	56
Figure 52: OLIVE Interface for Testing the Line Assignment Assessment Operation .....	56
Figure 53: DMN Model for the Worker Allocation Use Case.....	57
Figure 54: Example Decision Table.....	58
Figure 55: OLIVE Web Interface.....	59
Figure 56: OLIVE Microservice Configuration Interface .....	60
Figure 57: Example DMN Model with Microservice Definition Object.....	60
Figure 58: Attributes of the Microservice Definition Modelling Concept.....	62
Figure 59: OLIVE Controller Interface .....	63
Figure 60: OLIVE Test Interface .....	63
Figure 61: Triggering Decision Tree Functionality in the Modelling Tool .....	65
Figure 62: Example of Decision Tree in the Modelling Tool .....	66
Figure 63: Neural Network Resource Allocation.....	67
Figure 64: Historic Data.....	68
Figure 65: Problem Instance in Linear Sum Assignment Graph Representation .....	69
Figure 66: Extract of Production Plan (one line from several, two days from two weeks) .....	71
Figure 67: MAS-based Worker Allocation Dynamic.....	73
Figure 68: UML Sequence Diagram of the Multi-Agent-based Workload Balance .....	73
Figure 69: Worker Allocation Agent Message Exchange Example.....	74
Figure 70: Watchdog Raising the Alarm.....	76
Figure 71: Renegotiation to Comply with the Watchdog.....	76
Figure 72: UML Sequence Diagram Updated for Renegotiation with the Watchdog Agent.....	77
Figure 73: Watchdog Message Exchange after Renegotiation.....	78
Figure 74: Interface for the Agent-based Service .....	79
Figure 75: Concept for Machine Maintenance .....	81
Figure 76: RAG Concept for External Knowledge .....	81
Figure 77: Workflow for Machine Maintenance Prototype .....	83
Figure 78: Document Transformation Concept.....	84
Figure 79: Sample Output of Document Transformation .....	85
Figure 80: Model-based workflow for Document Transformation Prototype.....	85
Figure 81: Document Comparison Concept using Vectors.....	87
Figure 82: FAIRWork Resilience Monitor, with a Sample Number of Estimations of Physiological (blue) and Cognitive-emotional (red) Strain during a Time Course of 20 Days of a Potential Worker .....	90
Figure 83: Deployment Diagram DAI-DSS Prototype.....	95
Figure 84: UI Deployment and Registration.....	99
Figure 85: Available Endpoints to Registered Users .....	101
Figure 86: Knowledge Base Internal Architecture .....	102



# LIST OF TABLES

Table 1: Overview of FAIRWork Use Cases..... 12

Table 2: Orchestration Workflows ..... 35

Table 3: Summary of DAI-DSS Building Blocks ..... 92

Table 4: Summary of the AI Services ..... 93

Table 5: Overview of the Individual Deployable Components ..... 97

Table 6: Deployment Components of Configuration Environment..... 98

# LIST OF ABBREVIATIONS

Abbreviation	Meaning
AI	Artificial Intelligence
ANN/ NN	Artificial Neural Networks
CP	Constraint Programming
DAI-DSS	Democratic Artificial Intelligence – Decision Support System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HW	Hardware
LLM	Large Language Model
MAS	Multi-Agent System
MCTS	Monte Carlo Tree Search
ML	Machine Learning
RL	Reinforced learning
RAG	Retrieval Augmented Generation
SSH	Secure Shell
SSL	Secure Sockets Layer
SW	Software
UI	User Interface

# 1 INTRODUCTION

---

## 1.1 Purpose of the Document

This document provides the final implementation and prototype of the DAI-DSS of the FAIRWork project. Based on the architecture proposed in „D4.1 – DAI-DSS Architecture and Initial Documentation and Test Report“ and the initial prototype in “D4.2 – Initial DAI-DSS prototype” this deliverable captures and documents the updates and advances resulting in the final prototype highlighting how the technologies explored in the project are integrated to the DAI-DSS.

The main objective is to outline the final status of the methodologies and processes involved in developing and implementing User Interfaces (UI), Orchestration, Configuration, access to Knowledge Base, and AI services within the DAI-DSS framework. The document intends to be a comprehensive guide for understanding how the system operates including AI service integration, the role of conceptual modelling in decision-making, and deployment strategies. It is aimed at stakeholders interested in the composition of the DAI-DSS, its functionalities, and how it can be used in industry to support decision-making.

## 1.2 Document Structure

To provide a comprehensive overview of the final prototype, the document is organized into multiple key sections ranging from a description of the use cases and problem setting to the development of the individual components of the final DAI-DSS prototype.

As the prototype is extended with new services for other use cases besides “Worker Allocation”, in Section 2, a recap of the use cases provided by FLEX and CRF is given. For understanding each use case is described by a short overview of the problem setting and the proposed DAI-DSS solution.

In Section 3, the implementation of the architecture’s building blocks is approached. Starting with the UIs describing the creation and combination of UI components. Then the implementation of the DAI-DSS Orchestrator, Configurator, Knowledge Base, and AI Enrichment including the reflection on AI service and data reliability are presented. In Section 4, documentation on deploying the various prototypes, including AI services and UIs, is given. This section covers the deployment of a modelling tool, experimental services, and the management and hosting of AI services on servers.

In Section 5, some ideas for extending the DAI-DSS capabilities, and in Section 6 current developments and the final status of the prototype are summarized and future directions are presented in the outlook.

## 1.3 Change History

All components described in D4.2 are included in D4.3 for completeness and readability reasons. Main changes and updates of D4.3 are attributed to 1) the extension with further use cases besides worker allocation as well as with additional AI algorithms, 2) the advances in the implementation and deployment of the distinctive building blocks of the architecture, like the UI, Configuration, Orchestration or Knowledge Base and 3) reflections about reliability of AI and Data. Below the change history is presented differentiating between “updated components from D4.2” and “newly added” ones.

The whole of **Section 2** was added newly to recall the challenges of the existing use cases and to introduce the new scenarios targeted by different AI services. **Section 3** includes updates of the distinctive building blocks detailing the final versions of the UIs, the Orchestration, the Configurations, and the Knowledge Base. Additionally,

AI services were added to broaden the focus of the initial DAI-DSS prototype from a single to multiple use cases. The reliability sections were added as well. The following components were either updated from D4.2 or added newly:

- Section 3.1: Integrating User Interfaces
  - 3.1.1 Order Overview UI Component (**updated from D4.2**)
  - 3.1.2 Worker Overview UI Component (**updated from D4.2**)
  - 3.1.3 Allocation proposal through AI Resource Allocation Service UI Component (**updated from D4.2**)
  - 3.1.4 Production Planning Service UI Component (**new**)
  - 3.1.5 Truck Loading UI Component (**new**)
  - 3.1.6 Document Transformation UI Component (**new**)
  - 3.1.7 Machine Maintenance UI Component (**new**)
  - 3.1.8 Document Compliance UI Components (**new**)
  - 3.1.9 Calibration Document Validation UI Component (**new**)
- Section 3.2: Orchestration of Microservices
  - 3.2.1 Workflow-based Orchestration (**updated from D4.2**)
  - 3.2.2 Multi-Agent Orchestration (**updated from D4.2**)
- Section 3.3: Integrating Configuration
  - 3.3.1 Multi-Agent Orchestrator Configuration (**new**)
  - 3.3.2 Configuration Framework (**updated from D4.2**)
  - 3.3.3 Configuration Integration Framework (**updated from D4.2**)
- Section 3.4: Integrating the Knowledge Base (updated from D4.2)
- Section 3.5: Integrating AI services
  - 3.5.1 Support the Understanding of Decisions through Conceptual Modelling (**updated from D4.2**)
  - 3.5.2 Decision Support through Decision Tree (**updated from D4.2**)
  - 3.5.3 Resource Allocation using Neural Networks (**updated from D4.2**)
  - 3.5.4 Resource Allocation using Linear Sum Assignment Solver (**updated from D4.2**)
  - 3.5.5 Production Planning Service with a Hybrid Approach (**new**)
  - 3.5.6 Resource Allocation MAS-based (**updated from D4.2 and extended with new service 3.5.6.2**)
  - 3.5.7 Truck Loading Service (**new**)
  - 3.5.8 Support Machine Maintenance using RAG and LLM (**new**)
  - 3.5.9 Document Transformation using LLM (**new**)
  - 3.5.10 Support Compliance for Clean Room using RAG and LLM (**new**)
  - 3.5.11 Calibration Certification Service (**new**)
- Section 3.6: Real-World Data Providers (updated from D4.2)
- Section 3.7: Summary of the DAI-DSS Building Blocks (new)

In **Section 4**, the updated deployment of the final DAI-DSS prototype is described and extended with new components and services. The following components were updated or changed:

- Section 4.1: Deployment of Configuration Integration Environment, Workflow Engine and User Interfaces (updated from D4.2)
- Section 4.2: AI-services Deployment
  - 4.2.1 Decision Service Sever 1 and 2 (**updated from D4.2 and new services**)
  - 4.2.2 AWS-deployed AI-services (**new**)
- Section 4.3: Cost Factors for LLM Deployment (new)
- Section 4.4: Knowledge Base Deployment (new)

**Section 5** details the DAI-DSS components and their extension for new use cases. All subsections were updated from D4.2. **Section 6** provides the updated outlook and future directions of the DAI-DSS Prototype.

## 2 FAIRWORK USE CASES

This section aims to recall the use cases and problem settings described in previous D2.1 and D5.1 and the new use cases proposed by the use case partners CRF and FLEX. Each scenario includes an overview of the current problem and the suggestion to target it through the DAI-DSS in section 3. Relevant scenarios of CRF include 1) worker allocation by assisting decisions about fair worker allocation, 2) assistance in decisions for production planning as well as 3) delay of material by assisting decisions about truck loading. Relevant scenarios for FLEX are 1) to support machine maintenance through improving information access and 2) scenarios to enhance documentation, validation and information access through a) improving the reliability of “Documentation about Quality Check”, b) validation of calibration documents and c) improving information access to cleanroom compliance requirements. The Table 1 summarizes the use cases which are covered by the AI algorithms.

Use Case	Use Case Partner	Scenario	AI-Enrichment
<b>Workload Balance</b>	CRF	Assist Decisions about Fair Worker Allocation	(1) Support Understanding of Decisions through Conceptual Modelling (2) Decision Support through Decision Tree (3) Resource Allocation using Neural Networks (4) Resource Allocation using Linear Sum Assignment Solver (5) Resource Allocation MAS-based
<b>Production Planning</b>	CRF	Assist Decisions about Production Planning	(6) Production Planning Service with a Hybrid Approach
<b>Delay of Material</b>	CRF	Assist Decisions for Truck Loading	(7) Truck Loading Service
<b>Machine Maintenance</b>	FLEX	Improve Information Access to Support Maintenance	(8) Support Machine Maintenance using RAG and LLM
<b>Document Transformation</b>	FLEX	Improve Reliability of “Documentation about Quality Check”	(9) Document Transformation using LLM
<b>Compliance with Clean Room Regulations</b>	FLEX	Improve Information Access to Cleanroom Compliance Requirements	(10) Support Compliance for Clean Room using RAG and LLM
<b>Calibration Document Certification</b>	FLEX	Support Validation of Calibration Documents	(11) Calibration Certification Service

**Table 1: Overview of FAIRWork Use Cases**

## 2.1 Assist Decisions about Fair Worker Allocation

As introduced in previous deliverables (D2.1 and D5.1) this use case deals with the efficient allocation of workers to production lines. The workers usually work in two shifts on different machines. In each shift, a fixed number of specific parts have to be produced on each machine. These parts are produced by specific tasks on the machine. Each machine produces different parts, so each machine has different tasks. Depending on the part to be produced a different number of workers is necessary for the machine. Also, it is necessary to allocate the workers depending on the shift plan and the task-to-machine allocation to the different machines. Allocating the workers to the different machines is done by single humans based on experience. This can lead to situations where the same workers are always allocated to the same tasks on the same machines. This can lead to problems if the workers feel these tasks are harder to do than others.

### DAI-DSS Solution Suggestion

The decision of the allocation is based on different factors such as the training of the worker, medical condition of the worker, resilience of the worker, preference of the worker, tasks to be done, and availability of workers. It is difficult for a human to take all these factors into account. However, technical systems with access to the corresponding data can use this data to allocate workers taking all these factors into account. Therefore, we aim for worker allocation services that take all these factors into account. Especially with the focus on the worker's preferences and resilience, we aim to create a system that is felt to be fairer as it tries to fit all the preferences and the resilience of the workers as well as possible. So that all workers are considered in the allocation. This scenario is targeted by the AI services *Support the Understanding of Decisions through Conceptual Modelling*, *Decision Support through Decision Tree*, *Resource Allocation using Neural Networks*, *Resource Allocation using Linear Sum Assignment Solver*, *Production Planning Service with a Hybrid Approach* and the *Resource Allocation MAS-based*.

## 2.2 Assist Decisions about Production Planning

The production plan aims to define when which part is produced on which line in which quantity by whom. The production plan should consider that all parts are produced close to the deadline so that they won't occupy storage space for too long while also being produced on time. In order to allow high efficiency, the production plan should avoid working overtime and also try to have lines running at full capacity during working hours. Those aims can make the planning complex. This complexity is increased by unforeseen events as workers might be absent e.g. due to sickness. Also, sudden orders or line stops lack of resources, etc. can increase the complexity and create the need to be able to adapt to changes on short notice. Especially sudden changes but also the demands from different aspects are a challenge for the people creating the plan and make adaptations on short notice difficult.

### DAI-DSS Solution Suggestion

FAIRWork solutions aim to provide a full production plan, including two things: The allocation of tasks to machines and the allocation of workers to machines based on the different factors described in section 0. The allocation of tasks to machines is based on factors such as the order list, which describes which parts have to be produced until when for which customer. Additionally, it needs to include dates as to which parts can be produced on which machine and at which time. All this together describes a flexible job shop problem that has to be solved. By solving the flexible job shop problem and allocating workers to the machines, services tackling the production planning can provide a useful addition for the plant by automatically creating production plans based on order lists, shift plans, short notice adaptations, and fixed data saved in the knowledge base. This scenario is targeted by the AI service *Production Planning Service with a Hybrid Approach*.

## 2.3 Assist Decisions for Truck Loading

As introduced in previous deliverables (D2.1 and D5.1) this use case deals with efficient shipping to customers. The shipping process aims to efficiently load materials into containers and transport them to customers. Delays in material availability, production, or shipment can result in customers receiving their orders late. Each product geometry requires specific container types for transportation. The shipping plan must consider the entire customer order, which must be dispatched daily. Crucial decisions include selecting the optimal truck type for shipment and determining how to load the containers to maximize truck capacity while ensuring that all geometries are included. The process starts by evaluating the shipment's due date. Once the shipment is ready, containers are loaded, and truck saturation is analyzed to ensure minimal free capacity. If saturation is low, alternative truck types are evaluated. When no better alternatives exist, additional geometries might be included to improve truck utilization. If this is not feasible, the customer is contacted to explore postponement options. This decision-making process aims at optimal resource allocation while addressing customer expectations and operational efficiency.

### DAI-DSS Solution Suggestion

A decision support system can streamline the shipping process by optimizing material and container allocation within trucks. This system would analyze factors like order geometries, container types, truck capacities, and shipment deadlines to create an efficient shipping plan. The tool would dynamically adjust for changes in order requirements or truck availability, reducing costs and improving customer satisfaction. This scenario is targeted by the AI service *Truck Loading Service*.

## 2.4 Improve Information Access to Support Maintenance

As already introduced in previous deliverables (D2.1 and D5.1) this use case aims to support technicians to fix machine malfunctions. Due to the diverse problem settings, the machines' errors can range from simple operator failures to complex technical issues. The time to solve the machine failures can vary between minutes to hours. For the machine issues, the technicians can look up a solution procedure in different data sources. First, there is the "Ispro-NG", the current maintenance database also used as the task ticket system that contains information on all tickets and their timestamps, machine types, duration to solve the issue, the error source, and corresponding maintenance instructions etc. Second, there is an internal "Wiki System" for maintenance engineers to collect information about issues on equipment and procedures as well as graphical descriptions of how engineers solved errors in the past. Third, official documents from the manufacturers of equipment and machines such as operating instructions, manuals, specifications and handbooks for product descriptions, safety requirements or installation guidelines are stored in a specific database. The aim is to speed up the finding of appropriate solutions and reduce the time and cost of breakdown.

### DAI-DSS Solution Suggestion

As different pieces of information on machine errors or maintenance are available across multiple data sources, the challenge is to improve the information access to support maintenance. The aim is to reduce the corrective maintenance and machine breakdowns by receiving fast and correct proposals with the AI tool. Thus, the requirement to support the decision-makers in this use case is to create a service that can access information from the above-mentioned sources. There should be bidirectional communication available to communicate in written form or with a speech module. This scenario is targeted by the AI service *Support Machine Maintenance using RAG and LLM*.

## 2.5 Enhance Documentation, Validation and Information Access

During the project, a use cases dealing with various compliance issues for organizations was identified. Therefore, three different scenarios to support FLEX in compliance activities capture the needs for supporting workers in being compliant regarding 1) transforming outdated and unstructured documents into the newly defined electronic Work Instruction system (e-WI) template, 2) supporting employees with checking calibration documents and certification and 3) help to ease history and versioning changes of official compliance regulations for their daily operations like safety instructions for the clean room.

### 2.5.1 Improve Reliability of “Documentation about Quality Check”

Different types of documents, in various document formats and at diverse majority levels including outdated and very unstructured documents can be found at FLEX. In particular, most of the station instruction cards (SICs) – documenting the instructions on how to work at the individual stations in the factory – are paper-based. The digitization of such documents is envisioned to contribute to the overall factory digitalization including the implementation of appropriate systems. FLEX works on implementing an electronic work instruction system (e-WI). Therefore, paper-based instructions have to be transformed into digital instructions which include as one major aspect the conversion into a new electronic template that is required by the e-WI application. Among the major expected benefits are the elimination of paper-based documents, improved revision control including the reduction of mistakes caused by using wrong WI revision numbers as well as an increased approval process using electronic signatures. Digitization has already started and the employees transferred the paper-based documents into Word templates, however, using old and heterogenous formats. At the moment, around 27.000 SICs are used in an old format. Manually transforming all of these documents so that they match with the new digital e-WI template requires an enormous amount of human effort. In particular challenging are two aspects, which are (1) the huge diversity and the complexity of the SICs and (2) the process of editing the instruction documents. First, SICs are specifically tailored to each station in the FLEX production process flow, and even more critical, they are customized for individual products and each product version. Second, SICs at FLEX are edited by numerous process engineers over time. Those engineers have their different style of writing documentation, editing text, and including pictures using Word files in unstructured .doc format. Not even all of them are using the same (outdated) template.

#### DAI-DSS Solution Suggestion

Here, a solution for automatic transformation to the new format can support reducing the manual conversion effort. The idea is to develop an AI application that automatically converts the SICs from the old Word templates into the new digital e-WI template to reduce human effort and errors as well as to save time. This scenario is targeted by the AI service *Document Transformation using LLM*.

### 2.5.2 Support Validation of Calibration Documents

Verification of Calibration Certificates (CC) in PDF format is currently a manual, repetitive, time-consuming and error-prone process. Each calibration certificate received from the calibration service provider must be carefully checked for any discrepancies or missing or incorrect information. This check includes many critical data points, including

- Instrument information: model, type, manufacturer and serial number/ID of the instrument being calibrated.
- Calibration dates: calibration date and calibration expiry date.
- Calibration results: the actual measured values and calibration results, including any deviations or measurement uncertainties.
- Information on measuring equipment: details of the calibration of the measuring equipment used to ensure traceability of the calibration.



- Formal aspects: signatures of the issuer and approver, sequential page numbering of the certificate and completeness of all pages.

Currently, all this information is checked manually by an employee. This process is not only time-consuming but also prone to human error due to the large number of certificates to be checked. Overlooked errors can have serious consequences, for example, if faulty measuring equipment remains in use and affects the quality of products or services.

#### **DAI-DSS Solution Suggestion**

In order to optimize this process and address the issues mentioned above, the development of an application based on an AI-based service is proposed. This application will replace the manual verification of critical information from calibration certificates with an automated solution. The AI-based service should be able to analyze the certificates and automatically generate a report listing any discrepancies, or missing or incorrect information found. This report should provide the verifier with a quick and efficient overview of the potential problems with the certificate. This scenario is targeted by the AI service *Calibration Certification Service*.

### **2.5.3 Improve Information Access to Cleanroom Compliance Requirements**

As the performance of the cleanroom production is a matter of its proper operation and regulatory adherence, many critical aspects must be considered by the staff, which directly impact the performance of the cleanroom. Although employees are informed about changes and have to confirm them, it happens very often that changes are not adhered to because they are unconsciously not taken into account. This is due to the aspect that the documents for the behavior and procedures in the cleanroom change relatively often in their regulatory requirements. Additionally, the compliance and regulation landscape is complex and not applicable to every situation. Thus, the corresponding compliance documents are available in a high number and in many versions, where different measures and trainings are applicable only for specific types of employees. Thus, the employees are challenged with keeping the overview when only working occasionally in the cleanroom.

#### **DAI-DSS Solution Suggestion**

The aim is to develop an AI service that can support employees in ensuring compliance measures when entering a clean room. The idea is to reduce human error by preventing overlooking or forgetting changes in compliance and ensuring that the latest compliance regulations are followed. For this, the idea is to use AI to compare multiple versions of documents and summarize relevant updates on the compliance guidelines to the employee before entering to help avoid misbehavior. This scenario is targeted by the AI service *Support Compliance for Clean Room using RAG and LLM*.



### 3 BUILDING BLOCKS AND THEIR INTEGRATION

In this section, the DAI-DSS Architecture, its building blocks, and the different instantiations of the use-case-specific prototypes, including different AI services, Data, or UIs are detailed. In “D2.1 – Specification of FAIRWork Use Case and DAI-DSS Prototype Report”, an outline of the initial architecture of the project (see Figure 1) is given based on the overall project objectives and requirements. Key components of the FAIRWork service framework are illustrated, and described, and their relevant features are presented. A detailed description of the initial architecture is given in “D4.1 – DAI-DSS Architecture and Initial Documentation and Test Report”, including the technical implementation of the basic core services or application-specific services and their integration methodologies discussed in “D4.2 Initial DAI-DSS prototype”. In this document, D4.3, a brief description of building block components and the communication between these components will be presented.

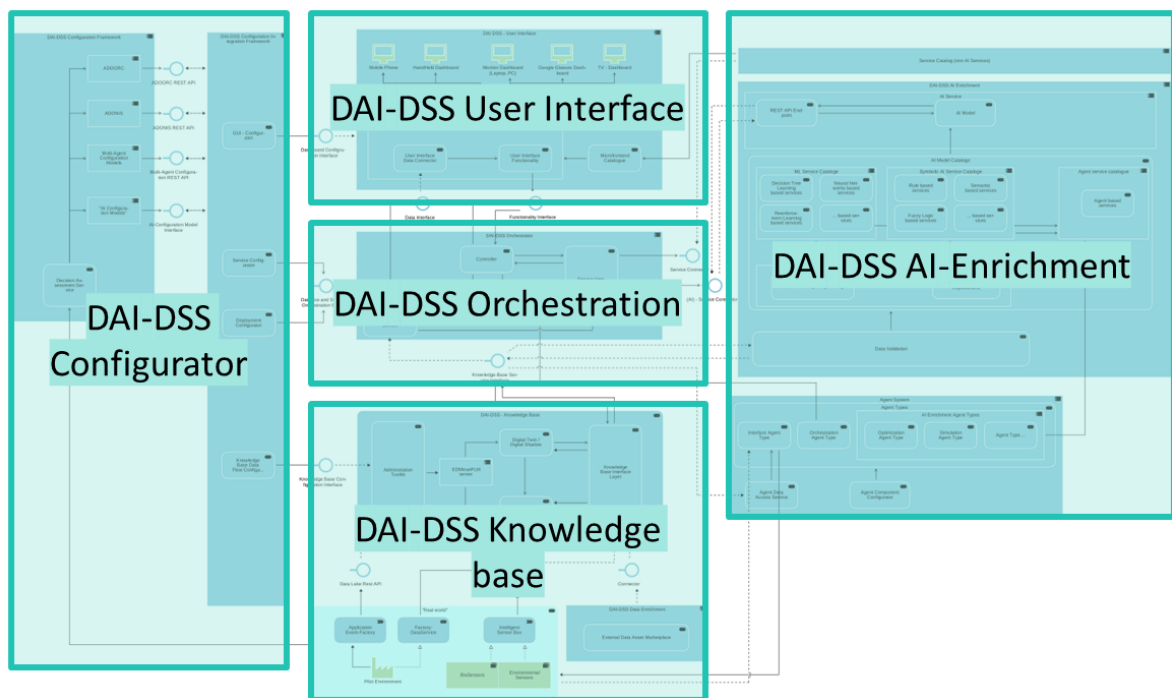


Figure 1: Overview of DAI-DSS High-Level Architecture

The DAI-DSS consists of the following key components:

1. **User Interface:** Provides clear, device-flexible displays for decision-makers, highlighting KPIs and enabling interaction in supportive environments like production monitoring.
2. **AI Enrichment:** Utilizes AI models and advanced algorithms tailored to individual agents, supporting decentralized and fair decision-making.
3. **Configurator:** Links various model environments (e.g., business processes, dashboards) to the Orchestrator and UI, consolidating data for integrated decision models.
4. **Orchestrator:** Coordinates services and workflows, ensuring seamless delivery of decision-making options.
5. **Knowledge Base:** Centralizes and organizes data from diverse sources, enabling AI learning, process optimization, and decision tracking for continuous improvement.

The DAI-DSS Framework data flow shown in Figure 2 begins with the Configurator, which aligns UIs and AI services with data from the Knowledge Base, forming workflows sent to the Orchestrator. Users interact through the interface to invoke required workflows, triggering AI or Multi-Agent services for decision support. If required, these services access additional data from the Knowledge Base via REST APIs, generate suggestions, and present results back to users with visualizations. The system ensures standardized interactions through REST APIs, while external data, like from Intelligent Sensor Boxes, flows into the Knowledge Base to enrich decision-making.

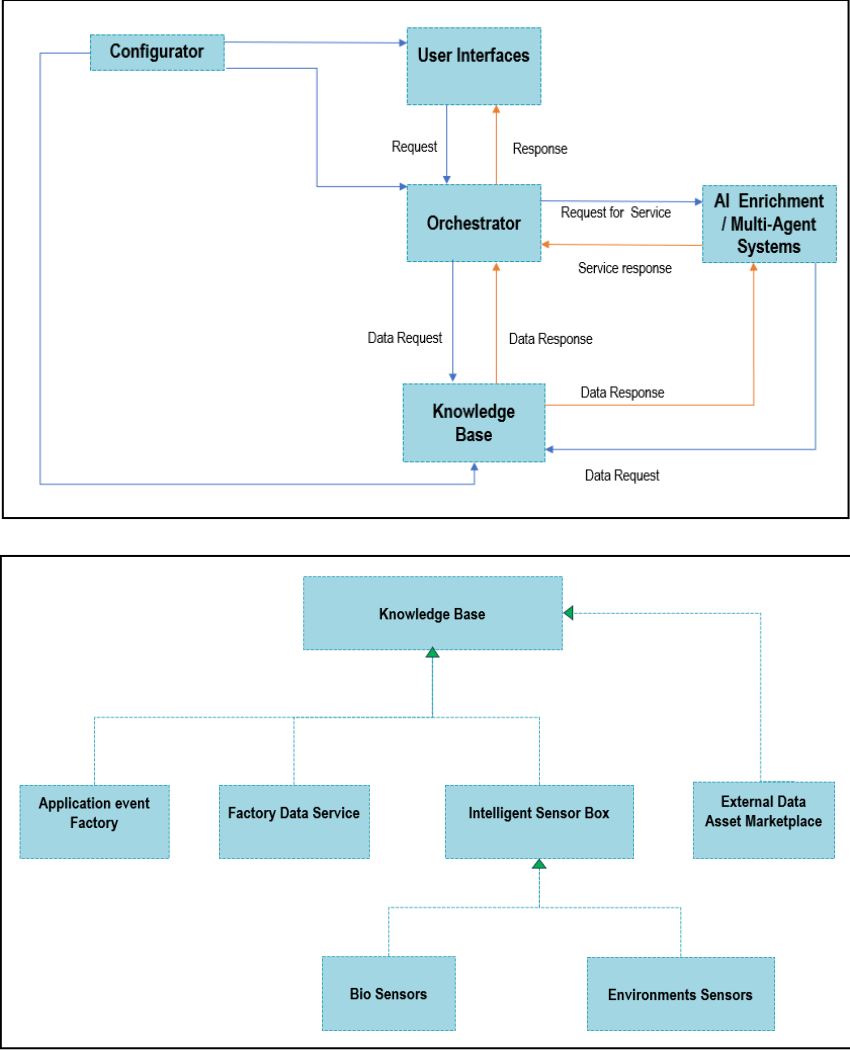


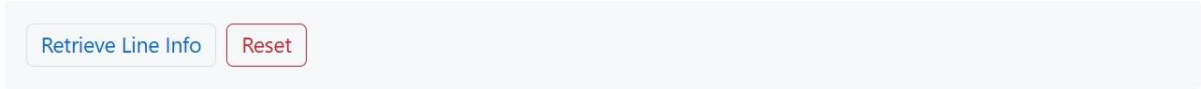
Figure 2: Data Flow in DAI-DSS Framework

### 3.1 Integrating User Interfaces

This section provides a detailed overview of the different UIs developed in the project for exposing the different DSS functionalities to the various participants in the decision process. Each interface is a single component of a micro-frontend framework that can be combined with other UI components to create web applications specific to the needed use case scenarios. Demonstration data was created based on our use case partner’s decision models and example data.

### 3.1.1 Order Overview UI Component

This UI component (Figure 3) is used for the scenario “Assist Decisions about Fair Worker Allocation” and shows the order details for each line. Clicking the “Retrieve Line Info” button, the line information is retrieved from the Knowledge Base through the orchestrator that requests the different data and combines them. Each order consists of a geometry that is produced in a specific production line and is assigned with a priority, the number of days until the due date and the number of workers required to handle the geometry.



Line	Geometry	Priority	Due Date	Required Workers
17	1343314080	True	3	6
18	6700083110	False	1	6
20	531359170	False	1	4

Figure 3: Order Overview UI Component

Since the previous report in D4.2 this component changed only the backend data alignment with the updated Knowledge Base, keeping the UI the same.

### 3.1.2 Worker Overview UI Component

This UI component (Figure 4) is used for the scenario “Assist Decisions about Fair Worker Allocation” and gives an overview of all workers relevant to the production unit and allows the manager to manually go through a list of workers and make proposals on how to allocate them to production lines. During the UI loading, the workers' info is retrieved from the Knowledge Base through multiple queries combined by the Orchestrator. After the data is loaded, the workers can be selected and the details about medical conditions for each line, line preferences, experience and resilience are visualized and enriched with color codes. After selecting a production line that fit the medical condition of the selected worker, the “Assign” button enables for the assignment of the worker to the line. A counter at the bottom of the card indicates how many workers are needed on the line and the number of assigned ones. Color codes are used to highlight if the worker preference has been respected and the total number of allocations for each line are respected.

**Manual Allocation Service**

Select a Worker and a line to perform assignment:

100023

Medical Condition: Line 17/Line 18/Line 20  
 Preference: Line 17: 0/Line 18: 0/Line 20: 0  
 UTE Experience: true  
 Resilience: 0

Line 20

Assign Reset Save Assignments

Line 17	Line 18	Line 20
ID: 100001 ✕ UTE Experience: true Resilience: 0 Preference: 0	ID: 100012 ✕ UTE Experience: true Resilience: 0 Preference: 0	ID: 100020 ✕ UTE Experience: true Resilience: 0 Preference: 0
ID: 100005 ✕ UTE Experience: true Resilience: 0 Preference: 0	ID: 100014 ✕ UTE Experience: true Resilience: 0 Preference: 0	ID: 100021 ✕ UTE Experience: true Resilience: 0 Preference: 0
ID: 100009 ✕ UTE Experience: true Resilience: 0 Preference: 0	ID: 100015 ✕ UTE Experience: true Resilience: 0 Preference: 0	ID: 100022 ✕ UTE Experience: true Resilience: 0 Preference: 0
ID: 100010 ✕ UTE Experience: true Resilience: 0 Preference: 0		ID: 100023 ✕ UTE Experience: true Resilience: 0 Preference: 0
ID: 100011 ✕ UTE Experience: true Resilience: 0 Preference: 0		
ID: 100013 ✕ UTE Experience: true Resilience: 0 Preference: 0		
Allocations: 6/6	Allocations: 3/6	Allocations: 4/4

**Figure 4: Worker Overview and Manual Assignment UI Component**

Since the previous report in D4.2, in addition to the improvement of the user experience, a “Save Assignment” button has been added that allows storing the result of the manual assignment to the Knowledge Base in the form of CSV file.

### 3.1.3 Allocation Proposal through AI Resource Allocation Service UI Component

This component is used for the scenario “Assist Decisions about Fair Worker Allocation” and allows the execution of a specific AI-based allocation service and visualizes the resulting assignment for each line, allowing to save the results in the Knowledge Base after their review. The currently supported services are the Multi-Agent Allocation Service (Figure 5), the LinSumSolver Allocation Service (Figure 6), and Rules-based Allocation Service (Figure 7). The AI allocation can be triggered using the “Trigger AI Service” button that contacts the orchestrator to execute the AI service and return the results. The different production lines are shown with the assigned workers and for each, the relevant characteristics are listed to give the decision-maker more insights.

### Multi-Agent Allocation Service

This service perform resource allocation in a multi-agent approach. Designed to support a decision process of workload balance, this service ensures a human centric balancing of tasks in a production environment.

Trigger AI Service

Reset

Save Assignments

Line 17	Line 18	Line 20
ID: 100092 UTE Experience: <b>False</b> Resilience: <b>0.32</b> Preference: 13	ID: 100098 UTE Experience: <b>False</b> Resilience: <b>0.91</b> Preference: 0.8	ID: 100093 UTE Experience: <b>False</b> Resilience: <b>0.86</b> Preference: 0.7
ID: 100064 UTE Experience: <b>False</b> Resilience: <b>0.87</b> Preference: 0.8	ID: 100054 UTE Experience: <b>False</b> Resilience: <b>0.8</b> Preference: 0.9	ID: 100029 UTE Experience: <b>False</b> Resilience: <b>0.8</b> Preference: 0.8
ID: 100077 UTE Experience: <b>False</b> Resilience: <b>0.82</b> Preference: 0.77	ID: 100051 UTE Experience: <b>False</b> Resilience: <b>0.77</b> Preference: 0.9	ID: 100106 UTE Experience: <b>False</b> Resilience: <b>0.8</b> Preference: 0.78
ID: 100100 UTE Experience: <b>False</b> Resilience: <b>0.87</b> Preference: <b>0.66</b>	ID: 100053 UTE Experience: <b>False</b> Resilience: <b>0.75</b> Preference: 0.9	ID: 100081 UTE Experience: <b>False</b> Resilience: <b>0.76</b> Preference: 0.77
ID: 100075 UTE Experience: <b>False</b> Resilience: <b>0.8</b> Preference: 0.77	ID: 100037 UTE Experience: <b>False</b> Resilience: <b>0.8</b> Preference: 0.7	
ID: 100010 UTE Experience: <b>True</b> Resilience: <b>0.92</b> Preference: <b>0.5</b>	ID: 100035 UTE Experience: <b>False</b> Resilience: <b>0.8</b> Preference: 0.7	
Allocations: 6/6	Allocations: 6/6	Allocations: 4/4

Figure 5: Multi-Agent Allocation Service

### LinSumSolver Allocation Service

This service allocates workers based on their preferences, their resilience, and the tasks. The service return an optimal solution with respect to a specific cost function that define what a 'good' allocation is.

Trigger AI Service

Reset

Save Assignments

Figure 6: LinSumSolver Allocation Service

**Rule-based Allocation Service**

This service focuses on a knowledge-based approach, allowing experienced users to encode their knowledge and make it executable. The definition of the decision knowledge is supported through conceptual modelling-based approach, which can be used as input for configuring a decision service allowing it to be integrated into more complex decision processes.

**Figure 7: Rule-based Allocation Service**

Since the previous report in D4.2, in addition to minor improvements in the UI, a “Save Assignment” button has been added that allows storing the AI service results in the Knowledge Base in the form of CSV file.

**3.1.4 Production Planning Service UI Component**

This UI component is used for the scenarios “Assist Decisions about Fair Worker Allocation” and “Assist Decisions about Production Planning” and allows visualizing the results of the Production Planning Service that performs workers' assignments across several orders for each production line. The service in this case requires as input a specifically crafted Excel file, containing the production plan. A prefilled template can be downloaded as a sample directly from the UI. As soon as the file is uploaded through the “Choose File” button, the service can be triggered via the “Trigger AI Service”. Also in this case the results can be saved in the knowledge base after review using the “Save Assignments” button. When the AI service results are returned by the orchestrator the orders in the different production lines are visualized in a timeline chart and for each order, a detail section shows the geometry involved and the workers assigned (Figure 8).

**Production Planning Service**

This service provides a full production plan based on a constraint-programming algorithm. This includes the allocation of orders/tasks to machines as well as the allocation of workers to the machines. The service needs the order list and takes the shift plan as well as worker preferences, resilience etc. into account.

Upload your schedule in Excel format. [Sample here](#)

Choose File schedule.xlsx

Experience: 0.9, Preference: 0.5, Resilience: 0.9, Transparency: medium

Line 17	Order 0 - Workers: 100023, 100066	Order 1 - Workers: 100002, 100121, 100133
Line 20	Order 1 - Workers: 100002, 100121, 100133	Line 20: 12:20 - 14:20
	12:20 :30 :40 :50 13:00 :10 :20	Duration: 2h 14:00 :10 :20

**Orders Details**

▶ Order 0

▼ Order 1

Line: Line 20  
 Geometry: 1340746080/8080  
 Workers:

- 100002
- 100121
- 100133

**Figure 8: Production Planning Service**

This UI component was not available in the previous report on D4.2.

### 3.1.5 Truck Loading UI Component

This UI component is an example for the scenario “Delay of materials” to “Assist Decisions for Truck Loading” and is used to display the arrangement of containers inside trucks, as proposed by the load optimization service. Each shipment involves the sending of around one hundred containers, to be allocated in a certain number of trucks of a particular type (called Mega, Mega high, Tractor, which have a different capacity).

In the 2D version, of the UI component, the containers loaded onto every truck are displayed from above.



Figure 9: 2D Verison of Truck Loading UI Component

In the Figure 9, the first row of containers is composed by 2 stacks of containers (type 1235, represented in red). In this case, each stack contains 3 containers. The number in brackets, next to the container type, indicates the maximum stackability of the containers, in the example cited above it is 3. In the case of the stack of containers of type 8589 (represented in turquoise), the maximum stackability is 3, but only 2 are foreseen. To highlight the lack of a container, therefore a non-optimal situation, the writing "None (3)" is shown in red.

The same truck can also be viewed in the 3D version of the UI component. The image below (Figure 10) is the equivalent of the 2D version seen from the lower right corner.

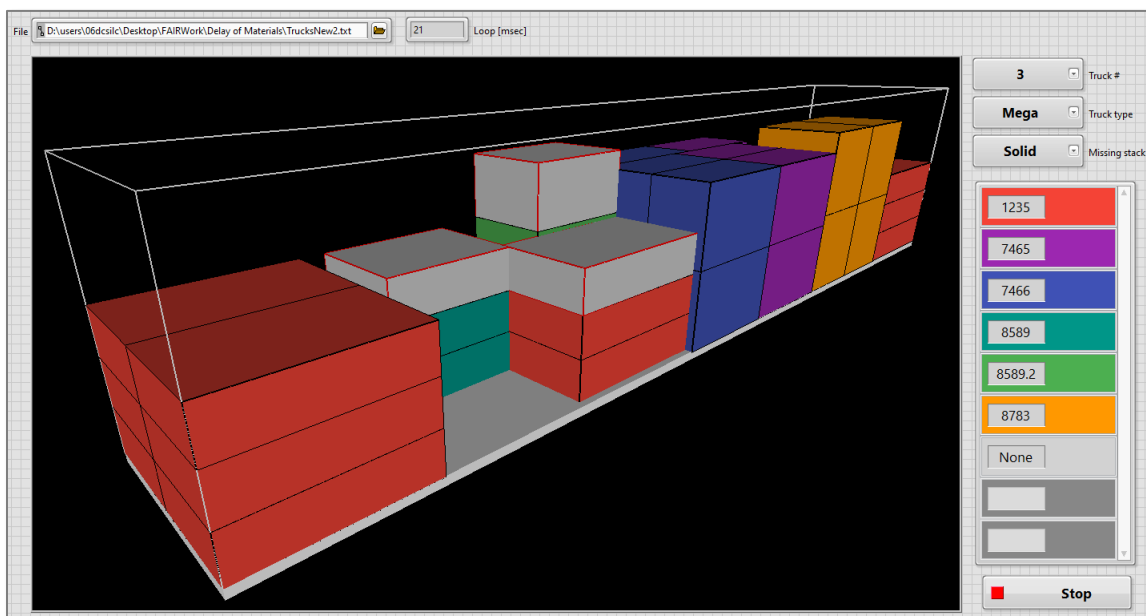


Figure 10: 3D Version of Truck Loading UI Component

In this case, the type of containers is written in the legend based on the color of the blocks. The lack of containers can be visualized with gray blocks bordered in red.

In the current release of the UI component, which is not the final one, the list of containers to be displayed on each truck (in which position) is loaded from a text file containing the information. Once the file is loaded, it's necessary to press the "Truck #" button to select the truck to display. You can also change the type of truck by selecting it from the "Truck type" button.

In the 3D version there is an additional "Missing stack" button that allows viewing the missing containers in the various stacks in different modes: "None" (they are not displayed), "Frame" (i.e. only the red border) or "Solid" (in grey with a red border). The 3D viewer also allows to rotate, move and zoom the truck shown. In both cases, to exit the UI component it's necessary to press the "Stop" button.

### 3.1.6 Machine Maintenance UI Component

This component is used for the scenario "Improve Information Access to Support Maintenance" and is composed of two UIs, one for the indexing of the different documents containing machine-specific procedures, and one for querying a specific issue and showing the results.

The UI for indexing a document allows only the upload of a document and triggering an AI that extracts relevant chunks of information and indexes them in a vector database (Figure 11).

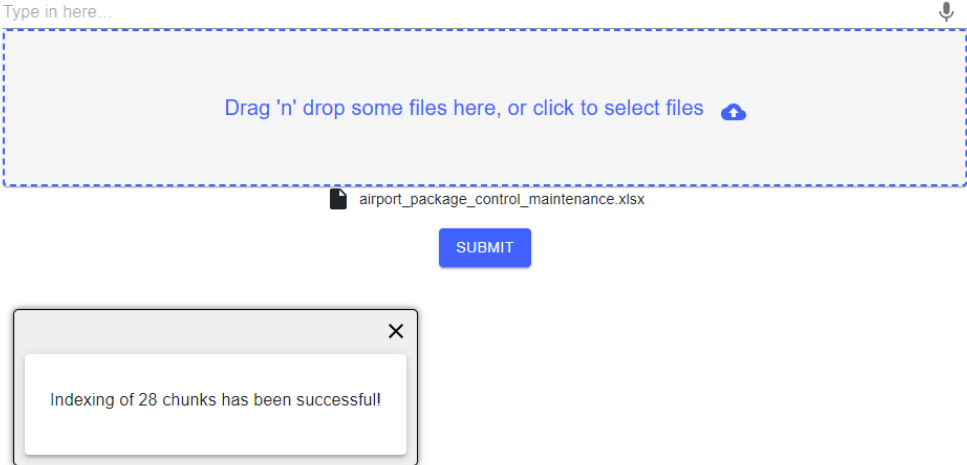
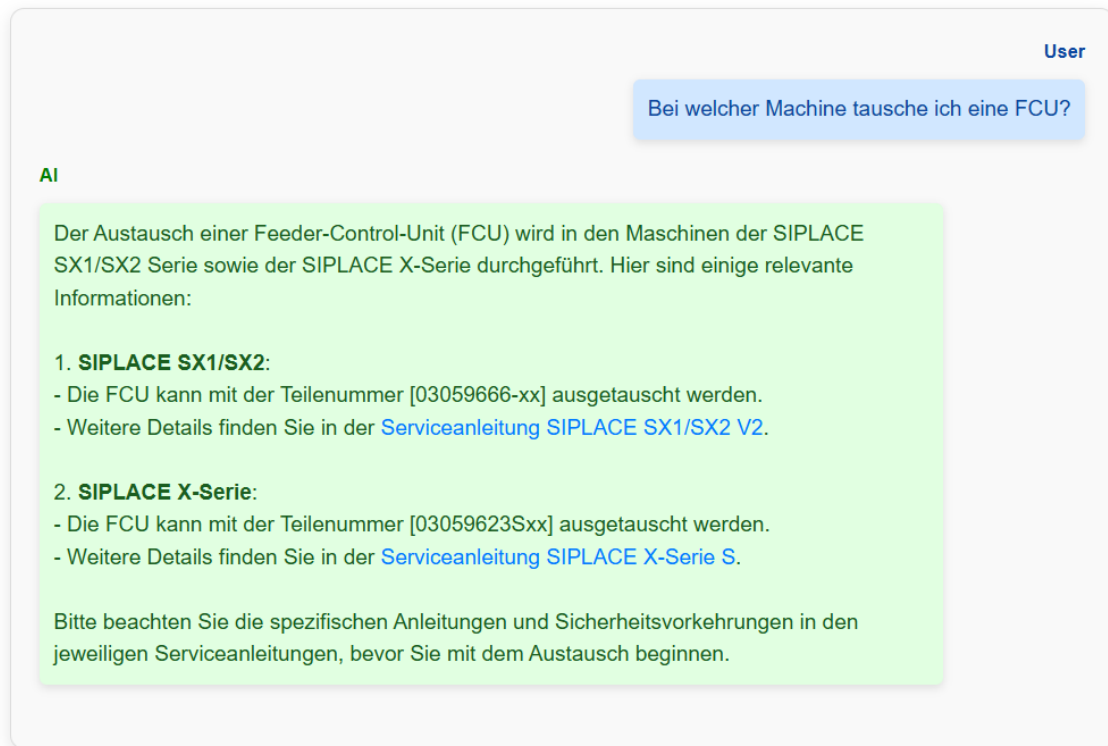


Figure 11: Machine Maintenance Indexing UI

After the indexing is performed for all the relevant documents, the query UI can be used to ask in natural language how to solve a specific issue in a machine. An LLM-based AI will process the request and looking at the information stored in the vector database, will provide the best solution for the issue, referencing the original document as well for further lookup (Figure 12).





Bei welcher Maschine tausche ich eine FCU?



Text-To-Speech

Submit

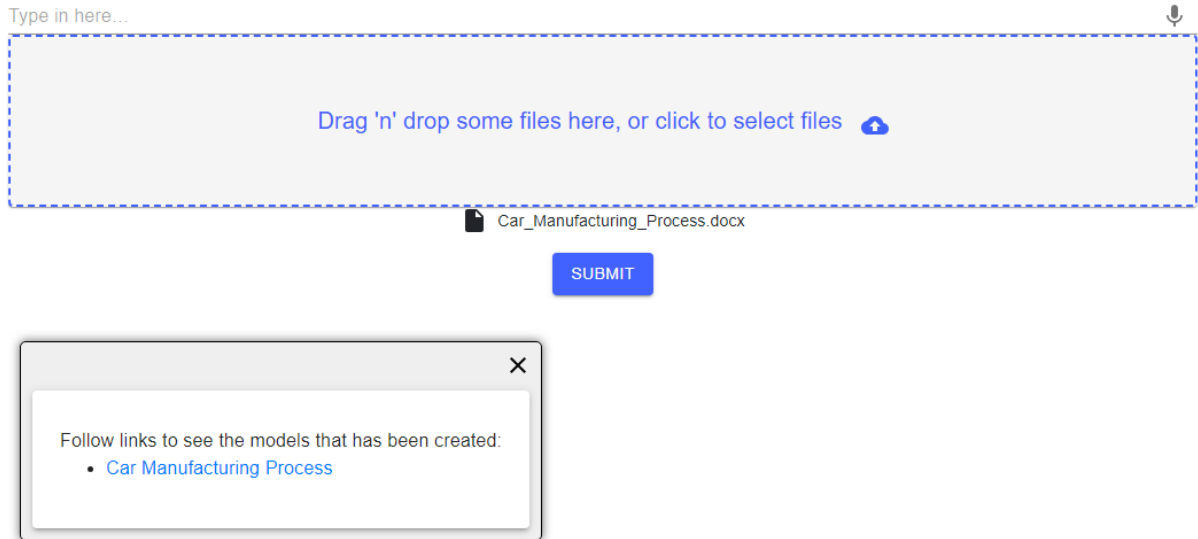
**Figure 12: Machine Maintenance Query UI**

Additionally, the UI provides speech-to-text and text-to-speech functionalities respectively for listening to the query of the user and for reading the results.

This component was not available in the previous report on D4.2.

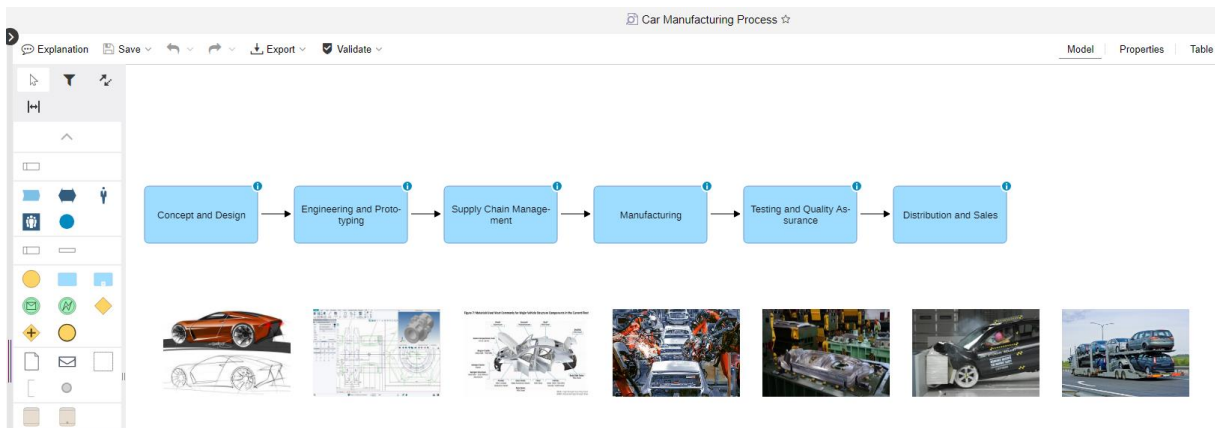
### 3.1.7 Document Transformation UI Component

This component is used for the scenario Improve Reliability of “Documentation about Quality Check” and enables the upload of a document describing a production process to an AI system that processes it finding and extracts the different process steps and the related information like description, details, and images, and generates a semantically enriched process model reflecting the process described in the document (Figure 13).



**Figure 13: Document Transformation UI**

A link to the generated model is visualized to the user as soon as the process is complete. With this link, the user is redirected to a modelling environment where the generated process model is stored.



**Figure 14: Document Transformation Model Result**

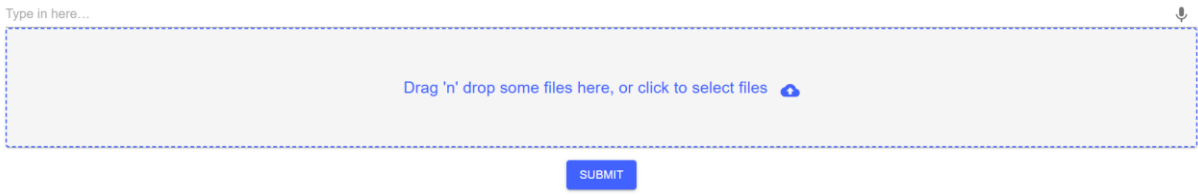
In Figure 14 is visible in a process generated for a Car Manufacturing Process where the different process tasks have been identified and connected in the right sequence order and details like images are visible near each task and in the task properties accessible by double clicking on the blue rectangle representing the task.

This component was not available in the previous report on D4.2.

### 3.1.8 Document Compliance UI Component

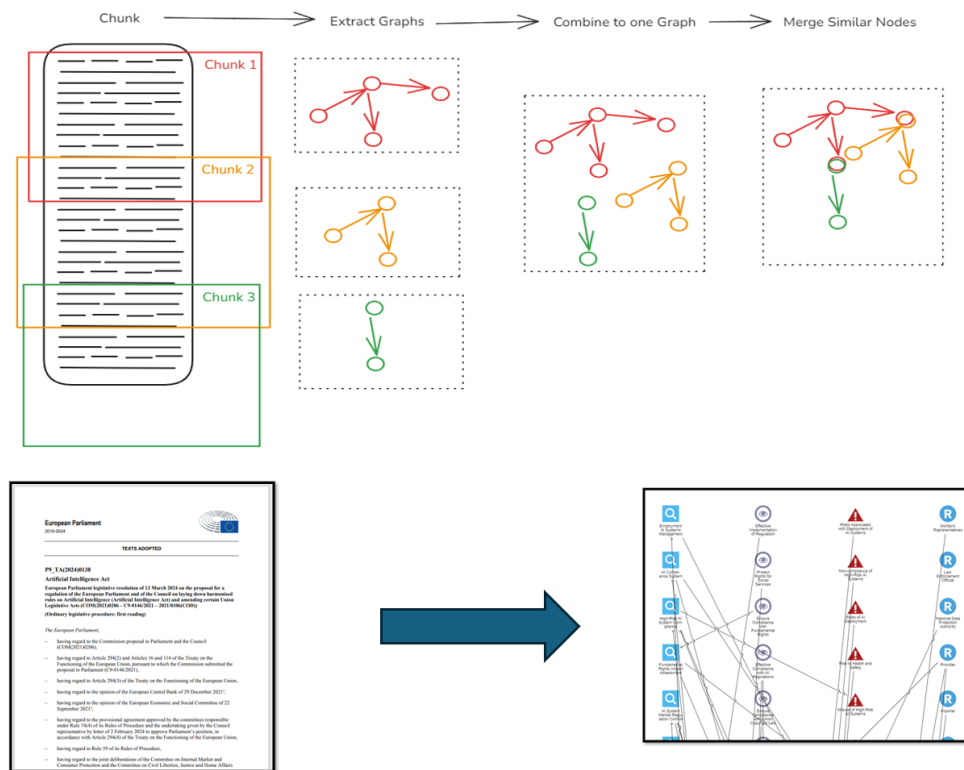
The component is used for the scenario “Improve Information Access to Cleanroom Compliance Requirements” and is composed of two UIs, one for the extraction and indexing of information from different documents and one for querying the semantically enriched differences between two documents.

The UI for indexing a document allows only the upload of a document and triggering an AI that extracts relevant chunks of information and indexes them in a vector database (Figure 15).



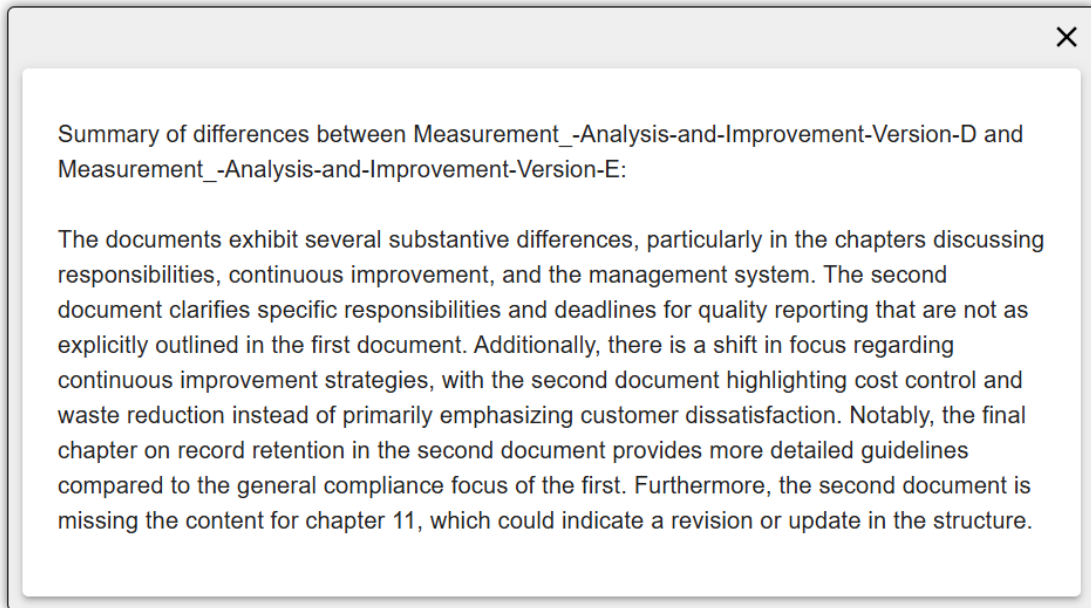
**Figure 15: Document Compliance Indexing UI**

Additionally, the indexing process for this component will generate knowledge graphs from the different chunks and combine them in form of a connected model, that is then used to query the relations between the different parts of the document (Figure 16).



**Figure 16: Document Compliance Information Extraction Process**

After the indexing process is completed for multiple version of the same document, the query UI can be used to call an LLM based AI that retrieve and explain the differences between two version of a document (Figure 17)



Measurement\_-Analysis-and-Improvement-Version-D

---

Measurement\_-Analysis-and-Improvement-Version-E

---

Submit

**Figure 17: Document Compliance Query UI**

This component was not available in the previous report on D4.2.

### 3.1.9 Calibration Document Validation UI Component

The following screenshot (Figure 18) illustrates an example of the user interface that shows the results of the Calibration Document Validation service.

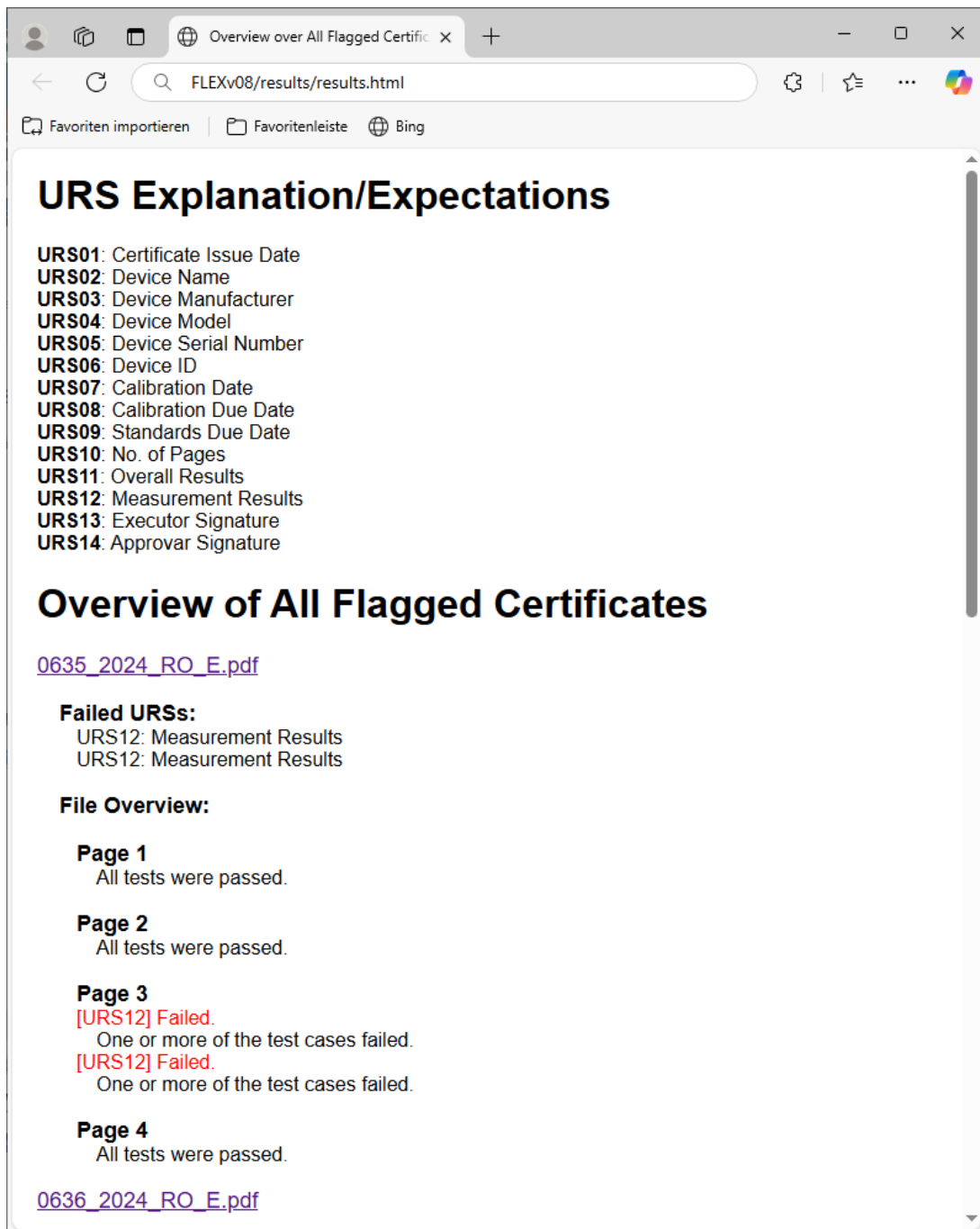


Figure 18: Calibration Validation UI Output

This is done via a web interface with information related to the validation check of URS01-URS14. The top section, entitled "URS Explanation/Expectations", lists the various certificate issues and their corresponding codes, from URS01 (Certificate Issue Date) to URS14 (Approver Signature). This section is key to understanding the flagged issues in the certificates listed below. The bottom section, 'Overview of all flagged certificates', provides details of specific certificates identified as '0635\_2024\_RQ\_E.pdf'. For this certificate, it lists the failed URS codes and provides a file overview showing which pages passed or failed the tests. In particular, the first certificate shows failures related to measurement results (URS12) on pages 3 and 4.

### 3.1.10 Outlook

No additional improvements are foreseen for the different UIs except data alignments in case of changes to the Orchestrator or Knowledge Base backends.

## 3.2 Orchestration of Microservices

This section describes the implementation of the DAI-DSS Orchestrator that manages how the microservices within the project architecture interact and function cohesively. It is designed to control the workflow and data exchange between the microservices and the interconnected building blocks.

### 3.2.1 Workflow-based Orchestration

Workflow-based orchestration was used to orchestrate the different microservices running some of the AI prototypes described in e.g. the UI section 3.1.3. As an engine, the Orchestrator relies on “Netflix Conductor”<sup>1</sup>, and its configuration environment can be accessed via the Configurator of our architecture (see section 3.3.2). Within the Conductor setup environment, users have the option to navigate between sections by utilizing the top navigation bar. In addition to other capabilities, the platform enables the definition and execution of workflows. While the “Definitions” tab provides an overview of the currently specified workflows and allows for editing or adding new ones, the “Workbench” tab enables the execution of these defined processes. The workflow definition area of the workflow engine’s configuration environment is displayed in Figure 19. The names of previously defined workflows are displayed in the window together with other details like the workflow’s version, a brief description, who and when the definition was created, etc.

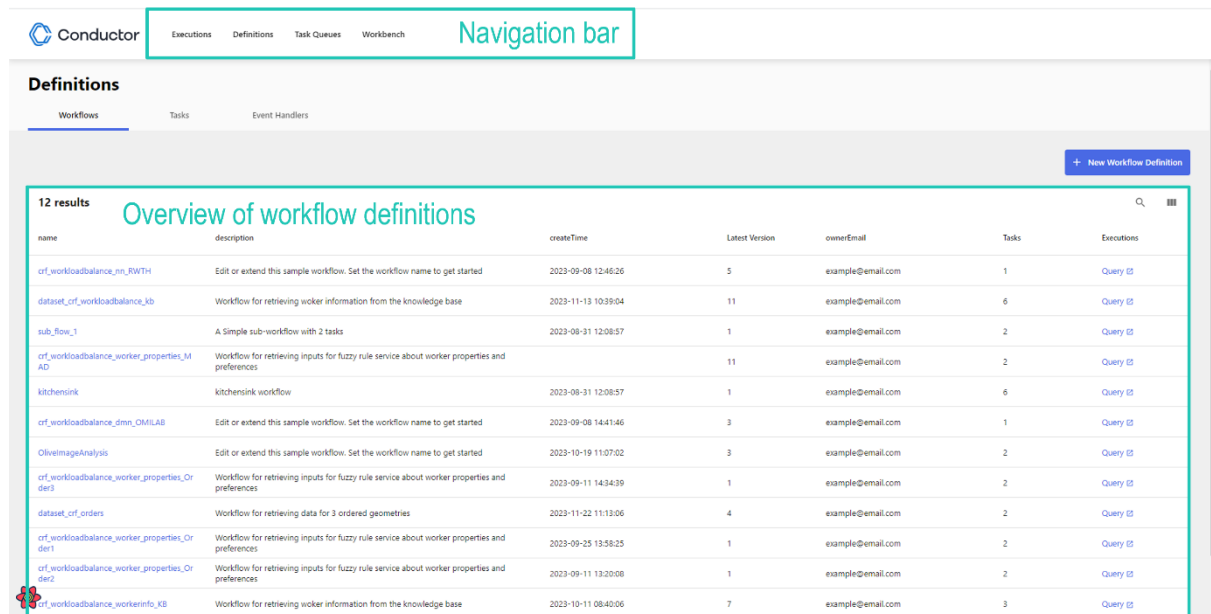


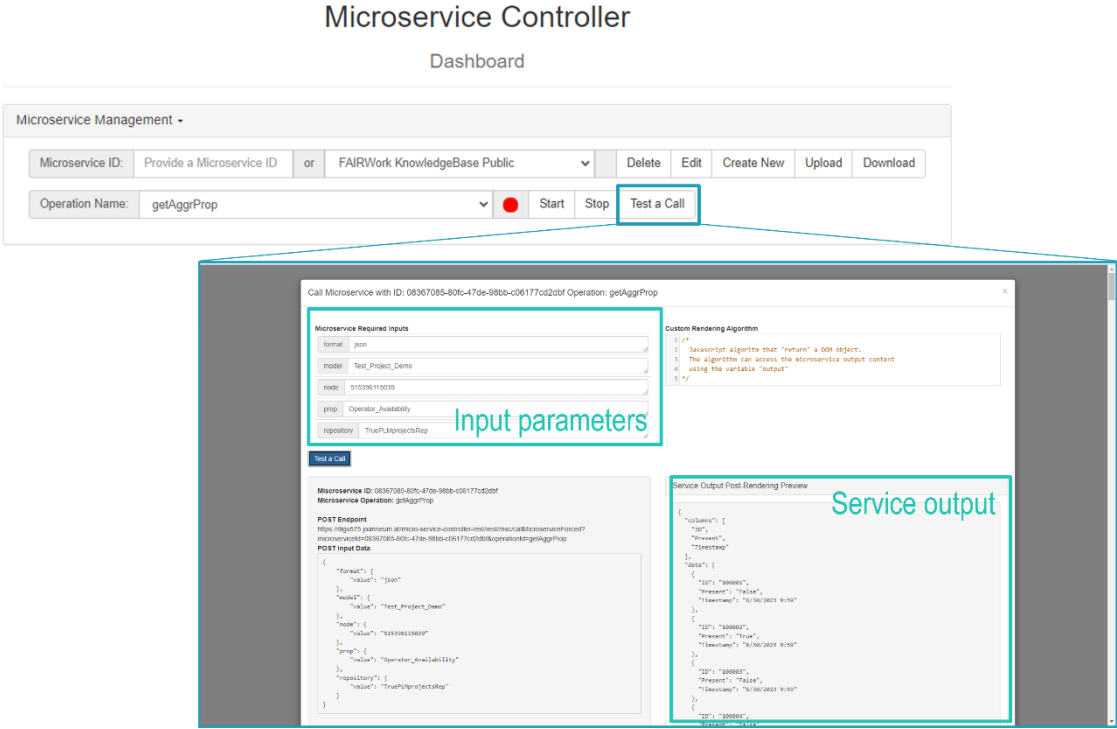
Figure 19: Workflow Engine Control Page

#### 3.2.1.1 Defining Services and Workflows

To tackle the different use case scenarios, retrieving data needed by the AI services or the decision maker from the Knowledge Base is often necessary. Depending on how the needed information is stored, different API calls or combinations of them are necessary to retrieve it. For each API, a service is registered in the microservice controller. The instance of the microservice controller used in the FAIRWork project can be reached via the “Microservice

<sup>1</sup> Conductor. (2023). Basic Concepts - Conductor Documentation. <https://conductor.netflix.com/devguide/concepts/index.html>

Controller Tile” in the Configurator (see section 3.3.3). Registered microservices can be accessed from the UI of the microservice controller. For example, the microservice “FAIRWork Knowledge Base Public” contains several microservice operations for accessing the knowledge base, fetching the required data, and performing necessary output transformations. An example of a test call of one of the created services can be seen in Figure 20. In the microservice controller dashboard, the user can select the name of the microservice and its operation. By clicking on the highlighted button named “Test a Call”, the selected microservice operation can be tried out. In the opening window, the user can specify the required input parameters of the service and send a request by clicking on the button below. The response is then shown on the right side of the window. For more information on the APIs of the knowledge base and how they can be used for retrieving data, see section 3.4 of this document. The so-created microservices have been further used and combined within workflows, which results in different workflow definitions.



**Figure 20: Test Call of a Microservice Operation for retrieving Data from the Knowledge Base in the Microservice Controller**

Workflows can be defined, visualized, and modified in the corresponding section of the workflow configuration platform. The main components of a workflow are tasks and operators<sup>2</sup>. A task is used to perform an activity, usually interacting with external systems or services. An example would be an HTTP task, which can be used to fetch data from an endpoint or make calls to other services. Operators, on the other hand, are utilized to handle branching and decision-making within the workflow. They aid in controlling the flow based on the conditions or outcomes of prior tasks.

An example of a workflow definition overview can be found in Figure 21. In the picture, you can see the workflow definition of “dataset\_crf\_workloadbalance\_kb”, which retrieves the required values of workers and the production line they may be allocated to. In this workflow, you can see multiple HTTP tasks, which call the created services for retrieving properties from the knowledge base. The input parameters in each task specify what information is needed from the Knowledge Base. Some input parameters depend on the output of other tasks. This can be specified by using placeholders referencing the name of the required task output. In the workflow example, the

<sup>2</sup> Conductor. (2023). Tasks — Conductor Documentation. <https://conductor.netflix.com/devguide/concepts/tasks.html>

different tasks for retrieving the necessary information are called in parallel. This is done by using operators to steer the workflow. In this example, you can see a “Fork\_Join” operation, which allows a specified list of tasks to be run in parallel. In this case, we have five HTTP task sequences running in parallel. They are followed by a “Join” operation that waits for the forked tasks to finish and collects their outputs.

The screenshot shows the Conductor interface. On the left, the workflow definition is shown in a code editor. It includes a 'forkJoin' operator that splits the flow into five parallel HTTP tasks: 'http\_task\_availability', 'http\_task\_medical\_condition', 'http\_task\_UTE\_experience', 'http\_task\_workerinfo', and 'http\_task\_required\_worker'. These tasks converge at a 'join' operator before reaching the 'final' state. On the right, a visual representation of this workflow is shown as a flow diagram with nodes and connecting arrows.

Figure 21: Example of a Workflow Definition for retrieving Information from the Knowledge Base

### 3.2.1.2 Testing workflows

While designing a workflow, it is helpful to repetitively test the workflow one is working on. This can be done by accessing the “Workbench” section over the navigation bar. After saving your valid workflow, it can be executed by choosing its name and version in the corresponding fields. If the workflow requires input parameters, they need to be added in the corresponding field in JSON format. By clicking on the play button on the top, the workflow gets executed. The user can click on the workflow ID on the right side to inspect the running workflow in more detail (see Figure 22).

The screenshot shows the 'Workflow Workbench' section of the Conductor interface. A navigation bar at the top includes 'Executions', 'Definitions', 'Task Queues', and 'Workbench'. Below the navigation bar, there is a form for selecting a workflow to execute. The 'Workflow Name' field is set to 'dataset\_of\_workloadbalance\_kb', and the 'Workflow version' is '11'. The 'Input (JSON)' field contains a JSON object with 'nodegeometrym' and 'nodegeometryru' properties. A 'Workflow ID' is shown on the right side of the interface.

Figure 22: Workflow Workbench



When clicking on the workflow ID, an overview of the workflow execution is shown in a new window. By navigating through the tabs, different aspects of the workflow execution can be inspected, like the individual tasks, a summary of the workflow execution, or the input and output of the workflow. In the task diagram, the user can see the execution path, where completed tasks are shown in green (see Figure 23).

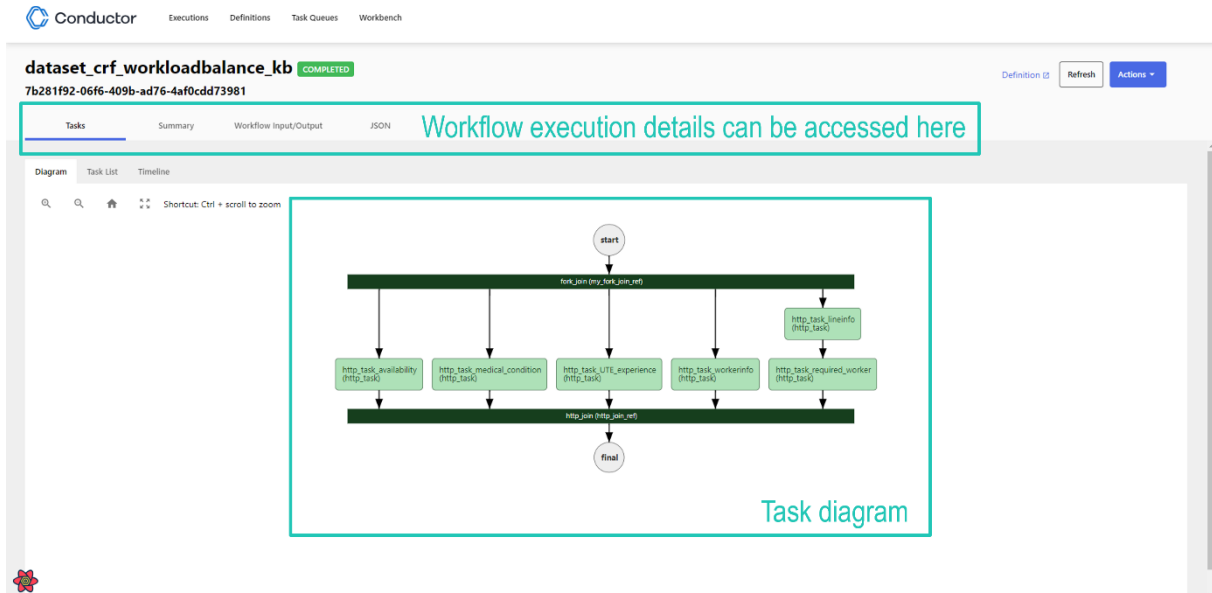


Figure 23: Workflow Execution Details

### 3.2.1.3 Storing and Using Defined Workflows

Workflows can be further used in other components of the system, like the UIs or other services and workflows. This can be done using the APIs provided by the Configuration environment. With the help of these APIs, the user can perform different interactions with the workflow engine, like searching, storing, or triggering workflows. The required action needs to be specified in the body of the API call. In Figure 24, you can see an example of an API call triggering a workflow using Postman. In the picture, you can see that when triggering a workflow, aside from the action, other parameters can be specified in the body of the API call. These properties can include the workflow name, the workflow version, required input parameters, and a filter option. The filter option allows to transform the workflow output using JSONata<sup>3</sup> expressions.

<sup>3</sup> JSONata.org. (2021). JSONata Documentation. <http://docs.jsonata.org/>

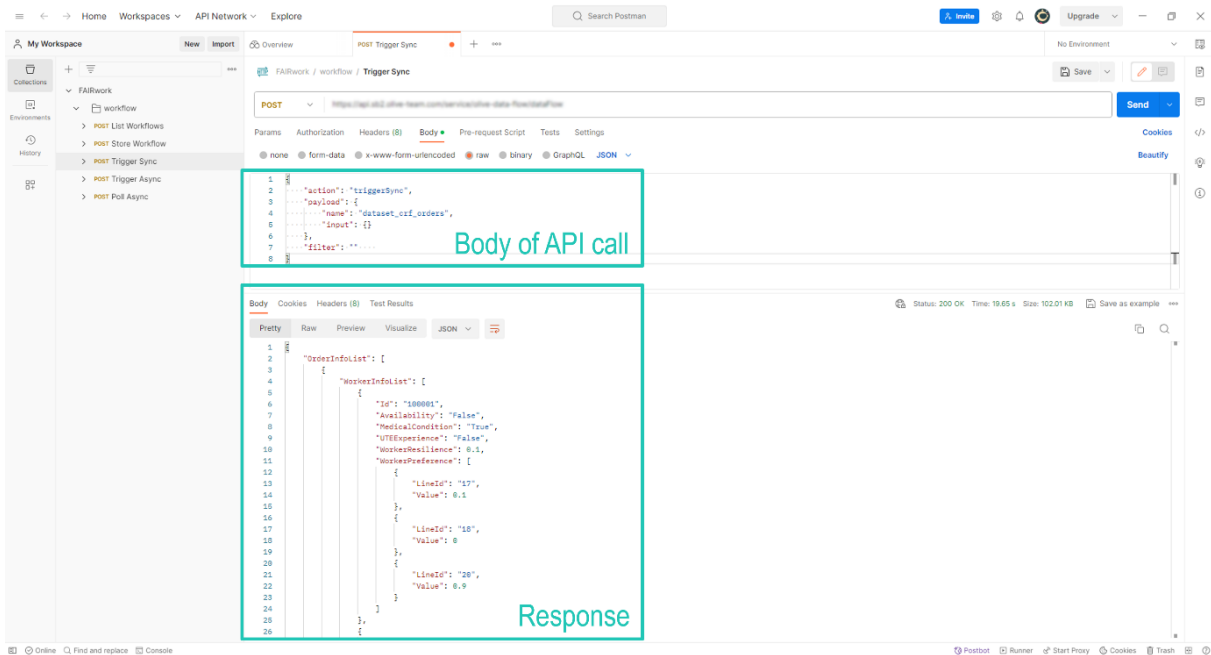


Figure 24: Example of an API Call for triggering a Workflow in Postman

### 3.2.1.4 Simplify the Definition of Linear Workflows

Most of the workflow definitions used for the AI services, e.g presented in the section 3.5.9, do not use complex logic but follow a linear flow. For such services using the Conductor workflow definition format is overwhelming and more error-prone for the development experience. This is the reason that a wrapper around the Conductor engine has been deployed enabling the definition of linear workflow in a more simple and powerful format for both the understandability of the service logic and the expressiveness of task activities (Figure 25).

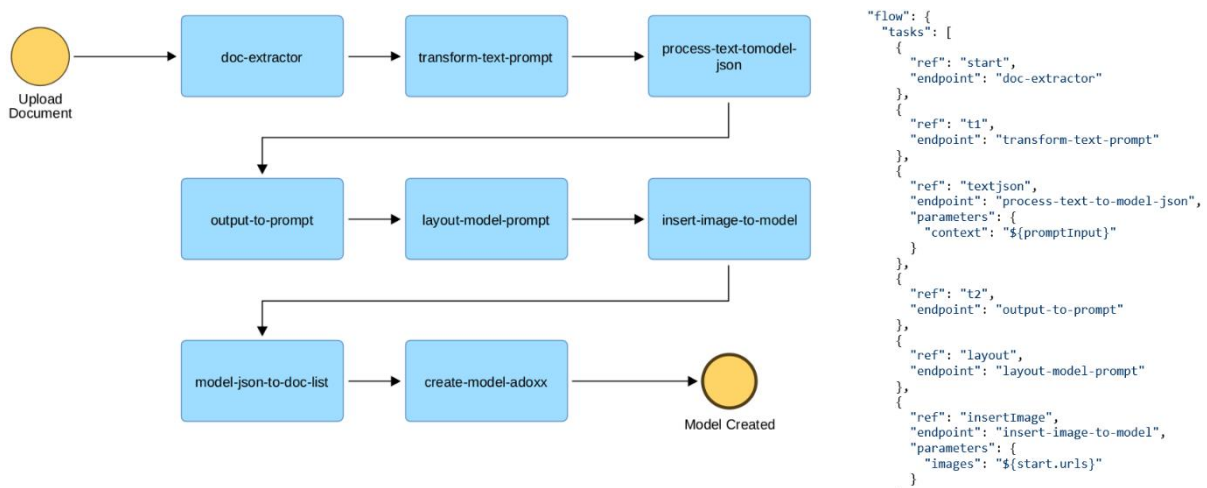


Figure 25: Simplified Workflow Definition

The Configuration environment for such linear workflow can be accessed as well via the configurator described in section 3.3.2.

### 3.2.1.5 Demonstrator Example

Currently, 8 workflows have been created to demonstrate a possible usage in a resource allocation scenario and to support the AI prototypes. The workflow engine can be accessed by clicking on the brown tile called “Workflow Engine” in the Configuration environment (see section 3.3.3). The user will be forwarded to the UI of an instance of the workflow engine “Conductor”. By clicking on the “Definition” section in the navigation bar, a list with currently available workflow definitions is displayed. The workflows relevant to this demonstration have the following characteristics (see Table 2) and can be examined in more detail by clicking on them:

Workflow name	Input parameters
dataset_crf_workloadbalance_kb	<p><b>node_geometry_mc</b> – indicates the ID of the breakpoint element in the knowledge base describing the medical condition for a certain geometry.</p> <p><b>linenumber</b> – indicates the ID of the line where the order will be produced.</p> <p><b>node_geometry_rw</b> – indicates the ID of the breakpoint element in the knowledge base describing the characteristics of a certain geometry.</p>
dataset_crf_orders	none
crf_workloadbalance_dmn_OMILAB	none
ai_document_transofrmation	<b>document</b> – the document to analyse in Base64
ai_machine_maintenance_index	<b>document</b> – the document to index in Base64
ai_machine_maintenance_query	<b>query</b> – the query to perform on the indexed document
ai_document_compliance_index	<b>document</b> – the document to index in Base64
ai_document_compliance_query	<p><b>document_version_1</b> – the name and version of the first document to compare</p> <p><b>document_version_2</b> – the name and version of the second document to compare</p>

**Table 2: Orchestration Workflows**

The input parameters need to be provided in JSON format. An example of input parameters that can be used in the workflow “dataset\_crf\_workloadbalance\_kb” would be the following:

```
{
  "node_geometry_mc": "515396245145",
  "linenumber": "18",
  "node_geometry_rw": "515396162776"
}
```

Considering the first workflow mentioned in the table as a sample, it retrieves necessary values from the Knowledge Base needed by the decision-maker by calling different APIs. In this case, five of the HTTP tasks are called in

parallel, as only one task is dependent on its predecessor. The data retrieved is needed to support a resource allocation decision challenge. It includes information about the workers, like their availability, if they fulfil the required medical condition to work with a certain geometry, their experience with a certain production unit, their preferences, etc., as well as information regarding the line on which a certain geometry is produced. In general, the output of this workflow contains all the required information to describe one order of a certain geometry. The completed task diagram can be seen in Figure 26.

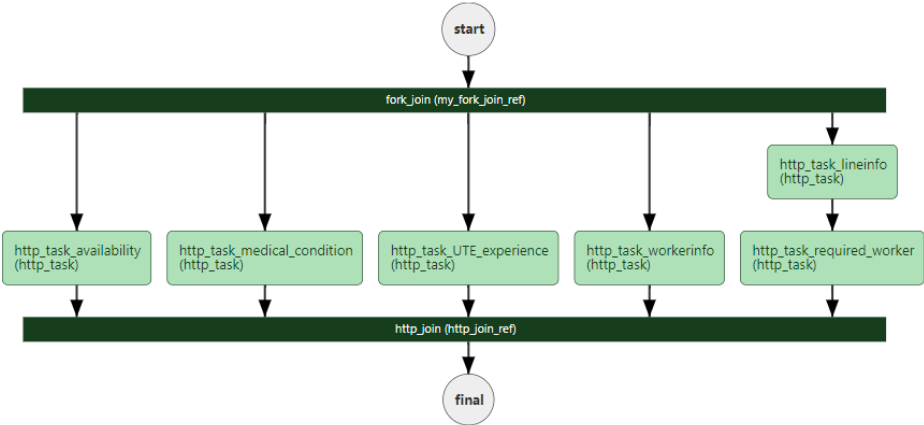


Figure 26: Execution Path of Workflow "dataset\_crf\_workloadbalance\_kb".

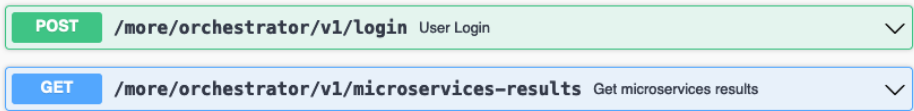
### 3.2.2 Multi-Agent Orchestration

When combined with microservices architectures, Multi-Agent Systems play an important role in creating decentralized and adaptable systems. This combination provides independent scalability for different system components, allowing tasks to be distributed among specific agents. The decomposition into microservices makes it easier to update and modify parts of the system without affecting the rest, enabling the dynamic introduction of new agents or services. Each microservice contributes to the system's resilience, ensuring that failures in one agent or microservice do not bring down the entire system. This also allows for the distribution of responsibilities and efficient adaptation to new requirements or scenarios. However, this integration may pose challenges, such as the complexity of managing various microservices, requiring a strategy for communication and coordination among them.

The main reasons for implementing multi-agents with a microservice architecture are explicitly presented below:

- Scalability: microservices enable independent scalability of diverse system components. By integrating agents into microservices, it is possible to achieve a decentralized control of services that can support decision-making.
- Flexibility and decoupling: decomposition into microservices makes it easier to update, modify, or replace parts of the system without affecting the whole structure. When combined with an agent-based approach, it enables dynamic system adaptation as new agents are introduced, or existing ones are modified.
- Resilience: if an agent or microservice fails, it does not necessarily bring down the entire system due to the decentralized nature of microservices.

The orchestrator developed provided access to the results of the microservices to Assist Decisions about Production Planning. This is a secured element developed in Python through the Flask framework and provides two endpoints to its users, as shown in Figure 27. This element also has administrative endpoints that are only accessible to the administrator and uses JWT – JSON Web Tokens – to protect data access. The login endpoint is used to acquire the necessary JWT token to read the data through the microservices-results endpoint.



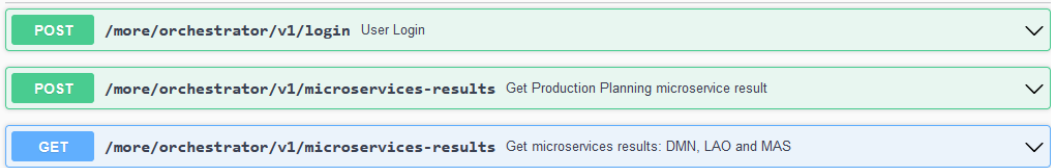
**Figure 27: Available Endpoints to Registered Users**

The Waitress Web Server Gateway Interface (WSGI) is used in production, serving the orchestrator web app in port 5051. The web app connects to the outside world through the machine’s Apache proxy server, using port 443 to do so. This server is used in both ways of communication, adding an extra layer of security to the system.

Deployment-wise, docker services were used to encapsulate the necessary elements to have the orchestrator running securely, allowing the integration of this element with the microservices developed within the FAIRWork platform. Thus, a docker image for the orchestrator was created, and hosted in Docker Hub and a docker service running a database was also deployed to manage the orchestrator user credentials information. The containerization enables the web app effortless deployment, ensuring the smooth, uninterrupted execution of the orchestrator on the server, even amid concurrent services also running on the same machine. The orchestrator runs on the server using SSL certification through the Apache server to encrypt/protect the communications.

This integration between the MAS Orchestrator and microservices enhances decision-making capabilities. They bring autonomous decision-making to individual components, enabling efficient resource management. When combined with the loosely coupled nature of microservices, these agents can interact and collaborate. Furthermore, the architecture ensures functional integration with the Knowledge Base and UIs for accessing and storing data useful to the microservices and displaying the output and receiving input from the decision-maker.

In FAIRWork, taking advantage of the scalability provided by the approach used to develop the orchestrator, the endpoint that provides the UIs with the microservices results was updated to also work with the POST verb, as shown in Figure 28. This allows the reception, by the orchestrator, of an excel file containing an order list used to create the input of the production planning microservice.



**Figure 28: Orchestrator Endpoints**

The login endpoint has to be used first to acquire the JWT (JSON Web Token) that will allow the usage of the microservices-result endpoint, regardless of the used REST verb on this endpoint.

**3.2.2.1 Login Endpoint**

The login endpoint requires the username and password to be sent in the body of the request. This process is depicted in Figure 29.

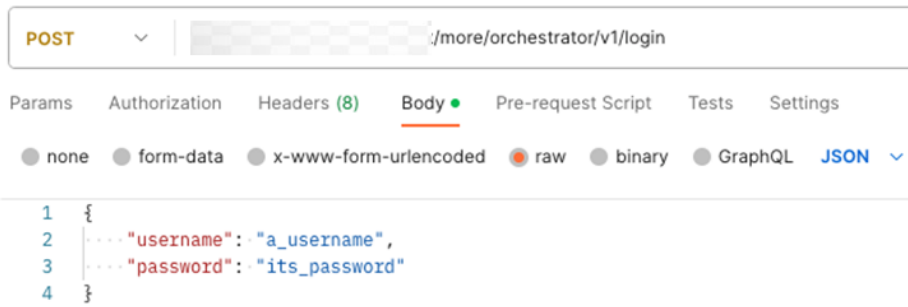


Figure 29: Login Endpoint Usage

The endpoint returns a 401 HTTP response status code if an unsuccessful login attempt is executed. In case of success, the server returns a 200 HTTP status code, and the JWT access token and its validity in the response's body, as presented in Figure 30. Token validity is set for 30 minutes.

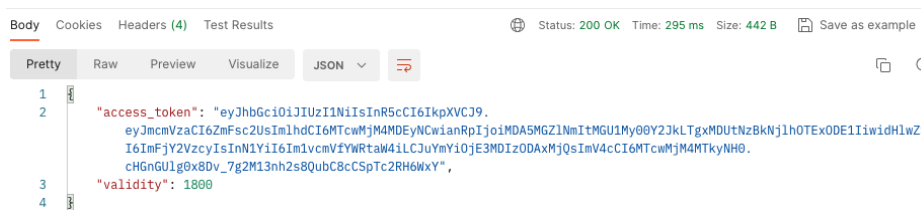


Figure 30: Successful Login Response

To maintain data integrity, input and output models were defined for this endpoint to force the data to conform to these models. The models are presented in Figure 31.

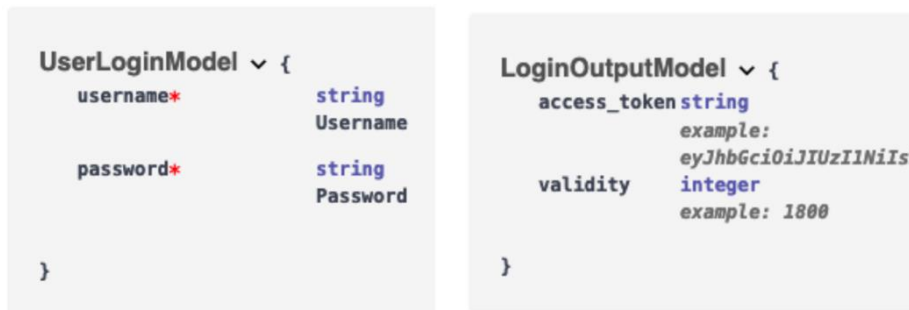


Figure 31: Login Endpoint Data Models

### 3.2.2.2 Microservices Results Endpoint

This endpoint is secured and requires a valid JWT access token, retrieved from the login endpoint, to be passed in the Authorization header as a Bearer token. If a malformed/invalid/expired token is used, the endpoint response uses a 401 HTTP response status code and a specific message in the response's body to signal this to the user making the request. If a valid token is used, the endpoint returns the microservices allocation results in a JSON according to the agreed-upon format to convey this information. Figure 32 shows the configuration of the REST request to use this endpoint properly.

Key	Value	Description
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJm...	

**Figure 32: Microservices Allocation Results Endpoint Usage**

When using the GET REST verb, the endpoint accepts parameters to define which microservices should be inquired to get their results. This is presented generically in Figure 33.

Key	Value
<input checked="" type="checkbox"/> microservice1	true
<input checked="" type="checkbox"/> microservice2	false
<input checked="" type="checkbox"/> microservice3	true

**Figure 33: Generic Parameters Structure to be used in the GET Request**

In this case, three models were devised:

- WorkerModel: to specify worker properties;
- AllocationModel: to specify worker allocation to a specific line;
- OutputModel: to specify worker allocation to all lines.

The Output Model includes the other two. Thus, it is presented in Figure 34 to depict all models and their integration as described above.

```

OutputModel {
  AllocationList
  [AllocationModel {
    LineId string example: 17
    WorkersRequired integer example: 6
    Workers [WorkerModel {
      Id string example: 100102
      Availability string example: True
      MedicalCondition string example: True
      UTEExperience string example: False
      WorkerResilience number example: 0.4
      WorkerPreference number example: 0.7
    }]
  }]
}

```

**Figure 34: Output Model**

For this endpoint, when using the POST REST request, to query the production planning microservice, two models were used, namely:

- Allocation: to specify workers to a given task;
- Planning: to provide multiple task allocations.

Figure 35, depicts these models, since Planning uses the allocation model to structure the data.

```

Planning {
  experience {
    number
    example: 0.85
    Experience value
  }
  preference {
    number
    example: 0.9
    Preference value
  }
  resilience {
    number
    example: 0.7
    Resilience value
  }
  transparency {
    string
    example: medium
    Transparency level
  }
  allocations {
    [
      example: List [ OrderedMap { "Task": "SEV-38", "Start": 1699096800, "Finish": 1699104000, "Resource": "Line 24",
      "geometry": "1342266080", "required_workers": 2, "workers": List [ 100002, 100087 ] }, OrderedMap { "Task":
      "SEV-36", "Start": 1699096800, "Finish": 1699111200, "Resource": "Line 20", "geometry": "533908540",
      "required_workers": 3, "workers": List [ 100039, 100143, 100121 ] } ]
      List of allocations
      Allocation {
        Task {
          string
          Task ID
        }
        Start {
          integer
          Start timestamp
        }
        Finish {
          integer
          Finish timestamp
        }
        Resource {
          string
          Resource name
        }
        geometry {
          string
          Geometry ID
        }
        required_workers {
          integer
          Number of required workers
        }
        workers {
          [
            List of worker IDs allocated to the Task
            string]
          ]
        }
      }
    ]
  }
}

```

**Figure 35: Production Planning Microservice Data Model**

This approach allows the orchestrator to easily scale by accommodating new microservices while ensuring their availability. When a new microservice is created, it only needs to grant access to its results. The orchestrator can configure the connectivity to this microservice using a new parameter in the request sent to the data acquisition endpoint created to identify that microservice. Consequently, this fosters system resilience since the system still functions even in the event of microservice failure. Figure 36 illustrates how the Multi-Agent Orchestrator interacts with the microservices of the CRF Workload Balance use case scenario.



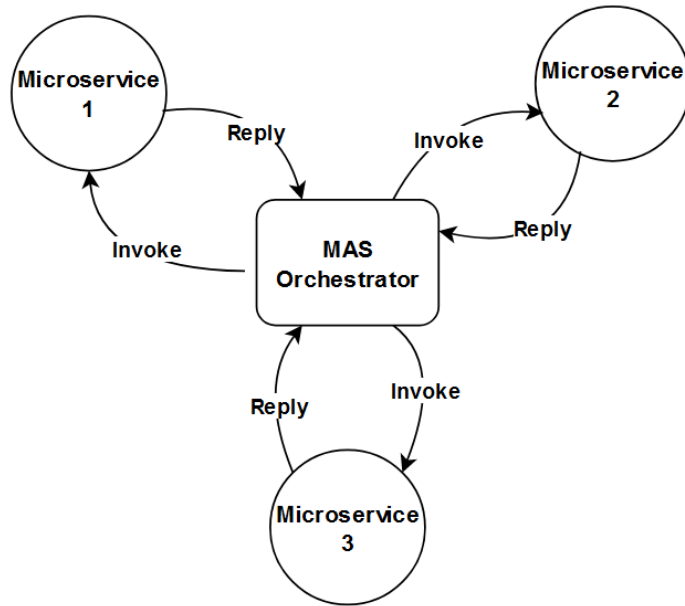


Figure 36: MAS Orchestration of Microservices

### 3.2.3 Outlook

The Orchestrator provides the link between the UI and the AI microservices. To do this, it collects data from the Knowledge Base and user input and processes it in order to meet the requirements of the microservices, interacting with this building block in order to offer the result to the user via the UI. The Orchestrator connects the components of the architecture developed in a cohesive and objective way, allowing for added scalability so that the system can grow as new services are added.

## 3.3 Integrating Configuration

### 3.3.1 Multi-Agent Orchestrator Configuration

#### *Microservices results endpoint*

In the Multi-Agent Orchestrator, an example of configuration is given below:

After login, the received JWT token is used in an Authorization header to gain access to the endpoint. This is shown in Figure 37.

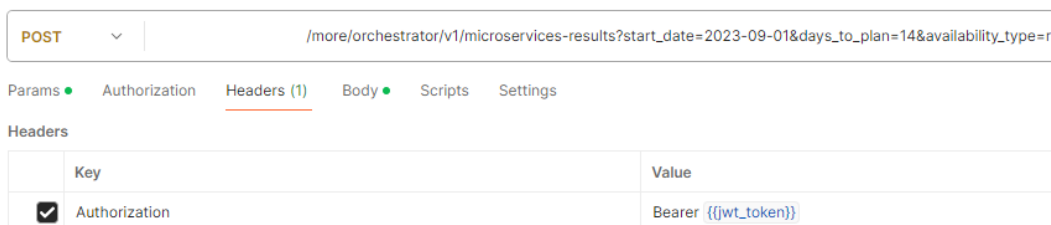


Figure 37: Required Authorization Header

The POST REST request to this endpoint uses parameters to configure the usage of the production planning microservice. Figure 38 shows an example of these parameters.

#### Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	start_date	2023-09-01
<input checked="" type="checkbox"/>	days_to_plan	14
<input checked="" type="checkbox"/>	availability_type	r

**Figure 38: POST Request Parameters**

The parameter *start\_date* is mandatory, has to be in accordance with the dates present on the Excel file, and represents the date to start the planning.

The parameter *days\_to\_plan* is optional and represents the time interval to be used in the planning algorithm. If not used, it defaults to 14 days.

The parameter *availability\_type* is optional and represents the worker's availability to be used in the planning algorithm given the data provided by the end user:

- Programmed availability: p
- Real availability: r

If not used, it defaults to (p)rogrammed availability.

After input preparation, contacting the microservice endpoint, and getting its response, it replies to the UI with the data from the microservice using the planning data model presented in Figure 35.

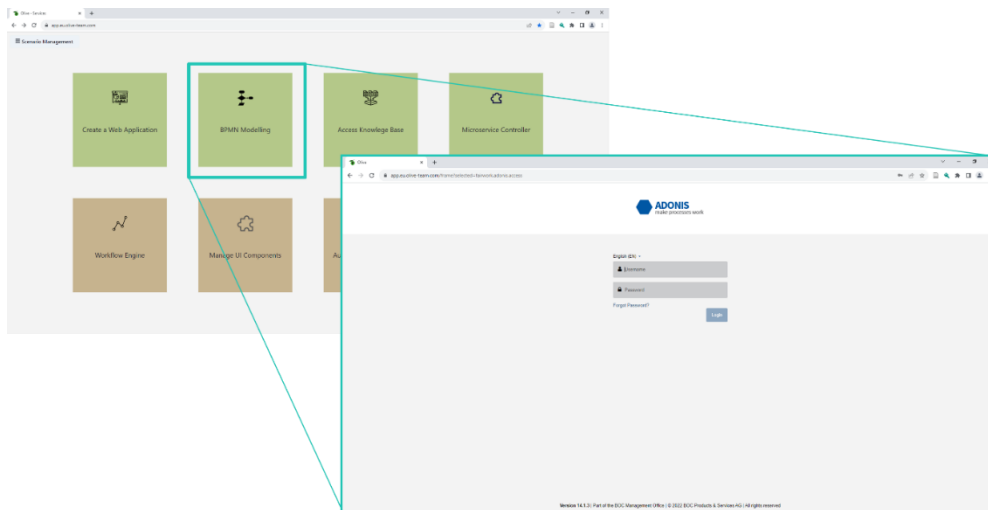
## 3.3.2 Configuration Framework

### 3.3.2.1 Decision Models with BPMN

The cloud-based modelling platform ADONIS<sup>4</sup> is used as the FAIRWork process modelling environment. Here decision models to concretize the use case requirements and ease the identification of relevant aspects have been modelled to define the challenges and parameters crucial to the decision-maker. ADONIS is a framework based on the ADOxx meta-modelling platform<sup>5</sup>, that is BPMN 2.0 compliant. The modelling tool can be accessed in the configuration environment by clicking on the tile with the name “BPMN Modelling”. Figure 39 depicts the configuration environment and the ADONIS login page (credentials can be requested at [fag@adoxx.org](mailto:fag@adoxx.org)). How you can navigate in the modelling tool is described in detail in FAIRWork D5.1. Additionally, the decision models can be found in the knowledge base under FAIRWork\_Project / CRF / Decision models and FAIRWork\_Project / Flex / Decision models.

<sup>4</sup> BOC Group <https://www.boc-group.com/en/adonis/> (last visited: 18-02-2025)

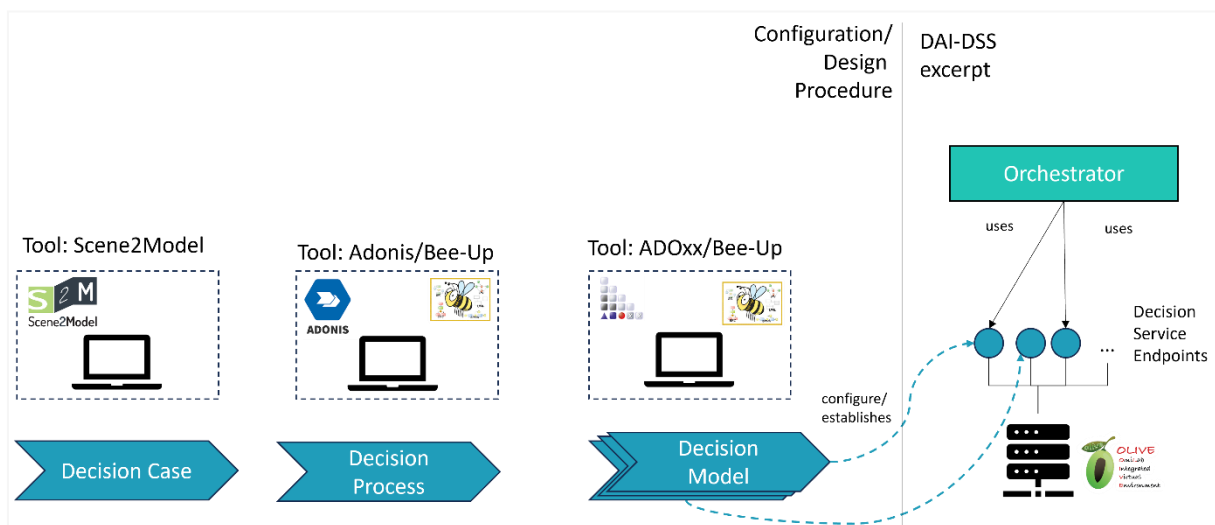
<sup>5</sup> OMILAB NPO. <https://www.adoxx.org/> (last visited: 18-02-2025)



**Figure 39: Accessing Login Page of Modelling Tool.**

### 3.3.2.2 Configuration of Decision Services Experiment

Within the DAI-DSS, not only should the combination of available services be configurable to achieve orchestration, but also, the decision services themselves should be adaptable to different situations. How well a decision service can be adapted to new situations depends on the used decision algorithm and on its design. For this DAI-DSS prototype, an experiment was created to configure a rule-based decision service using conceptual modelling. The service and how it can be configured to create tailored decision support is described in section 3.5.1. In this section, we want to discuss the procedure and environment that were used to create this experiment for the configuration.



**Figure 40: Overview Experiment for Decision Service Configuration**

The procedure for the configuration within this experiment is based on the one that was used to identify the use cases and decision challenges for the FAIRWork project (see FAIRWork D2.1 for details). The procedure for this experiment and its connection to the DAI-DSS are visualized in Figure 40. The procedure starts with understanding and defining the decision case which should be supported. This is an important step, as one must know exactly what should be decided and what the context is if automated decision support should be established. Afterward, the different steps and the needed information for making the decision must be defined to have a more sophisticated understanding of what must be done to make a final decision. This can include preparing steps, or maybe the decision itself is done through a series of sub-decisions, which must be done in a certain order. This knowledge is

presented in the decision process and is derived from the decision cases. Finally, the decision and its possible sub-decision must be specified in a way that a decision support system can suggest a solution. This includes a way of defining the decision logic and also to create an instance of a decision service, which can then be called by the overall decision support system.

To understand and define the decision knowledge in this experiment, we used conceptual modelling with corresponding modelling tools. We used modelling tools based on a common metamodelling platform; in our case, this was the ADOxx meta-modelling platform. This should later help add new and adapt existing experiments with the modelling methods used.

For defining understanding and defining the definition case, the Scene2Model<sup>6</sup> tool was used, which supports physical workshops to foster information exchange and understanding between the participants in the workshops. Additionally, it supports the capturing of the created knowledge through digital, conceptual modelling, which is automatically generated from the physical artifacts created in the workshop. The generated models can then be further enriched and are available in a machine-processable way.

For modelling the decision processes and the decision models for this first experiment, we used a standardized modelling method to start with tested and well-known modelling methods. In particular, *Business Process Model and Notation (BPMN)* and *Decision Model and Notation (DMN)*. Both of these are available in the Bee-Up<sup>7</sup> modelling tool (which is also based on ADOxx), and therefore, this tool was used for this experiment.

To use the decision models created with DMN as input for configuring a decision service, the Bee-Up tool was extended to fit the needs of the experiment. After these extensions, the models can be used during the configuration of a decision service. How the models can be used as configuration input in this experiment is explained in the service that will be introduced in the section 3.5.1.

Finally, to ease the creation of a callable decision service, a framework is needed that supports the users in offering callable endpoints and running the needed algorithms. For this part, the microservice controller framework<sup>8</sup> was used. For this, a connector for consuming the modelled information and instantiating a decision service was created, which can be used together with the models created for this experiment. If the decision endpoints are tested and found feasible, they can be used by the orchestrator to support the decision-making in concrete use cases.

### 3.3.3 Configuration Integration Framework

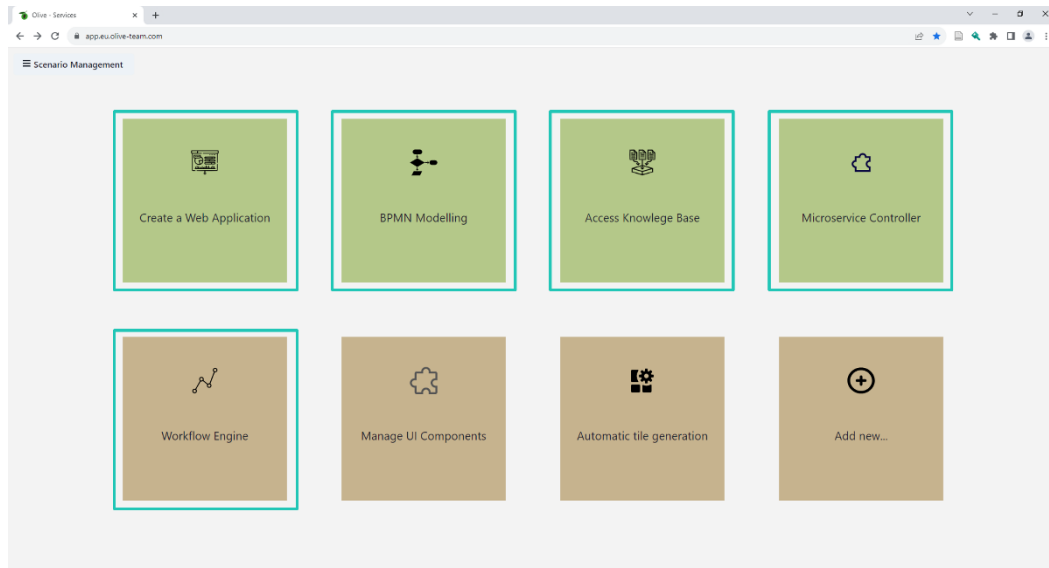
The entry page of the Configuration environment can be seen in Figure 41. Each tile on the web page gives access to configuration wizards or platforms for defining necessary parts of the system, e.g. UIs or workflows. The tile “Create a Web Application” enables the configuration of a UI for a web application, while the tiles “BPMN Modelling”, “Access Knowledge Base”, “Microservice Controller” and “Workflow Engine” provide access respectively to the ADONIS based BPMN modelling environment described in section 3.3.2.1, the Knowledge Base described in the section 3.4, the microservice controller, and the workflow engine described in the section 3.2.1.

---

<sup>6</sup> OMiLAB NPO. <https://www.omilab.org/activities/scene2model/> (last visited: 18-02-2025)

<sup>7</sup> OMiLAB NPO. <https://bee-up.omilab.org/activities/bee-up/> (last visited: 18-02-2025)

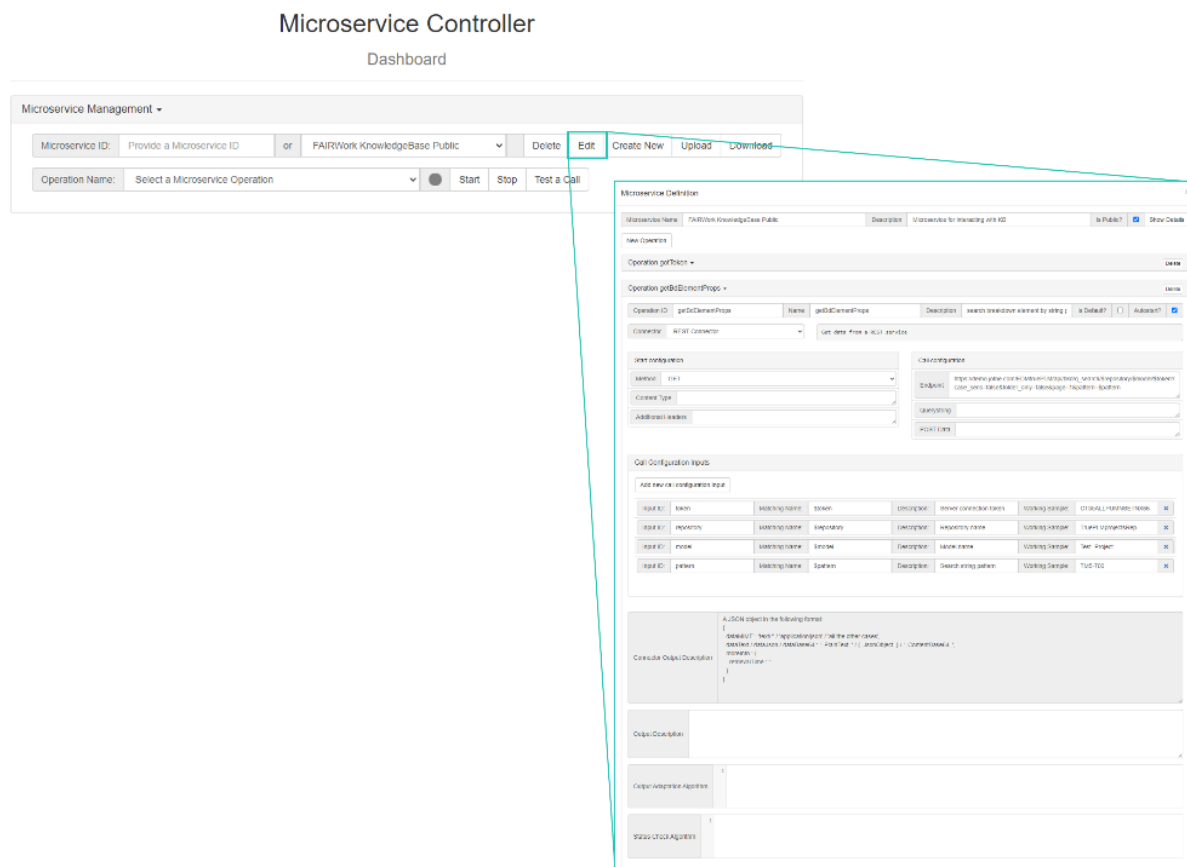
<sup>8</sup> Adoxx.org. <https://www.adoxx.org/live/olive> (last visited: 18-02-2025)



**Figure 41: OLIVE instance showing the Configuration Environment.**

### **3.3.3.1 Configuration of Services**

The microservice controller, accessible from the relative tile in the Configuration environment entry page, allows to register and manage microservices to follow a configuration approach. Microservice operations can be defined through the configuration of so-called connectors. There are different types of connectors available. Currently, we use mostly the REST Connector for getting data from a REST service, like the APIs available for accessing the knowledge base and the “Javascript Nashorn Engine Connector”, which executes a JavaScript using the Nashorn Engine and returns its output. In the microservice controller dashboard, the user can choose a predefined microservice and click on “Edit”. Now, all currently registered microservice operations and their definitions can be inspected. Depending on the selected connector, different configuration options are available (see Figure 42). An example of how the microservice controller was used in this project so far can also be seen in the experimental decision support service described in section 3.5.1.



**Figure 42: Example of Microservice Definition.**

### 3.3.3.2 Configuration of Workflows

For now, the Configuration environment allows the management of workflow by providing access to the workflow engine either by manually accessing it by clicking on the tile named “Workflow Engine” or by using the provided APIs. How such a workflow definition can look is described in section 3.2.1.

### 3.3.3.3 Configuration of User Interfaces

Besides handling microservices and workflows, this environment enables the configuration of UIs by combining different UI components, and after their configuration, they can be deployed as web applications. By clicking on the Tile “Create a Web Application,” the user is forwarded to a wizard, which assists in configuring the needed web application. In the first step, the user can choose a layout template. Currently, only the “Table Layout” can be selected, but additional templates can be added. After the selection, the “Next”-Button on the right can be clicked to proceed with the second step of the configuration wizard. In the second step of the wizard, the UI can be designed by dragging and dropping the needed UI components into the fields of the given layout template. The UI components available are all the ones described in the section 3.1. Configuration of specific properties for each UI component is possible in the right panel of the UI builder after selecting each component. After the user finalizes the placement of the components, he/she can proceed with the next step by clicking on the next button on the right side again. In the third step, a preview of the web application is displayed. By clicking again on the next button, the web application is automatically deployed and can be accessed via the displayed link. Figure 43 shows the different configuration steps.

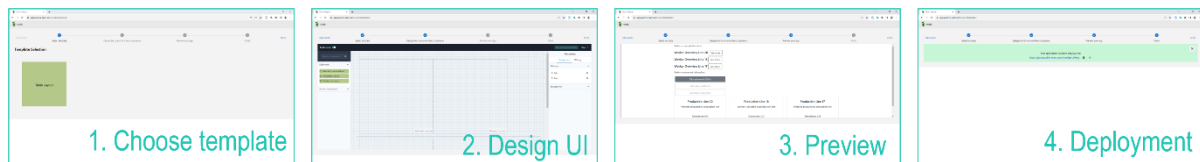


Figure 43: Configuration Steps of creating a Web Application.

### 3.3.4 Outlook

At the configuration level, one can select microservices and configure parameters to define important input data for the use case. This makes it possible to adjust how each microservice will be triggered according to the project's needs. In addition, this approach offers greater flexibility and scalability, as it makes it possible to evolve and maintain each component independently without affecting the others.

## 3.4 Integrating the Knowledge Base

The integration between the Knowledge Base and the Orchestrator is facilitated through a REST API, enabling seamless data exchange and decision-making within the DAI-DSS framework. Figure 44 and Figure 45 illustrate the UML sequence diagrams that detail the interactions between various components for achieving production planning and worker allocation tasks in the CRF workload balance and production planning use case scenarios.

Below is an overview of the process:

1. **User Interaction with the DAI-DSS framework:** The process begins when the user interacts with the DAI-DSS User Interface. The UI prompts the user to upload a CSV file containing production order details. This file typically includes "Production orders", "Geometry codes" and "Due dates".
2. **CSV File Processing by the Orchestrator:** After the user uploads the CSV file, the Orchestrator processes its content by, parsing the file to extract geometry codes. Using these geometry codes as query parameters to fetch relevant data from the Knowledge Base.
3. **Knowledge Base Queries and Information Retrieval:** The Orchestrator queries the Knowledge Base via REST API, retrieving the following essential information for each geometry code like Production throughput specific to each geometry code (Throughput Values CNO). Similarly data on production lines associated with the geometry code, including "Line number ", "Backup line", "Mold" and Number of operators required to produce a geometry on a given line. If any of the above information is missing, the geometry code is excluded from further processing.
4. **Enrichment with Additional Data:** For geometry codes with complete information, the Orchestrator retrieves further data from the Knowledge Base:
  - Suitability Information: The Orchestrator validates data from the stevedore suitability dataset retrieved from the Knowledge Base to ensure all required geometry codes match the production planning sheet. Geometry codes not found in the response are excluded.
  - Operator Preferences: Preferences of individual operators for handling specific geometry codes.
  - Resilience Information: Each operator's resilience in relation to a geometry code.
  - Geometry codes or stevedore IDs lacking any of the above data are excluded from further processing.

5. **Data Aggregation for AI Services:** All the retrieved data including suitability, throughput, line information, operator preferences, and resilience data is aggregated into a structured JSON format. This JSON file serves as the input for AI Services, enabling decision-making.
6. **AI Services and Decision Support:** The AI Services analyze the aggregated JSON and generate recommendations for Worker Allocation by providing optimal assignment of workers based on skills, preferences, and production requirements. Similarly, for the production planning scenario, the services provide an efficient scheduling and workload distribution plan across the available production lines.
7. **Result Presentation via the UI:** The AI Services send their recommendations back to the Orchestrator.
  - The Orchestrator formats the results and displays them on the DAI-DSS UI.
  - Users are presented with clear, actionable insights, enabling informed decision-making for production planning and workload balancing.

This structured process ensures that the DAI-DSS framework effectively integrates data, AI analytics, and user interaction to optimize CRF workload balance.



Production Order Request Processing

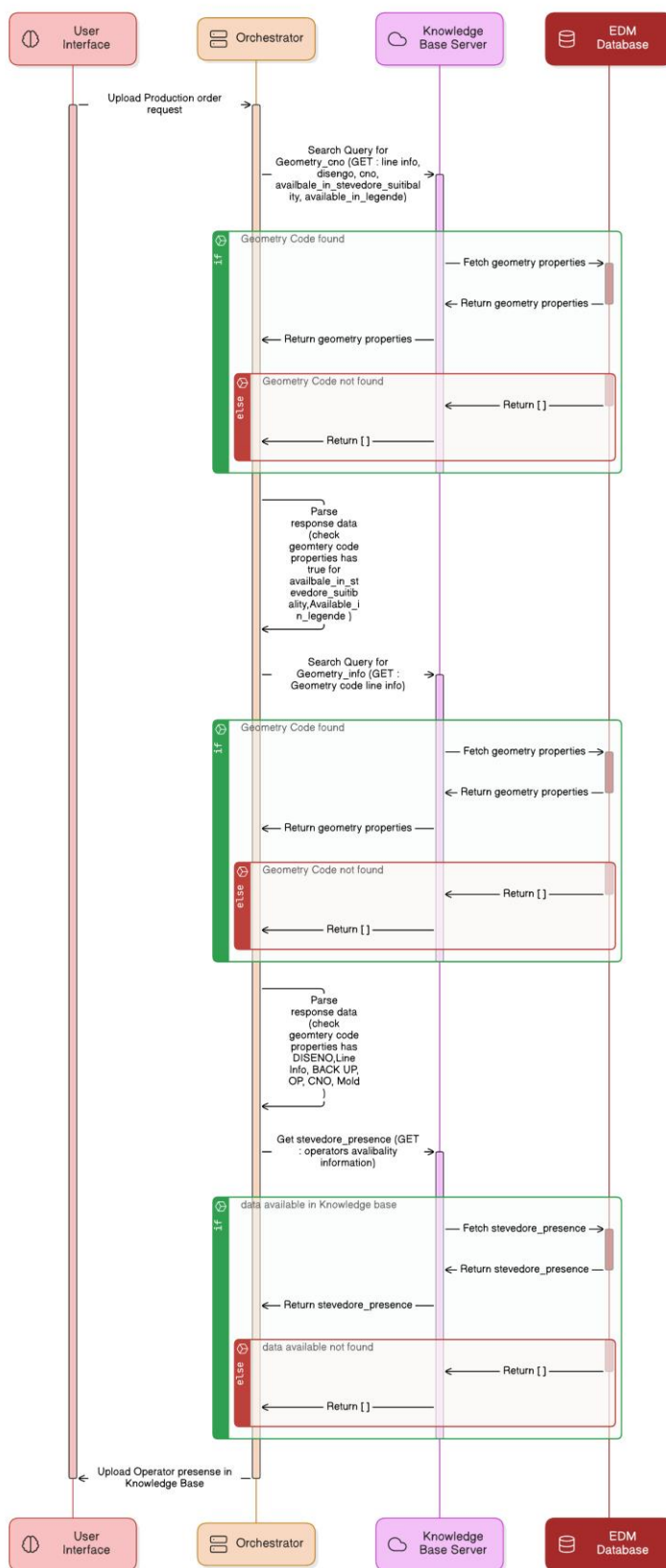


Figure 44: Sequence Diagram for DAI-DSS Interactions Part 1

Production Order Request Processing

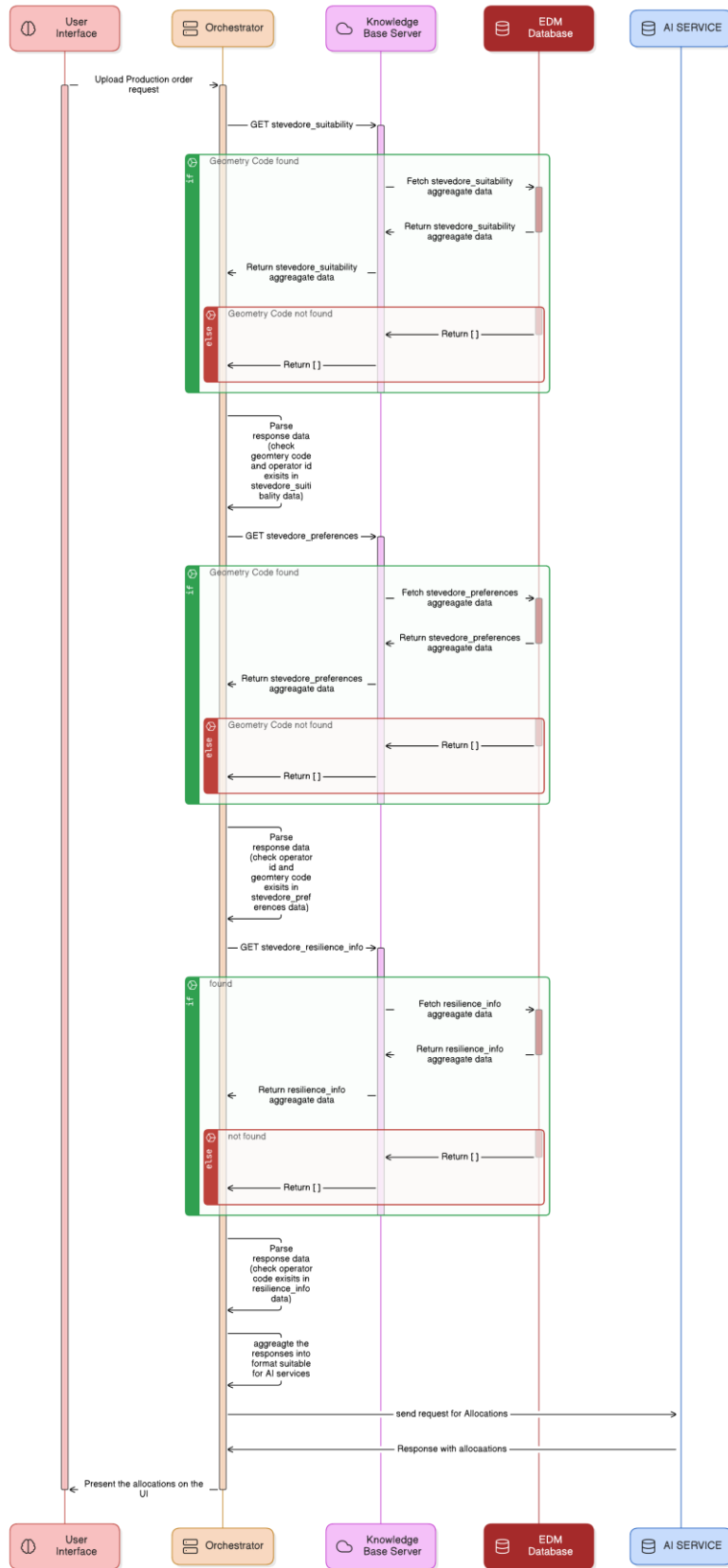


Figure 45: Sequence Diagram for DAI-DSS Interactions Part 2

## 3.4.1 Reliability

### 3.4.1.1 Definition of Reliability for a Data Source

Data reliability refers to the dependability and trustworthiness of data in supporting effective data lifecycle management and decision-making. Reliable data ensures that stakeholders can confidently rely on it to make informed decisions, as it consistently meets the standards for accuracy, usability, and consistency.

Key attributes of reliable data include:

- **Accuracy:** The extent to which data accurately represents the real-world values or events it is meant to reflect.
- **Consistency:** The absence of contradictions or discrepancies within datasets, records, or across different data versions.
- **Completeness:** The presence of all necessary information, ensuring no critical data fields are missing.
- **Timeliness:** The availability of data at the right time, ensuring it is current and relevant when needed.

### 3.4.1.2 Objectives of the Reliability of Knowledge Base

The objectives of ensuring the reliability of the Knowledge Base include:

- **Ensure Data Trustworthiness:** Ensuring that the stored data is accurate, consistent, and complete, fostering confidence in its validity.
- **Facilitating Informed Decision-Making:** Providing stakeholders with dependable data that serves as a robust foundation for decisions.
- **Promoting Ethical Use of Data:** Upholding the integrity of data, particularly in AI-driven or critical systems, to align with ethical standards.
- **Enhancing Data Quality Continuously:** Establishing processes for ongoing monitoring, auditing, and improvement to maintain and elevate data reliability over time.

### 3.4.1.3 Reliability Aspects Covered by Knowledge Base (ISO 10303)

#### 1. Accuracy

- **Standardized Data Representation:** The schema based on AP239 ensures that data is modelled consistently and structurally correct, minimizing errors in representation.
- **Controlled use of RDL classes/properties** ensures that attributes conform to a predefined ontology, enhancing semantic accuracy Figure 46.

Name	Inherited From	Type	Values/Expression	Units	RO	+	✎	✖
Line		Text			<input type="checkbox"/>			
MODELLO		Text			<input type="checkbox"/>			
DISEGNO		Numeric			<input type="checkbox"/>			
DENOMINAZIONE		Text			<input type="checkbox"/>			
CND		Numeric			<input type="checkbox"/>			
Available_in_stevedore_sustainability		Boolean	True, False		<input type="checkbox"/>			
Available_in_legende		Boolean	True, False		<input type="checkbox"/>			

Figure 46: Reference Data Library Definition

## 2. Consistency

- Hierarchical Data Organization: A Product Breakdown Structure (PBS) organizes data hierarchically, maintaining consistent relationships and dependencies between nodes and associated documents.
- Version Control: Implementing versioning prevents overwriting of previous data, ensuring that updates do not compromise consistency or historical records.

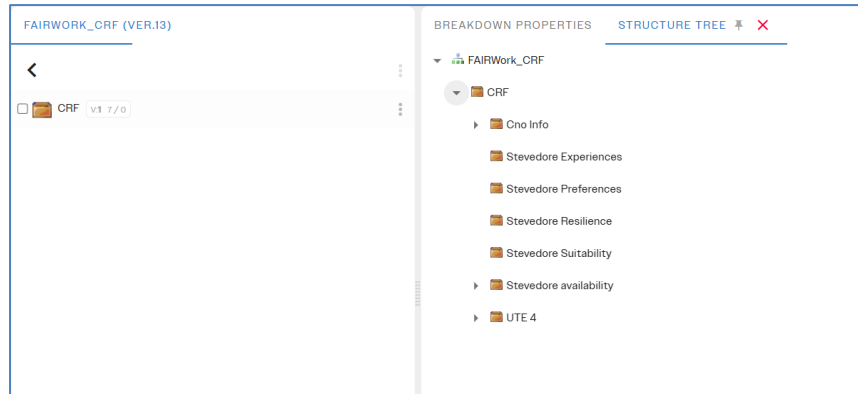


Figure 47: Breakdown Structure of the CRF Use Case

## 3. Completeness

- Mandatory Metadata Fields: Every node and document include essential metadata such as:
  - Creation Date: Origin of the data.
  - Updated By User: Information on ownership and modification history.
  - Description Field: Contextual details about the node.
- Default Metadata Attributes:
  - Figure 48 and Figure 49 shows the default metadata attributes that will be populated when a node or document has been created in the knowledge base. This metadata provides an overview of the created date, updated date, and person who updated this information.
- Traceability and Provenance: Version control ensures a complete and traceable history for every node or document, enhancing accountability and data lineage.

Name	Inherited from	Type	Description	Role	Subject
Name *		Identification	Name of the breakdown element	Breakdown_element_name	BREAKDOWN_ELEMENT_VERSION ✓
Type *		Classification_assignment	Type of the element	Node type	BREAKDOWN_ELEMENT_VERSION ✓
GUID *		Identification	Global unique id of the element	Breakdown_element_identification_code	BREAKDOWN_ELEMENT ✓
Instance ID *		Identification	Instance id of the breakdown_element in target data model	Instance ID	BREAKDOWN_ELEMENT ✓
Version *		Identification	Breakdown revision number of the element was creation	Version_identification_code	BREAKDOWN_ELEMENT_VERSION ✓
Description *		Descriptor_assignment	Description of the breakdown element	Breakdown_element_description	BREAKDOWN_ELEMENT_VERSION ✓
Created *		Date_or_date_time_assignment	Date time of the element creation	Creation_time	BREAKDOWN_ELEMENT ✓
Updated *		Date_or_date_time_assignment	The last date time of the element update	Date_actual_modified	BREAKDOWN_ELEMENT_VERSION ✓
Created by *		Person	User who created the breakdown element	Creator	BREAKDOWN_ELEMENT ✓
Updated by *		Person	The last user who updated the breakdown element	Updated by	BREAKDOWN_ELEMENT_VERSION ✓
Project Phase *		Classification_assignment	Project phase when the element was added	Phase	BREAKDOWN_ELEMENT_DEFINITION ✓

Name	Value
Name	Line 18
Type	Unit
GUID	2VlpOYcu0Hku0005MkOdM
Instance ID	751619301422
Version	30
Updated	11/5/2024, 3:17:00 PM
Created by	jotne_rishyank
Updated by	jotne_rishyank
Project Phase	0

Figure 48: Node System Properties – Metadata

Name	Inherited from	Type	Description	Units	Subject
Title *	Identification	Document title	Document_title	DOCUMENT_VERSION	✓
Name *	Descriptor_assignment	File name	File_name	DOCUMENT_VERSION	✓
GUID *	Identification	Global unique id of the document	Project_document	DOCUMENT	✓
Instance ID *	Identification	Instance id of the document	Instance ID	DOCUMENT	✓
Size *	Identification	Size of the document	File Size	DOCUMENT	✓
Revision *	Identification	Document version	Document_version_number	DOCUMENT_VERSION	✓
Description *	Descriptor_assignment	Description of the document	Document_description	DOCUMENT_VERSION	✓
Made from *	Descriptor_assignment	Made from the template of the document	Document_generated_from_template	DOCUMENT	✓
Comment *	Descriptor_assignment	Comment to the document version	Document_changes_description	DOCUMENT_VERSION	✓
Type *	Classification_assignment	Document content type	Document type	DOCUMENT	✓
Discipline *	Classification_assignment	Discipline	project_discipline	DOCUMENT	✓
Phase *	Classification_assignment	Project phase	Phase	DOCUMENT	✓
Source *	Classification_assignment	Information source	Info source	DOCUMENT	✓
Format *	Descriptor_assignment	File format	Document_format	DOCUMENT	✓

Name	Value
Title	CRF_Deliv_of_Materials_Shipment
Name	CRF_Deliv_of_Materials_Shipment_DS1_en_bpm
GUID	076804_44e0089WV6
Instance ID	4380872028
Revision	1001
Description	this is material shipment model
Discipline	Uncertain
Phase	0
Source	Unknown
Format	ipgm
Status	Approved
Updated by	gtrm_rishyark
Reviewer	gtrm_rishyark
Approver	gtrm_rishyark
Responsible	gtrm_rishyark
Release Manager	gtrm_rishyark
Updated	6/26/2025, 8:28:43 AM
Circle	0
IID	0
Document content type	sem:ill:AP242:Domain:DigitalFile
Size	0.028 Mbytes

Figure 49: Document System Properties - Metadata

#### 4. Timeliness

- Timestamps for Currency Verification: Metadata includes creation and update timestamps, allowing stakeholders to confirm the data's relevance and recency.
- The use of a REST API ensures that users have immediate access to the most up-to-date versions of data, supporting timely decision-making.

#### 3.4.1.4 Implementation Approach

##### 1. Data Storage

- Implementation Details:
  - Data is stored in a hierarchical product breakdown structure (PBS) modeled according to the AP239 schema, ensuring structured and standardized storage.
  - Versioning is implemented for each node to prevent overwrites and allow recovery of historical data.
    - Note: Nodes with aggregate properties are not versioned.
  - RDL classes/properties enforce semantic correctness and compatibility across projects.

##### 2. Data Validation

- Implementation Details:
  - Validation of incoming data against the AP239 schema ensures compliance with structural and semantic rules.
  - The system enforces mandatory metadata fields (e.g., creation date, updated user, and description) to maintain record completeness.
  - Version conflicts are handled via a controlled update mechanism, ensuring consistent relationships across nodes.

##### 3. Metadata and Provenance

- Implementation Details:
  - Metadata fields (creation date, updated by user, and description) are captured for every node to ensure transparency and traceability.
  - Version control is integral, enabling stakeholders to trace all changes and recover previous states if needed.
  - Logs are maintained for updates, capturing user actions and timestamps to support accountability.

##### 4. Real-Time Data Access via REST API

- Implementation Details:
  - A REST API provides direct access to the database, exposing endpoints for querying node data, metadata, and versions.

- APIs are designed to return the latest version of data by default but allow retrieval of historical versions for auditing purposes.
- The API ensures timeliness by enabling real-time access to updated and accurate information.

### 3.4.2 Outlook

To meet the data requirements of AI services, the Knowledge Base needs to be adapted to support efficient storage and retrieval. Currently, FLEX data files are centrally archived within the Knowledge Base. Integration may be required if new services are made available (Figure 50).

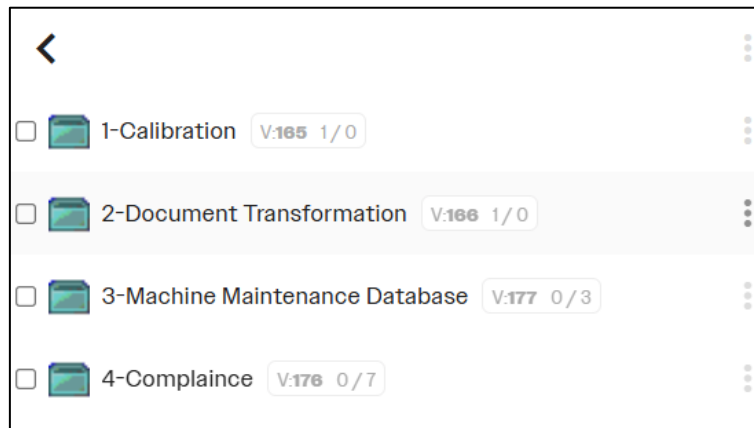


Figure 50: FLEX Data stored in the Knowledge Base

## 3.5 Integrating AI-Services

Within the FAIRWork project, several AI services for different use cases were developed to showcase different solutions to the problem situations and challenges. In order to sort all relevant input data from the UI and the Knowledge Base, the AI services are triggered by the Orchestrator, which collects and combines the different data in a JSON format that is sent to the AI service. The AI service runs the algorithm considering the relevant data and sends the result JSON back to the Orchestrator that then decides on what has to be shown to the user and what has to be saved in the Knowledge Base.

The requests between the UI, Orchestrator, Knowledge Base, and AI services are done via HTTP requests. Those requests can handle the JSON files that contain the relevant data in both ways: requests and results. Also, depending on the HTTP request, different services are triggered so that the user gets the results requested.

### 3.5.1 Support the Understanding of Decisions through Conceptual Modelling

The goal of this experiment is to create a decision service using a knowledge-based approach to support decision-making in the *Assist Decisions about Fair Worker Allocation* use case, which belongs to the resource mapping decision problem, which is introduced in FAIRWork's D2.1.

For this experiment, decision-makers should be enabled to encode their decision knowledge directly and are then supported in creating a decision service out of it, which can be used within the DAI-DSS. This service is connected to the goal described in the section 3.3.2, which enables the configuration of decision services. The configuration is supported through diagrammatic conceptual models, which are then used as input for the decision service. The

service itself offers an endpoint where the parameters are sent, and the decision is returned. This endpoint can then be used by the DAI-DSS Orchestrator.

The decision service itself was implemented in the free OLIVE<sup>9</sup> framework. Here, a connector was enhanced to consume the models and instantiate a decision endpoint out of it. To do so, the decision logic must be defined, which is, in this case, done by using models created with the *Decision Model and Notation (DMN)* language, which were created with the Bee-Up tool for this experiment.

The decision-making in this experiment is separated into two steps. One is to decide if a worker is allowed on a specific line to produce a certain product, and the second is to take the group of allowed workers for a group of production lines and assign them based on their preferences. Then, it is checked that no worker is assigned to multiple lines. In this experiment, we use the rule-based service configured through the models to make the decision if a worker is allowed on a production line. Then, the separate worker allocation endpoint is used as a preliminary prototype to assign the allowed workers to the line.

### 3.5.1.1 Prerequisite

To prepare a new decision endpoint within this experiment, three parts must be prepared. One is the Bee-Up modelling tool to define and describe the decision that should be made. The second and third parts consist of the needed services, which are an OLIVE instance and the worker allocation service.

As a modelling tool for this experiment decision service, the ADOxx-based Bee-Up tool was used, as it is freely available and had the modelling languages which were used for this experiment. To use it, first, the Bee-Up tool must be installed on the machine, and then the extensions must be added. Information on how the extension can be added is available on the GitLab project.

To use a new OLIVE instance with this experiment, the corresponding endpoints must be defined. This must be uploaded to the OLIVE instance, and then the DMN file must be reuploaded again to use everything (how the DMN file can be uploaded is described below).

To fully use the implemented decision service for this use case, an additional service is needed, which is called *Rule-based worker allocation*. Its task is to take workers who are allowed on certain lines and assign them to the different lines. Thereby, it takes the provided preferences of the workers and assigns those workers who have the highest preferences, making sure that one worker is not assigned to multiple lines.

If these three parts are available, the service can be used. Below, it will be described how it was used for resource allocation.

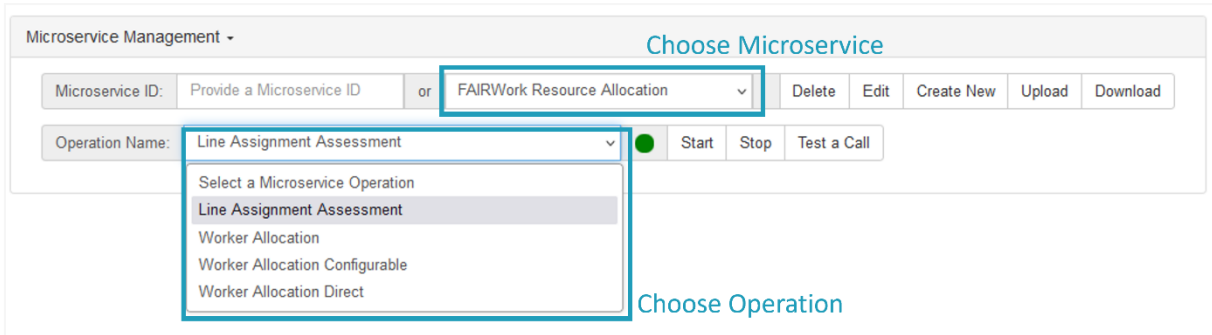
### 3.5.1.2 Testing the Existing Endpoint

This section describes how the publicly provided interfaces of this experiment can be used with test data. These public endpoints were also integrated for an overall decision made with the Orchestrator.

An example of the OLIVE UI can be seen in Figure 51. To test the provided functionality, one has to choose *FAIRWork Resource Allocation* in the microservice drop-down menu and then specify the operation. After the operation is chosen, the *Test a Call* button must be clicked.

---

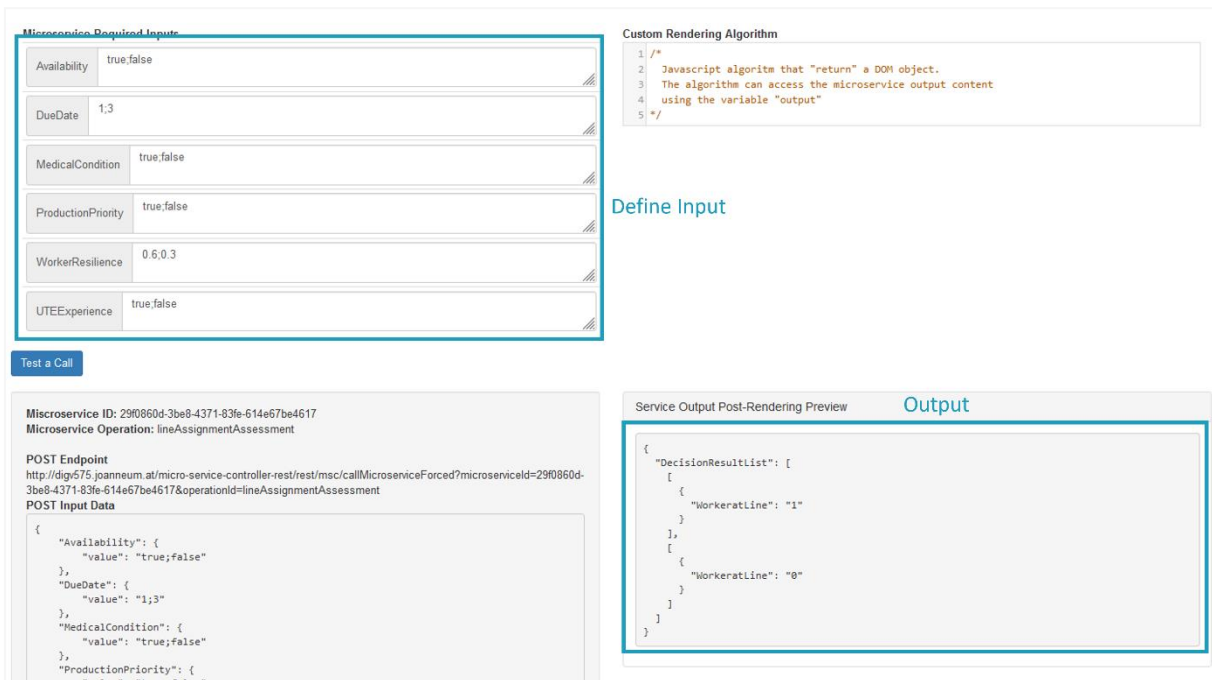
<sup>9</sup> <https://www.adoxx.org/live/olive> (accessed: 8.1.2025)



**Figure 51: OLIVE User Interface for Resource Allocation Experiment**

For testing the service defined with the model, the *Line Assignment Assessment* operation must be chosen (see Figure 52). This opens the interface for testing the decision if a worker can be assigned to a specific line. The input that should be used can be entered in the top left corner. For each parameter used, a field is provided. The values for different workers can be provided at once by separating the values with “,” (semicolon). The number of values for each parameter must be the same, the following values:

- Availability: “True” and “False”
- DueDate: Integer between -10 and 10
- MedicalCondition: “True” and “False”
- ProductionPriority: “True” and “False”
- WorkerResilience: Float between 0 and 1
- UTExperience: “True” and “False”



**Figure 52: OLIVE Interface for Testing the Line Assignment Assessment Operation**

*Service Output Post-Rendering Preview* shows the output of the decisions. For each set of values, an entry in the array with the key “WorkeratLine” (provided through the model) is given. The value “1” means that the worker can be assigned, and “0” means that the worker cannot be assigned by the line.



To test the full experiment, including the assignment of workers to specific lines, then the other operations *Worker Allocation* or *Worker Allocation Configurable* can be used. Both use the *Line Assignment Assessment* endpoint introduced above to decide which workers are allowed on a line and then use the results to assign the allowed worker to the given lines. The difference is that the second one allows you to choose the endpoint for the *Line Assignment Assessment* endpoint, and the first one has it pre-configured, so that just the information about the worker and the lines must be provided.

For the worker allocation examples, default values for the input are provided, which can be used once the interface is opened. Both can be executed, and the results can be returned in a JSON file containing the production line and the assigned worker with their preferences.

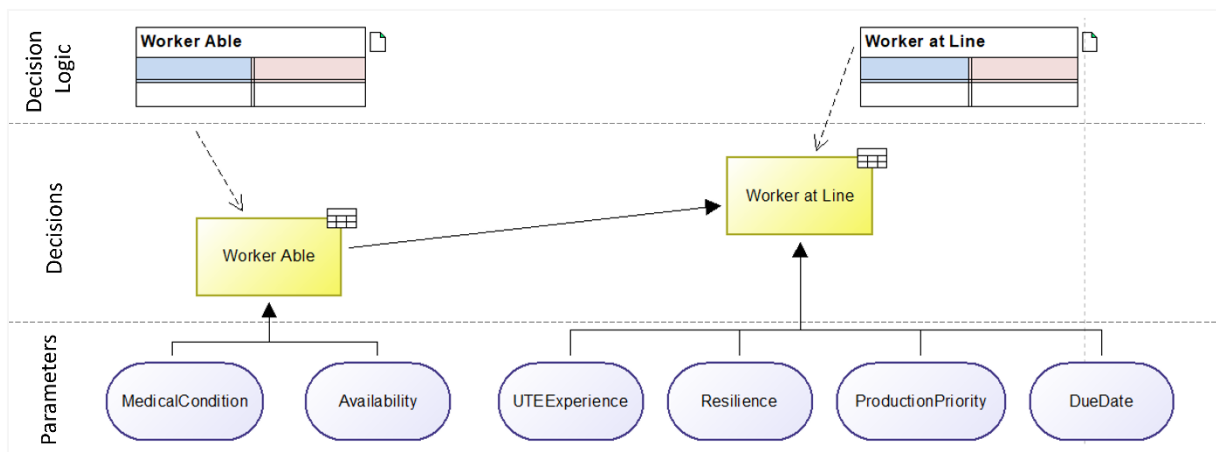
The endpoints can also be tested by sending POST HTTP requests to them, with the header *Content Type* set to *application/json*. The test data sent directly over the POST request does not completely overlap with the information provided in the OLIVE Interface.

### 3.5.1.3 Creating a New Decision Endpoint with Conceptual Modelling

If a new decision endpoint, using a rule-based approach should be established, the following steps must be taken. Here, we will not go into detail in how the knowledge for this decision can be gathered, but how it can be encoded so that it can be made usable as a decision service.

To decide if a worker can be assigned to a specific production line and the products that should be produced on them, a rule-based decision service should be used, for which first, a DMN model with the decision knowledge must be created.

The model contains a visual representation of the structure of a decision and the decision logic in the form of a decision table containing rules. The graphical model with some annotations can be seen in Figure 53. In the middle, the decision of whether a worker can be assigned to a production line, and the sub-decision, if a worker is on principle able to work on a line. At the bottom of the figure, one can see the parameters used to make this decision, and at the top, the objects containing the decision knowledge can be seen.



**Figure 53: DMN Model for the Worker Allocation Use Case**

For the parameters, the name and the type must be defined. The type can be defined in the attributes of each parameter object. The available attributes of the object can be shown by double-clicking it, and to save the type, the *Type reference* attribute is used. For sub-decisions (like *Worker Able* in Figure 53), a type must also be defined using its *Type reference* attribute. The *Type reference* of the main decision (*Worker at Line*, in this example) defines the decision service's result type. The following types can be used:

- string (in the CSV, strings must be start and end with quotation marks, e.g., "healthy")
- boolean
- integer
- long
- double

The goal of the model is to export the knowledge about the decision in an XML structure based on the DMN standard<sup>10</sup>, which can then be uploaded to the decision service. Therefore, the decision logic is saved in the *General purpose attribute* of the decision logic elements, which are instances of the *Boxed expression (DMN)* class in the modelling tool. But to make the writing of the decision rules more user-friendly, they can be written in CSV (e.g., using Microsoft Excel) and then automatically imported into the model. The modelling tool supports the handling of these definitions of decision rules, which are called decision tables in this context.

To use the DMN export functionality, the decisions must be linked to the *Boxed Expression (DMN)* objects using the *Linked Boxed Expression* attribute. Therefore, the decision object must be double-clicked, the *Linked Boxed Expression* attribute must be found in the *Decision logic* tab, the plus symbol must be clicked, and the corresponding *Linked Boxed Expression* object must be searched.

Under the menu entry *DMN Decision service*, the different functionalities that help define the decision logic and export it can be found. Here, the empty decision table with the correct parameters can be created through the *Create decision table*. To do this, an object of *Decision (DMN)* or *Boxed Expression (DMN)* in the modelling tool must be marked. Further, the decision table file can be opened with the standard system application by clicking *Open decision table* having marked an object of *Decision (DMN)* or *Boxed Expression (DMN)* in the modelling tool. Afterward, the decision rules can be added; an example is shown Figure 54. Each line represents one rule. Within the rules, either the direct value that must be set for a parameter to be defined, or an expression, like an interval, using [*start*..*end*] or >, <, >= and <=. After the decision table is defined, it can be loaded into the model by making the decision or the boxed expression and then clicking *Import decision table* under the *Decision service* menu when an object of *Decision (DMN)* or *Boxed Expression (DMN)* is marked in the modelling tool.

Worker Able	UTEEexperience	Resilience	ProductionPriority	DueDate	Worker at Line
TRUE	TRUE	[0.0..1.0]	FALSE	[-10..10]	1
TRUE	TRUE	[0.0..1.0]	TRUE	[-10..10]	1
TRUE	FALSE	>=0.5	FALSE	[-10..10]	1
TRUE	FALSE	>=0.5	TRUE	[-10..10]	1
TRUE	FALSE	<0.5	TRUE	[-10..10]	1
TRUE	FALSE	<0.5	FALSE	<=1	1
TRUE	FALSE	<0.5	FALSE	>1	0
FALSE	FALSE	[0.0..1.0]	TRUE	[-10..10]	0
FALSE	TRUE	[0.0..1.0]	TRUE	[-10..10]	0
FALSE	FALSE	[0.0..1.0]	FALSE	[-10..10]	0
FALSE	TRUE	[0.0..1.0]	FALSE	[-10..10]	0

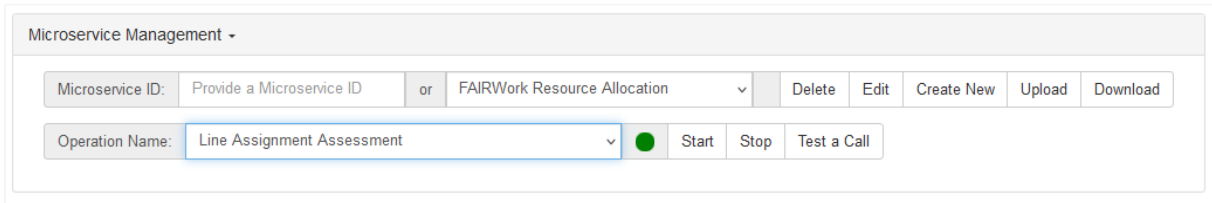
**Figure 54: Example Decision Table**

After the model is completely defined, one can create the DMN file, which can be used as the input for creating the decision endpoint. This endpoint can be created in two ways, one is to manually upload the DMN file and the other is to add the information of the OLIVE instance to the model and automatically upload the file to instantiate the endpoint.

<sup>10</sup> Object Management Group. (2023). Decision Model and Notation (DMN).

To manually upload the DMN file, the *DMN export* menu entry under the *Decision service* must be clicked. At the end of this algorithm, all the information needed for configuring the new decision endpoint is shown. This is the *Decision key* and the *Decision variables* that will be needed later.

To create a decision endpoint, the OLIVE web interface must be opened in a browser, as shown in Figure 55. To define a new decision endpoint, an existing microservice must be chosen over the drop-down menu, or a new one must be created. Then, over the *Edit* button, a new endpoint can be added.



**Figure 55: OLIVE Web Interface**

The interface for adding a decision endpoint can be seen in Figure 56. Here, a new *Operation* must be added, and then the needed information must be provided, like the *OperationID*, *Name*, and *Description*. As *Connector*, the *Camunda DMN Engine Connector Multiple Input* must be chosen. Then, the exported DMN file from the modelling tool, can be uploaded to the *DMN File Path*, and also the *Decision Key* and *Decision Variables* must be configured as provided at the end of the export algorithm of the DMN file export of the modelling tool. Finally, the *Call Configuration Inputs*, which are the parameters the REST call will consume, must be defined. Their names can be freely chosen, but the *Matching Name* must correspond to the used in between the % signs in the *Decision Variables*. Afterward, this configuration must be closed by using the *Continue* button on the end, and then the rest of the endpoint can be used.

Microservice Definition

Microservice Name: FAIRWork Resource Allocation | Description: This Microservice contains operations for the resource allocation | Is Public?  | Show Details

New Operation

Operation Line Assignment Assessment - Delete

Operation ID: lineAssignmentAssessment | Name: Line Assignment Assessment | Description: Line Assignment with data from kno... | Is Default?  | Autostart?

Connector: Camunda DMN Engine Connector Multiple Inp... | A connector to an internal Camunda DMN Engine library

Start configuration: DMN File Path: \_3b79c5ed-e9e9-4381-946b-6b58939c86f5/workerAllocation-2023-Nov-22.dmn | Upload

Call configuration: Decision Key: Decision\_32603 | Decision Variables: [{"Availability": "%Availability%", "DueDate": "%DueDate%", "MedicalCondition": "%MedicalCondition%", "ProductionPriority": "%Production"}]

Call Configuration Inputs

Input ID	Matching Name	Description	Working Sample
Availability	%Availability%		true,false
DueDate	%DueDate%		1,3
MedicalCondition	%MedicalCondition%		true,false
ProductionPriority	%ProductionPriority%		true,false
WorkerResilience	%Resilience%		0.6;0.3
UTEExperience	%UTEExperience%		true,false

Figure 56: OLIVE Microservice Configuration Interface

The automatic instantiation directly from the modelling tool, was added as an extension to the version described in D4.2. The extension is based on the concept introduced in D3.2.

The automatic instantiation reduces the manual configuration effort and allows users to quickly deploy, change, and test the defined rules in a service. Further, different environments can be connected, allowing them to easily deploy the rules in a testing or productive environment.

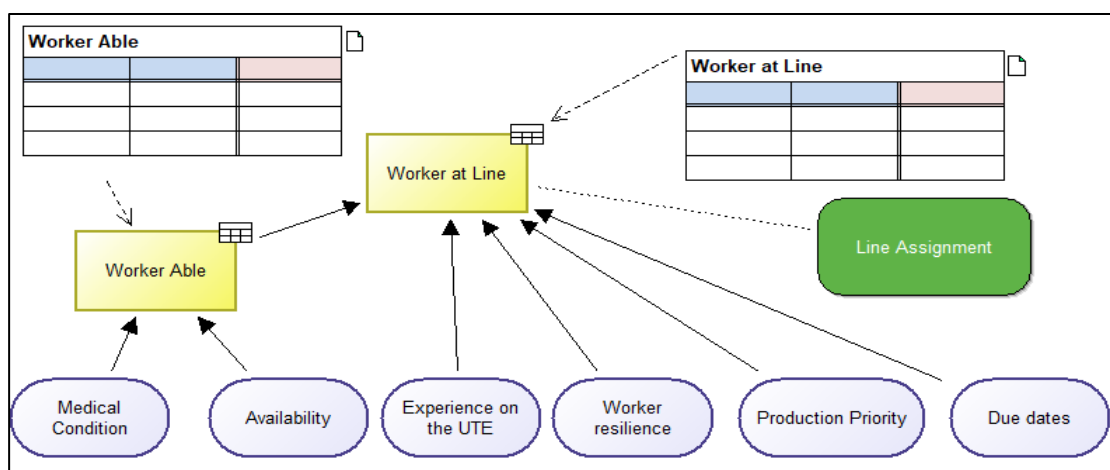


Figure 57: Example DMN Model with Microservice Definition Object

To achieve this, the standard DMN language was not enough and had to be extended to not only contain the decision logic but also allowing to save technical information needed to instantiate the service. Therefore, we enhanced the Bee-Up tool and its DMN language with an additional concept which we called *Microservice Definition* (shown as the green box in Figure 57). This concept allows to specify the information needed to instantiate the rule-based decision service within a running OLIVE Controller. As prerequisite, the generic connector implementation for the decision service must be implemented in the OLIVE Controller. This generic implementation is then instantiated with the information provided in the model, which together with the decision logic is then gathered by the modelling tool and sent to the OLIVE controller, instantiating a decision service, which can then be called over a REST interface.

Before the decision service can be instantiated on the OLIVE instance, the *Microservice Definition* object must be created and related to a *Decision (DMN)* object using the *Association (BPMN)* relation. An Example of such a model can be seen in Figure 57. The instantiation can be triggered in the modelling tool by clicking *Decision service* in the menu bar and then *Instantiate Microservice*.

The information that can be added to the model is shown in Figure 58. The First three attributes are used only internally in the modeling tool to identify the object and adapt its visualization. The important information for the instantiation of the decision service is the OLIVE *controller* attribute. Here the base URL of the used OLIVE controller must be provided. For FAIRWork we set up an instance on the server provided by JR.

The information afterward is separated into information about the microservice and the operation. More detailed information about OLIVE and its concept of microservice and operations can be found at OLIVE (Falcioni, 2021)<sup>11</sup> For this section, it is important to know that a connector is a generic implementation of an endpoint, which can be instantiated and configured to fit a specific case. A microservice is a collection of instantiated connectors that are configured to be used together. The attributes of the decision service are the following:

- **Microservice id:** This is a unique ID for the microservice on this instance of the OLIVE controller. If an ID is provided it is used to create or update an existing microservice. If this ID is empty, a new service with a unique ID is created, which is then saved in the model. If the microservice ID already exists on the OLIVE instance, the provided operation will be overwritten.
- **Microservice name:** This is a human-readable name for the microservice.
- **Description:** A short description of the service.
- **Operation name:** An identifier for the instantiated connector, which is unique within the microservice. If this name already exists for this microservice, it will be overwritten.
- **Operation description:** A short description of the instantiated operation.
- **Operation is autostart:** Checkbox to define that the operation starts automatically
- **Operation is default:** Checkbox to define that this operation is the default operation of the microservice.

---

<sup>11</sup>Falcioni, D., & Woitsch, R. (2021). OLIVE, a Model-Aware Microservice Framework. In E. Serral, J. Stirna, J. Ralyté, & J. Grabis (Eds.), *The Practice of Enterprise Modeling* (pp. 90–99). Springer International Publishing

Microservice definition for DAI-DSS Prototype (Microservice Definition)

Name:

Show operation name  
 Show olive controller

Olive controller:

Microservice information

Microservice id:

Microservice name:

Description:

Operation configuration

Operation name:

Operation description:

Operation is autostart  
 Operation is default

**Figure 58: Attributes of the Microservice Definition Modelling Concept**

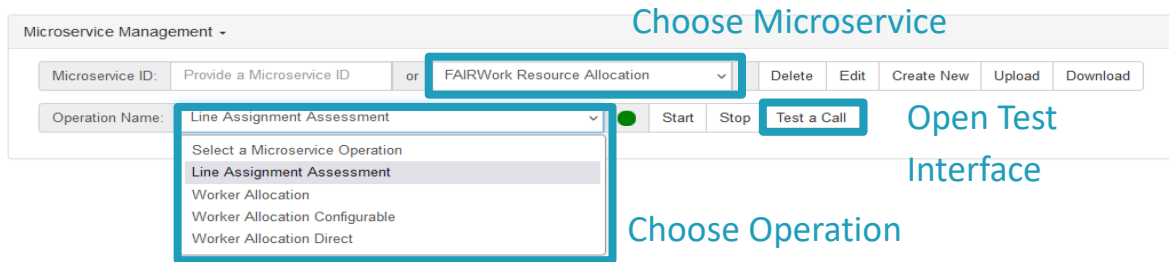
The sources for the Bee-Up extensions can be found in:

- <https://code.omilab.org/research-projects/fairwork/decision-services/bee-up-dmn-extension>

To use this extension, one cannot install the Bee-Up tool but must install the ADOxx<sup>12</sup> metamodeling platform and import the library. More information can be found in the GitLab repository.

Independently, if the decision service endpoint was instantiated manually or automatically, it is available as an OLIVE REST endpoint which can be used by the DAI-DSS Orchestrator. Additionally, a test interface is available which can be opened on the web interface, by choosing the microservice, then the operation, and click on the *Test a Call*. An example for the OLIVE web interface is shown in Figure 59.

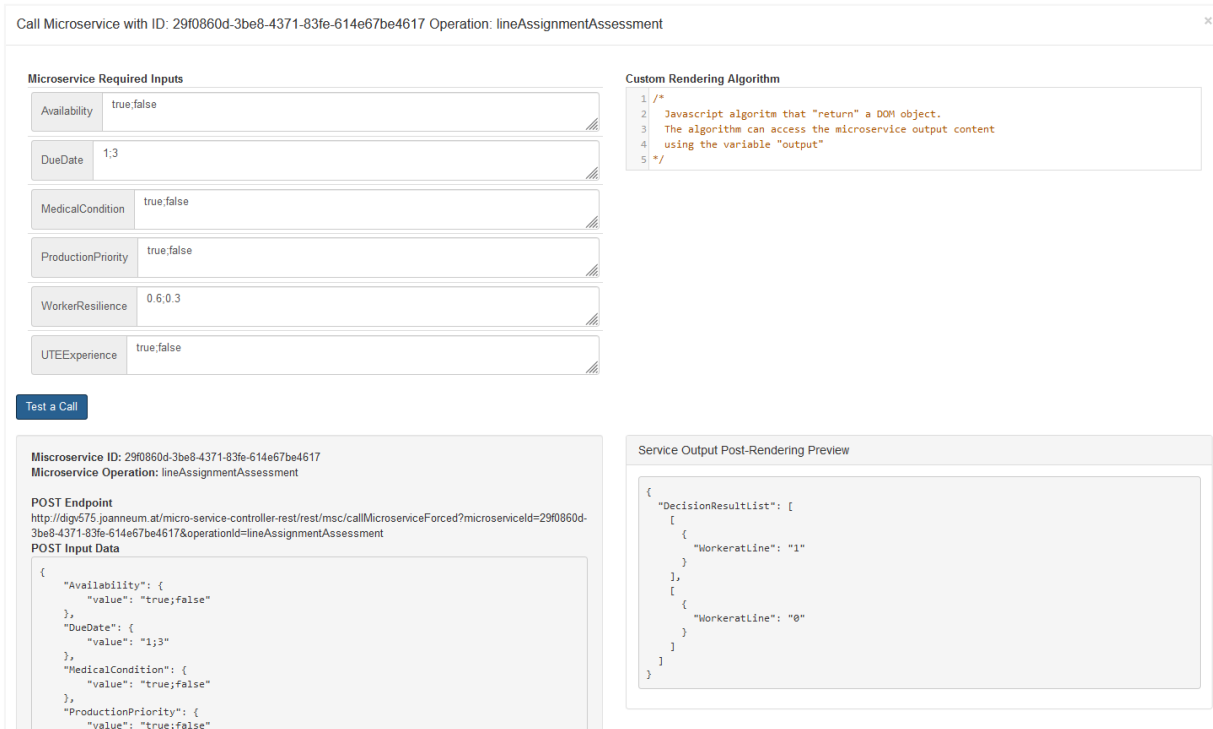
<sup>12</sup> <https://www.adoxx.org/> (accessed: 8.1.2025)



**Figure 59: OLIVE Controller Interface**

This opens the windows shown in Figure 60. Here, values for the defined parameters can be defined, and with the *Test a Call* button, the call can be made. When it is finished, the result is also shown in this window. But this endpoint can now be called with any tool that is able to make POST HTTP calls. The concrete endpoint for this configured decision service can be found in the test interface under *POST Endpoint*.

After this configuration, the endpoint, making the decision can be integrated into the overall decision-making procedure created within the DAI-DSS.



**Figure 60: OLIVE Test Interface**

### 3.5.1.4 Reliability

This experiment and the general procedure that comes with it (as introduced in FAIRWork D3.2), support reliability by using conceptual models to understand the decision logic of the stakeholders. Diagrammatic conceptual modelling, as a purpose-oriented abstraction of complex systems or situations, enables stakeholders to better understand how the decision is made. To understand the decision scenario and configure the decision support the related methodology uses steps which are again supported by conceptual models, starting in a workshop and later defining it in more detail.

Further, the automatic reuse of the modelled information to create the decision service directly uses the captured knowledge without the need to manually translate therefore reducing the possibility for human error during the translation. Additionally, defining the decision logic independently in models enables us to instantiate the decision service in different environments, like an experimentation environment or a productive one. As the model stays the same, the resulting decision service will behave the same way.

### 3.5.1.5 Outlook

This experiment was used to analyze and showcase one possibility of using modelled information directly as input for decision services, for one symbolic AI approach. In other experiments and services within FAIRWork, other ways of using modelled knowledge as input for decision services. In this context further work focuses on how different services can be integrated with conceptual models, to ease the knowledge exchange. Future work focuses on both sides, what must the modelling method be able to capture and which functionality must be supported. Additionally, the service design must consider the possibilities for configuration or understand the modelled information as input.

But not only individual implementations on both sides (modelling tool and service) should be considered but a general characteristic should be identified to ease the future integration of new modelling methods or services, increasing the flexibility of the approach. In OMILAB this integration is used and will be further explored within their *Digital Innovation Environment (DIEn)*, where a flexible usage of domain-specific modelling methods with services is needed to establish early experiments to evaluate innovative ideas.

## 3.5.2 Decision Support through Decision Tree

This section describes an early-stage experiment for using decision tree as configurable decision services with the DAI-DSS. Therefore, the aim was not to create one decision tree, best suited for a specific decision but to analyse if and how decision trees can be adapted to new or changing decisions. Here, a configurable service should be used to allow the creation of different trees for different decision problems.

In the context of this decision service, a model-based approach was also used. This aligns with the concepts introduced in section 3.3.2. The difference to the service introduced in section 3.5.1 (the rule-based service) is that decision trees are counted to machine learning approaches, meaning that they are not created by defining the concrete knowledge but by providing data, and the decision tree is derived from it. The models in this experiment should be used to visualise the decision tree so that users also have a visualisation of how a decision is made. Having a comprehensible visualisation capability is also one benefit of decision trees. Additionally, the models should be used to create test training data for the decision trees.

For this experiment, two parts were created the *Decision Tree Resource Allocation* service and a modelling environment, where the decision trees can be visualised. As a decision service, a Python-based service was created, and for the conceptual modelling, the *Decision Model and Notation (DMN)* implementation of the Bee-Up<sup>7</sup> modelling tool was enhanced.

The main endpoints are the one for triggering a training of a decision tree and the one where a decision tree can be used to provide an answer. For training the different parameters with their values, the wanted output and an identifier must be provided. For getting a decision, the parameters and their values must be provided. The additional provided endpoints can be used to gather information about the available decision trees.

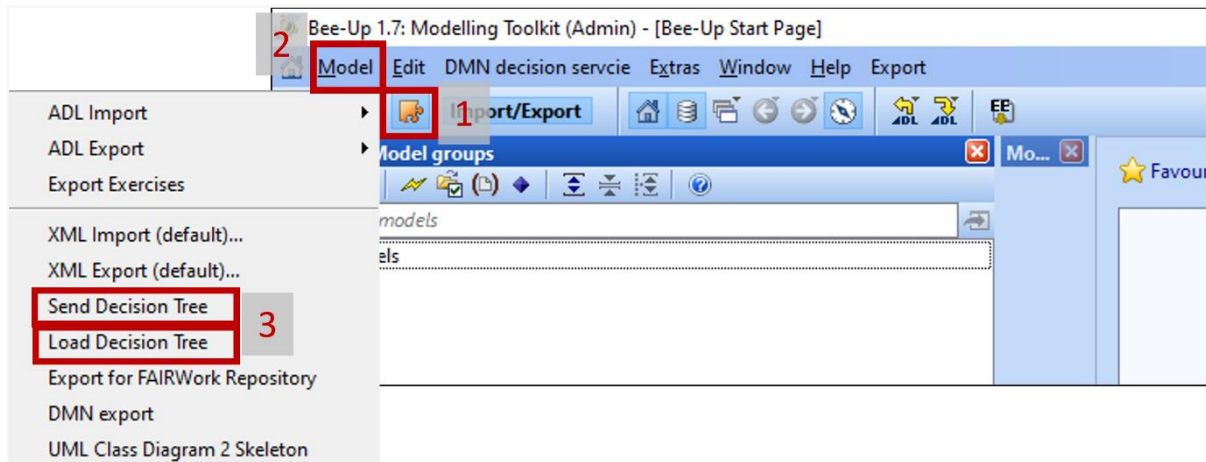
To use the conceptual modelling approach, the Bee-Up tool must be installed, and the extension must be loaded into the modelling tool. More information on that can be found on the experiments GitLab page<sup>13</sup>. This extension

---

<sup>13</sup> <https://code.omilab.org/research-projects/fairwork/> (accessed: 18.02.2025)



uses the DMN models, which is already available and adds to functionalities. They can be triggered in the modelling tool by opening the *Import/Export* component, then click on *Model* and choose *Send Decision Tree* or *Load Decision Tree*. An example model for a decision tree can also be found on the GitLab project. Figure 61 provides the steps on how to trigger the functionality.

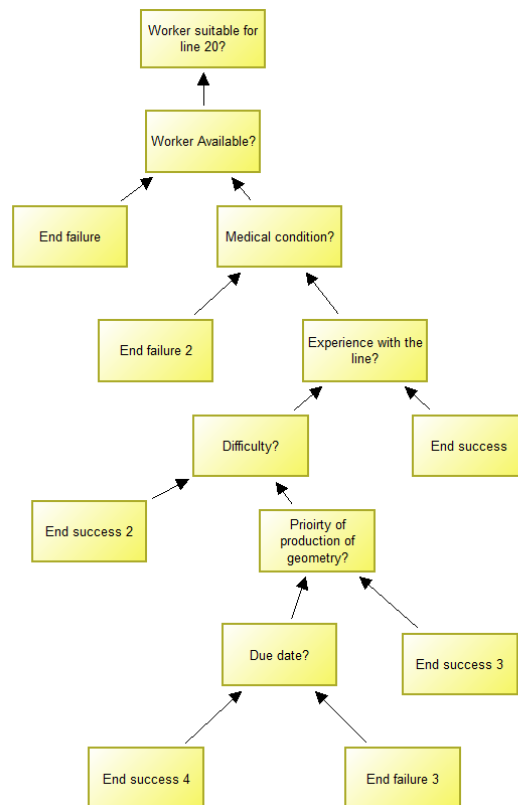


**Figure 61: Triggering Decision Tree Functionality in the Modelling Tool**

*Load Decision Tree* is to visualise an already trained decision tree in the modelling tool. Therefore, the identifier of the decision tree and the used endpoint must be provided. Afterwards, the data is gathered, and the decision tree is visualised. An example can be seen in Figure 62. Additional information about the decision is saved within the model.

Within this experiment, another functionality of the modelling tool was provided. The idea is that if a new decision case should be supported, one does not necessarily have all the historical data available to train a decision tree. Based on the procedure introduced in section 3.3.2, we want to enable people to create test data for the new decision based on the established decision process and decision models. For this experiment, it is possible to create mock-up data out of a modelled decision tree, which can be used as the basis for training a decision tree. This can then be integrated into the overall prototype and evaluated.

For the preliminary experiment introduced here, this data can be created. Therefore, a model like the one in Figure 62 must be created. The leaves are then the end of the decision and must contain the information on what the result should be. Each other node must contain the possible values for this parameter. The relations in the tree must contain the conditions of which path should followed based on the linked node. Additionally, the highest node is the root node, containing the information on how the result should be called and what possible results are available. For example, a Boolean if a worker is allowed on a specific line or not. What should be mentioned here is that in this way, the logic of the decision is described and is not necessarily how the trained decision tree looks. If the decision knowledge should be encoded directly, a knowledge-based approach like the one introduced in section 3.5.1 can be used.



**Figure 62: Example of Decision Tree in the Modelling Tool**

To save the needed information in the nodes, the *General purpose attribute* is used. The information must be provided in JSON format. Below, you will find an overview of the different types of information that must be provided for the different nodes:

- Root note
  - Example: {"possibleValues":[0,1],"type":"root"}
    - Possible values of the outcome of the tree
- Leaf:
  - Example: {"type":"end","result":0}
  - What is the result of this end
- Intermediate node:
  - Example: {"possibleValues":[0,1],"type":"node"}
    - Possible values which can be defined for this parameter
- Relation between nodes
  - Example: {"operator":"=", "value":1}
    - Condition when this path is followed. Consists of operator and value

This service was not changed from the one provided in section 2.6.2 of FAIRWork D4.2.

Implementing the prototype gave us insight into how to combine conceptual modelling with the decision services but was not further investigated as the rule-based approach seemed better fitting to the project. Further, machine learning approaches are also used by other partners, and therefore we focused on a symbolic AI approach with the rule-based system, which also enabled us to use it together with conceptual models.

### 3.5.2.1 Reliability

The reliability in this prototype is based on using conceptual models to support the explainability of the decision scenarios. The idea for this experiment was not only to reuse the modelled knowledge but to enrich models with information from the DAI-DSS. This concrete experiment was not further developed, but the idea was taken into the general approach for integrating information into models, which is described in FAIRWork D3.3.

### 3.5.2.2 Outlook

This experiment was not further developed, as other approaches were pursued within the project. But one core idea of representing the results of the decision service within models was taken and further developed in the conceptual modelling research track of FAIRWork D3.3.

## 3.5.3 Resource Allocation using Neural Networks

The Resource Allocation using Neural Networks aims to assist in the decision of allocating workers to machines in a fair manner within a production environment. Allocating workers to machines is a routine but crucial task in manufacturing industries. Decisions must be made daily regarding which worker is assigned to operate which machine. Traditionally, experienced employees manually handled this allocation in the CRF-use case.

The allocation process, initially performed manually by experienced employees, generated substantial historical data over time. This historical data comprises past allocations and is a valuable resource for optimizing the worker-machine assignment process. The allocation performed by experienced employees is a function that maps available workers to specific machines. This function can be approximated using a neural network, allowing the network to learn and mimic the solution strategies employed by experienced employees without explicitly encoding them.

The neural network employed in this project was implemented using the PyTorch library, a powerful tool for deep learning applications. The primary objective of the neural network is to predict the "final allocation" column in a given table based on various input parameters (see Figure 63). By leveraging supervised learning techniques, the neural network is trained on a diverse dataset, encompassing numerous examples of worker-machine allocations.

This iterative learning approach enables the neural network to discern complex patterns and relationships inherent in the historical data, effectively capturing the decision-making strategies employed by experienced employees during manual allocations.

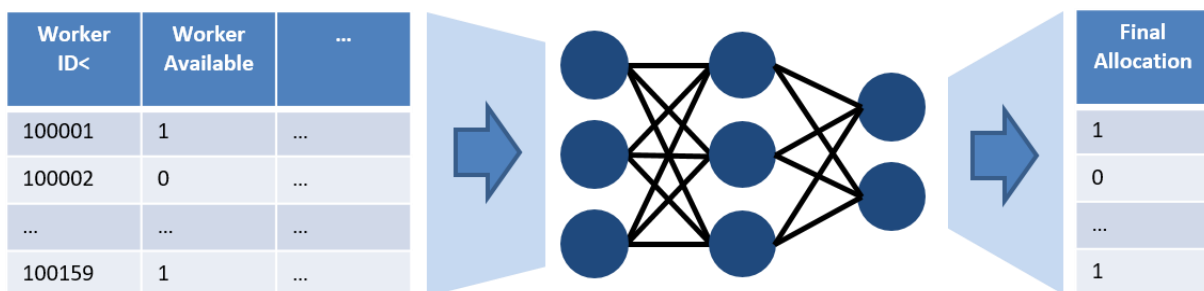


Figure 63: Neural Network Resource Allocation

The data structure used in the demonstrator can be visualized in tabular format (see Figure 64). The central challenge revolves around predicting this table's "final allocation" column. The neural network model, implemented with PyTorch, is made accessible through a REST API, providing an interface for interaction with other components. The deployment of this neural network on a server is documented with the Swagger Documentation Standard,

which enhances accessibility and usability, allowing users to submit their data and receive predictions via a UI in a web browser. The Swagger UI is incorporated purely for documentation and test purposes. In the overall system, the orchestrator interacts with the services via HTTP-calls.

	Worker ID	Worker	...	Final
10	10	100001	1	1
10	10	100002	0	0
...	10	...	...	...
11	...	...	...	...
11	...	...	...	...
	100159	1	...	1

Figure 64: Historic Data

The training process involves adjusting the network's weights and biases to minimize the disparity between the predicted "final allocation" and the actual values in the training set. Hence, the quality of historical data significantly influences the effectiveness of the Pattern-Based Resource Allocation Service. Since the neural network learns from past allocations, any biases, inefficiencies, or errors in the historical data will be reflected in future decisions. High-quality data—characterized by consistency, accuracy, and optimal past allocations—enables the model to generate reliable and efficient solutions. Conversely, if the historical data contains suboptimal decisions, inconsistencies, or incomplete records, the service may replicate these flaws, leading to inefficient or biased resource distribution. Therefore, ensuring high-quality training data is crucial for maximizing the service's performance and achieving optimal allocation outcomes.

3.5.3.1 Reliability

Resource Allocation using Neural networks fulfills the reliability factors to a high degree, though its effectiveness depends on data quality and system integration. Accuracy is largely achieved through machine learning models that analyze structured input data; however, its reliability depends on the correctness of the data provided. Consistency is maintained by neural networks that ensure uniform allocation principles, though discrepancies in input data may still introduce occasional inconsistencies. Completeness is well-supported as long as all relevant workforce, machine, and task information is available, but missing or outdated records can limit its effectiveness. Timeliness is highly dependent on the data update frequency - real-time or near-real-time updates enable dynamic and responsive allocations, while outdated information can reduce efficiency.

3.5.3.2 Outlook

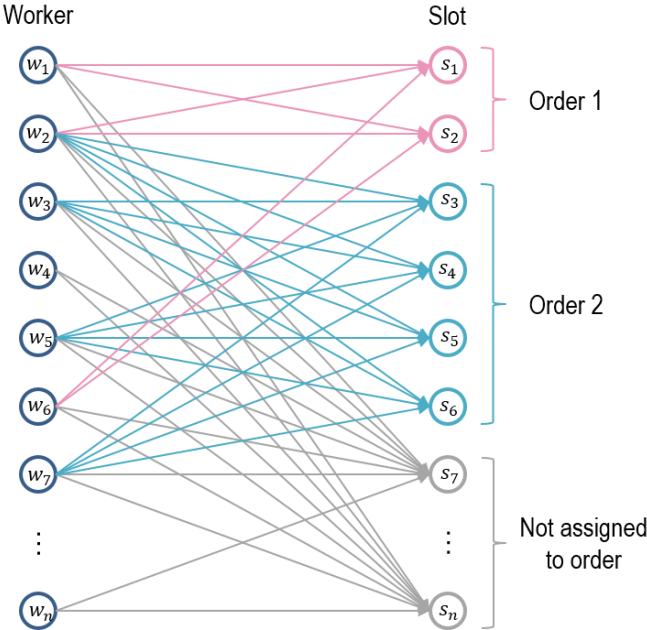
The performance can only be validated within a validation environment at CRF. As the amount of historical data is limited, it would need to be tested in the existing environment. Hence, the performance evaluation is still an open point. Also, the algorithm is trained to allocate workers for one specific plant. The way it is designed it can be trained for other plants by inputting the corresponding historical data.

3.5.4 Resource Allocation using Linear Sum Assignment Solver

The Linear Sum Assignment Solver (LinSumSolver) deals with the use-case in assisting in the decision of allocating workers to machines. The problem instance has several workers and several slots on a machine. One machine, or more specifically, one order processed on a machine, requires different amounts of workers to accomplish that order on the specified machine. Some workers cannot perform certain tasks and, therefore, not be assigned to

specific slots. For a given cost function that defines how "good" or "bad" the assignment of a worker to a specific slot is, a cost-optimization problem can be determined. It is required to assign at most one agent to each slot and at most one slot to each worker to minimize the total cost of the assignment.

Assignment Problems are a generic class of problems, and optimization libraries like Google OR-Tools have specialized solvers for certain assignment problems. In the case of the demonstrator, one can represent the allocation problem as a Linear Sum Assignment problem. An example problem instance is visualized in Figure 65.



**Figure 65: Problem Instance in Linear Sum Assignment Graph Representation**

Every order introduces several slots, encoded as nodes, that need allocation. A node is introduced for every worker, regardless of whether that worker is present. In Figure 65, order one introduces two slots; order two, four slots. For every worker that is suitable to be assigned to a slot, an edge is introduced between the node encoding the worker and the slot. The cost function  $f_c$  determines the weight of the edge. The cost function can be any composition of functions and mapping of the properties of a worker and the order that results in a scalar value. The implementation of the library requires that the number of lots equals the number of workers. To adhere to that requirement, one can introduce not-assigned slots. For every worker and every not-assigned slot, an edge is introduced with a constant weight that is a supremum of the cost function. That way, optimizing the cost still corresponds to optimizing the assignment. The not-assigned slot weight offsets the total cost, which does not influence the cost optimization. The cost of assigning a Worker to a slot can be organized in a matrix.

This service efficiently ensures optimal allocations by meeting machine requirements (e.g., each machine needing  $n$  workers), respecting worker constraints (e.g., skill limitations preventing certain assignments), and considering personal preferences. By integrating linear sum assignment, the solver guarantees mathematically optimal workforce allocations. Unlike heuristic or machine-learning-based approaches, this method provides provably optimal solutions that balance productivity, efficiency, and worker satisfaction. The result is a streamlined assignment process that enhances operational efficiency while ensuring fairness and compliance with workforce constraints.

One limitation of the Linear Sum Assignment Solver lies in its computational complexity, especially for large-scale problems. As the number of workers and machines increases, the size of the cost matrix grows exponentially, which can lead to significant processing times. Thus, they can become inefficient when handling highly complex

assignments with large datasets. In such cases, the time required to find an optimal solution may increase, making it less practical for real-time or large-scale applications without further optimization techniques or approximations.

#### **3.5.4.1 Reliability**

The reliability of the LinSumSolver is shaped by several key factors, including accuracy, consistency, completeness, timeliness, and the effective use of user-defined weights for goal prioritization. The LinSumSolver guarantees an optimal solution by considering predefined goals, which are prioritized according to the weights assigned by the user. Conflicting goals are resolved by favoring those with higher weights, ensuring that the allocation aligns with the most critical objectives. However, the quality of the solution heavily depends on the accuracy of the underlying model and the appropriateness of the weights, as these directly impact the final allocation. Accuracy is ensured as long as the input data and cost matrix are correctly defined, but errors in data or weight assignment can lead to suboptimal outcomes. Consistency is maintained through the solver's mathematical approach, ensuring that the same optimal result is produced for identical inputs. Completeness depends on the full representation of all constraints and goals within the model, with missing or incomplete data affecting the solver's ability to generate accurate solutions. Timeliness is influenced by problem complexity, as the solver's computational time increases with larger datasets and more complex scenarios. Overall, the service is highly reliable for well-structured, smaller-scale problems but may face challenges with very complex, large-scale assignments.

#### **3.5.4.2 Outlook**

The LinSumSolver could serve as a benchmark for small planning horizons, offering optimal solutions that help validate or calibrate more complex algorithms used in services with longer planning horizons. By providing a reliable standard for comparison, it can support the development and refinement of advanced planning tools for large-scale, long-term resource allocation scenarios.

### **3.5.5 Production Planning Service with a Hybrid Approach**

This service addresses the use-case of assisting in the decisions about production planning where suggestions for resource allocation for the production plan are made. This service provides a suggestion for a two-week planning horizon (see Figure 66) in two steps. First, it solves a job-shop problem by developing a schedule for the production which includes which parts are to be developed on which machine at which time and order. Second, it assigns the workers to the tasks. That way this combination can offer a full production plan based on an order list and shift plan. Therefore, the combines Constraint Programming (CP) for job shop scheduling and Monte Carlo Tree Search (MCTS) as a Reinforcement Learning approach for worker allocation. This hybrid methodology ensures an optimized production schedule while efficiently assigning workers to tasks, balancing both machine utilization and workforce constraints.



## Production Plan Week 49

Line	Shift	Monday			Tuesday		
		Part(s)	Amount	Worker(s)	Part(s)	Amount	Worker(s)
<b>17</b>	1	55555	600	Michael Jackson	44444	250	Elton John
				Jimi Hendrix	33333	700	Tina Turner
							Bob Marley
	2	55555	600	John Lennon	22222	1540	Led Zeppelin
				Evis Presley			

**Figure 66: Extract of Production Plan (one line from several, two days from two weeks)**

The first step in the service uses CP to solve the Flexible Job Shop Problem by determining an optimal production schedule. CP formulates the scheduling problem as a set of constraints, such as machine capacities, task dependencies, and processing times, and systematically searches for feasible solutions while minimizing costs. By applying constraint propagation and systematic search, CP efficiently allocates tasks to machines, ensuring high machine utilization while meeting production deadlines. The result of this stage is a structured production plan specifying which tasks should be executed on which machines, at what times, and in what sequence to achieve optimal workflow efficiency. However, CP alone does not scale well for workforce allocation due to its computational complexity, necessitating a separate approach for assigning workers.

Once the machine-task assignments are determined, the second stage employs Monte Carlo Tree Search (MCTS), a Reinforcement Learning technique, to assign workers to tasks. MCTS is well-suited for sequential decision-making problems like workforce scheduling, as it explores possible worker-task assignments through simulated rollouts, evaluating different allocation strategies before selecting the most promising one. This method accounts for worker availability, skill constraints, and workers preferences, ensuring an optimal and adaptive worker distribution. Unlike CP, which struggles with the complexity of worker allocation, MCTS dynamically explores different assignment strategies and refines them based on performance feedback, making it scalable and adaptable to changing conditions such as last-minute worker absences or priority shifts.

By integrating CP for job scheduling and MCTS for workforce allocation, the service provides a comprehensive production planning solution that efficiently coordinates machine utilization and worker assignment. This hybrid approach ensures that production schedules are both cost-effective and operationally feasible, improving productivity while maintaining flexibility in workforce management.

### 3.5.5.1 Reliability

The reliability of the Production Planning Service depends on accuracy, consistency, completeness, and timeliness, as well as the strengths and limitations of its two-stage optimization approach. Accuracy is generally high, as CP guarantees optimal or near-optimal job shop scheduling, while MCTS explores numerous worker allocation strategies to refine solutions dynamically. As MCTS does not rely on prior training and remains robust across different problem instances, making it adaptable. Consistency is strong in CP, which always produces the same result for identical inputs, whereas MCTS, as a probabilistic algorithm, may generate slightly different worker allocations between runs, though it converges toward optimal solutions over time. Completeness is achieved when all constraints - such as machine availability, worker skills, and production priorities - are properly defined; missing or inaccurate input data can reduce solution quality. Timeliness is a crucial factor, as MCTS requires sufficient computational resources to generate high-quality solutions. While it is an algorithm, meaning it can return a result

within a set time frame, at least 5 seconds of computing time is typically needed to achieve near-optimal results. Overall, the service delivers reliable and high-quality production planning, but its effectiveness depends on problem complexity and available computing resources,

### 3.5.5.2 Outlook

The integration of CP for job shop scheduling and MCTS for worker allocation has demonstrated strong optimization capabilities, balancing structured decision-making with adaptability. However, there is significant potential for further enhancement. One key area of improvement is increasing the scalability and generalizability of the Reinforcement Learning approach. To achieve this, we plan to incorporate advancements in transferable and generalizable RL techniques, enabling the system to adapt to new and unseen problem instances more effectively.

Additionally, we aim to refine our MCTS framework by integrating AI-driven innovations inspired by AlphaZero and MuZero. This involves leveraging deep learning for policy and value estimation, allowing MCTS to become more efficient and adaptive over time. By combining the backward-looking learning of RL, which extracts patterns from historical data, with the forward-looking simulations of MCTS, which optimizes future decisions, we seek to develop a unified and intelligent decision-making framework. This integration would pave the way for Neural MCTS, making the system more flexible, scalable, and applicable to a wider range of complex production planning and workforce allocation challenges.

## 3.5.6 Resource Allocation MAS-based

In the evolving landscape of the manufacturing industry, the advent of MAS has marked a paradigm shift in how resource allocation is approached and managed. A Multi-Agent-based prototype for resource allocation was developed for the DAI-DSS prototype. This is an AI service aiming to provide a viable solution to support decision-making in industrial use cases. The aim of this prototype is to consider characteristics such as worker resilience and preferences in the decision-making process of allocating workers to production lines.

The system was developed using the JADE<sup>14</sup> framework through object-oriented programming with the Java programming language, which provides a set of tools to manage and deploy autonomous agents in distributed environments. This framework is interesting for its ability to facilitate communication and interaction among industrial agents and for being one of the most widely used in industrial applications.

Two stakeholders were identified: the worker and the production line. The production line needs to have workers allocated to carry out production during work shifts. This allocation takes place according to parameters identified in the business modelling. These include the worker's availability, their medical condition related to work in certain lines, their resilience regarding physiological and psychological strains and whether they have previous experience working on specific production lines. Furthermore, the number of required workers for each line, whether the production is mandatory or not, and the remaining days for the required order are also considered in this prototype.

In terms of the explainability of the current prototype, this algorithm can be described as follows: in order to provide support in deciding the most suitable workers for each line requiring worker allocation, this algorithm takes into consideration the workers registered in the form of a pool of 159 workers (see Figure 67). The agent representing a specific line requests a certain number of workers to be allocated according to the production priority and the order due date. The agents representing the workers calculate a score for each worker based on each worker's resilience and preference for working on the specific line that requests their allocation. Currently, a weight of 65% is given to the worker's resilience, and a weight of 35% is given to the worker's preference. Based on this score,

---

<sup>14</sup> JAVA Agent DEvelopment Framework. (2023). <https://jade.tilab.com/>



the worker agents are ranked, and the fittest ones are recommended to be allocated through a negotiation process carried out by the Contract Net Protocol (see Figure 68 and Figure 69).

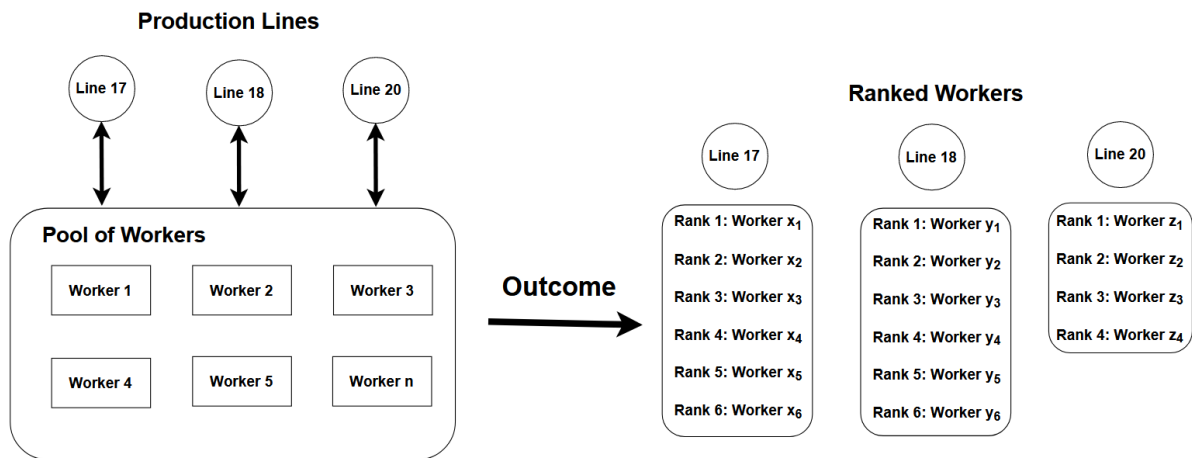


Figure 67: MAS-based Worker Allocation Dynamic

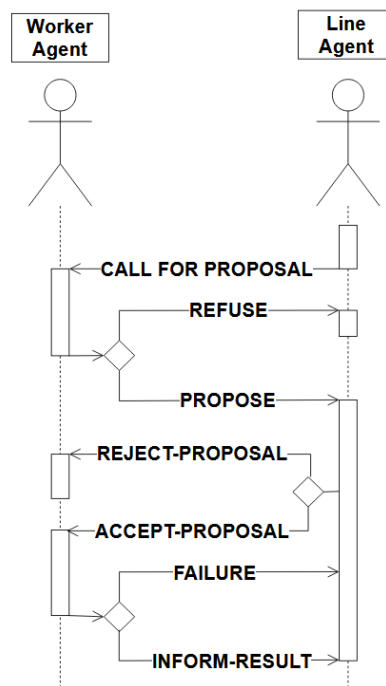
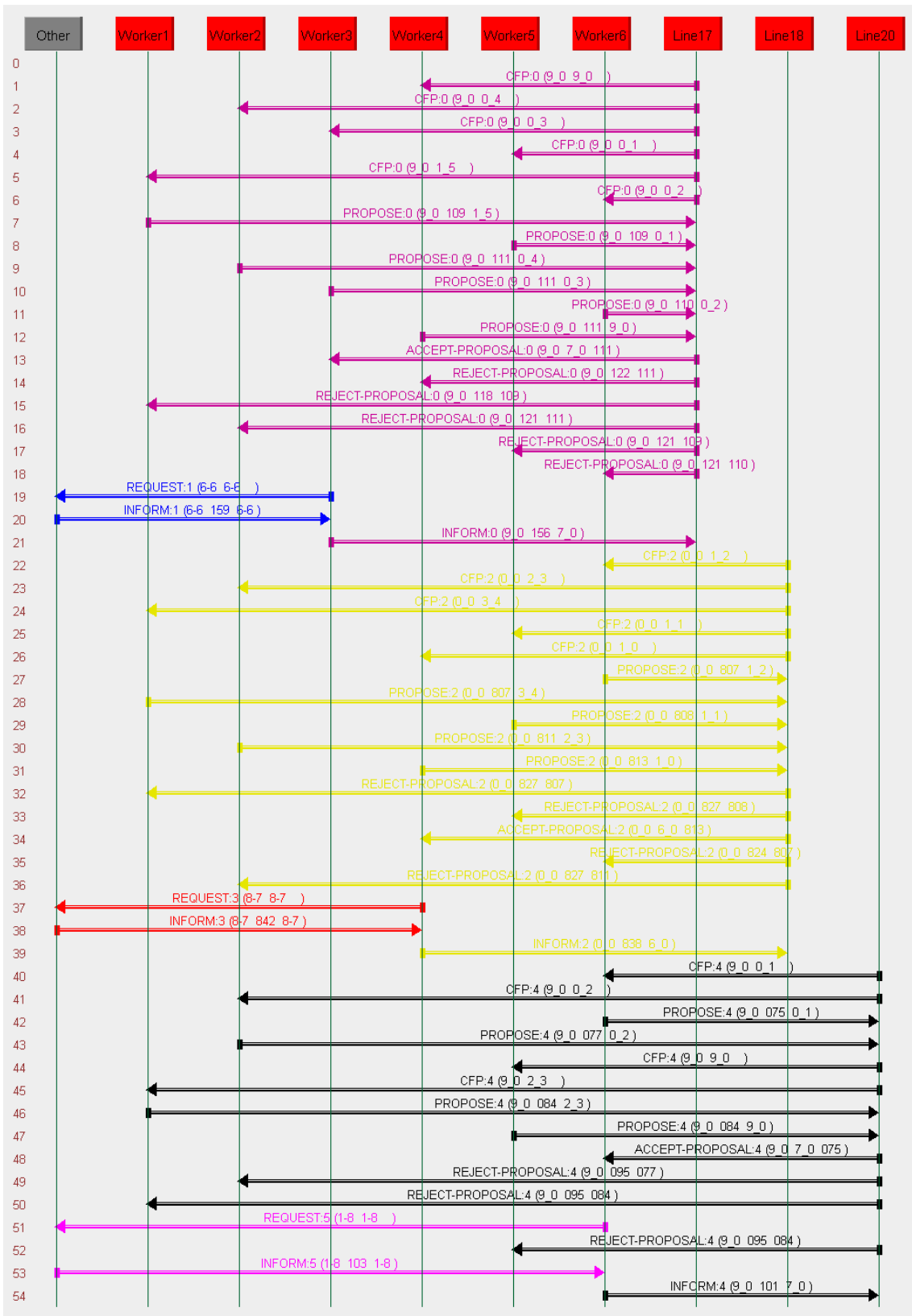


Figure 68: UML Sequence Diagram of the Multi-Agent-based Workload Balance



**Figure 69: Worker Allocation Agent Message Exchange Example**

This AI service focuses on the multiplicity, scalability and decentralisation aspects where multiple agents can be deployed in multiple production lines spread along the factory. The number of existent agents can vary over time,

and the system can deal with this dynamically, adapting to the available agents and recommending the workers that best suit the particular line.

This application of MAS for resource allocation in manufacturing not only represents an interesting approach to workforce management but also represents a form of integration between AI technologies with humans playing a core role in the task involving people and machines. By integrating AI services into decision-making processes, this prototype prioritizes factors like worker resilience and personal preferences, ensuring a more human-centric approach to industrial operations.

### **3.5.6.1 Reliability**

Reliability in MAS comprises robust communication, fault tolerance, and adaptive decision-making capabilities. The development of agent systems in closed environments is particularly advantageous for industrial applications, where reliability and consistency are paramount. Closed environments provide a controlled setting, enabling the system to function with consistency and precision. A key benefit of closed environments is the controlled agent population. In such settings, the behavior of agents is well-defined, and their interactions are carefully designed to avoid destructive competition or defection. This level of control significantly reduces uncertainties that could otherwise arise from the introduction of unverified or unknown agents. Additionally, the controlled nature of closed environments facilitates testing for industrial applications. Comprehensive testing is relevant to minimize the risk of unforeseen errors during operation, further enhancing system reliability. In industrial applications such as manufacturing, the reliability of MAS is non-negotiable.

Another relevant aspect is the standardization of agents. Predefined protocols and standards such as FIPA-ACL (Foundation for Intelligent Physical Agents – Agent Communication Language) ensure consistent communication and interoperability. Furthermore, JADE (Java Agent DEvelopment Framework) ensures reliability in MAS through its modular architecture and robust agent management features. It provides a middleware platform that adheres to the FIPA standards. It supports fault tolerance by enabling agent recovery and replication, ensuring that the system remains operational even in the event of individual agent failures. It facilitates the creation of robust MAS suitable for both research and industrial applications. With tools for monitoring agent interactions, JADE facilitates the creation of MAS suitable for both research and industrial applications.

Ultimately, the structured design of MAS developed makes them safe and compliant with industrial requirements in terms of reliability. The systematic approach ensures that MAS can deliver reliability.

### **3.5.6.2 Ethical Watchdog Agent**

The Ethical Watchdog Agent was developed to provide decision support in order to alert the decision-maker to any pre-established and previously analyzed factor in the process that is infringing on an ethical issue. Therefore, the aim of this watchdog is to sound an alarm every time the result of a worker allocation process infringes an ethical rule observed by the watchdog. Figure 70 illustrates the process in which the watchdog operates.

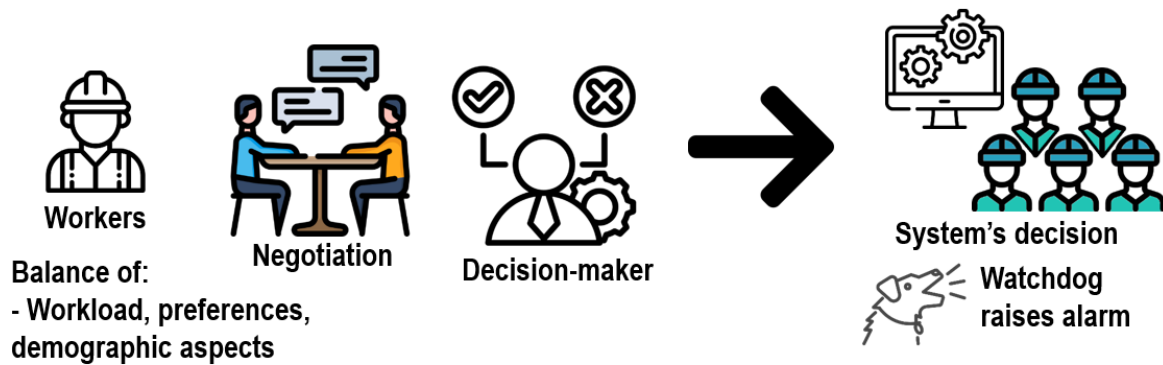


Figure 70: Watchdog Raising the Alarm

The negotiation for allocating the most suitable workers to a given production line takes place taking into account data such as the resilience to carry out the given task on the line and the worker's preference. Also, important parameters for conflict resolution were added to the newest version of the prototype. Whenever the score of a worker is assigned ties, a parameter for experience on the line and job rotation plays a role in improving the conflict situation. Job Rotation is the value that represents how repetitively the given worker has been assigned to the current task in a sequence, with the experience on the line parameter measuring how many times the worker has been assigned to that task since they began working in the factory. The result of this first round is composed by the score formed by the worker's resilience and preference with potential conflicts addressed by looking into the job rotation and experience on the line parameters. Then, after this first round, the watchdog analyzes whether any of the parameters it monitors have been trespassed. In this example, the result of the allocation was 5 workers of gender male. However, for this demographic variable, it was expected that the result would include at least 50% workers of gender female allocated to this production line. The Watchdog Agent then raises the alarm and informs the decision-maker which parameter exceeded the expected value for the given ethical issue.

Figure 71 illustrates the rest of the process and Figure 73 is an example of the watchdog message exchange in the JADE interface. The decision-maker receives the alarm sounded by the watchdog as input, and being responsible for triggering the renegotiation action, does so. Then a second round of negotiation begins in which the parameter monitored by the watchdog is observed. Finally, the system recommends a solution in which that parameter is obeyed. At the same time, the workers' resilience and preference are also observed in order to provide a result that simultaneously maintains attention to the parameters that led to the result of the first negotiation and that are desired for production.

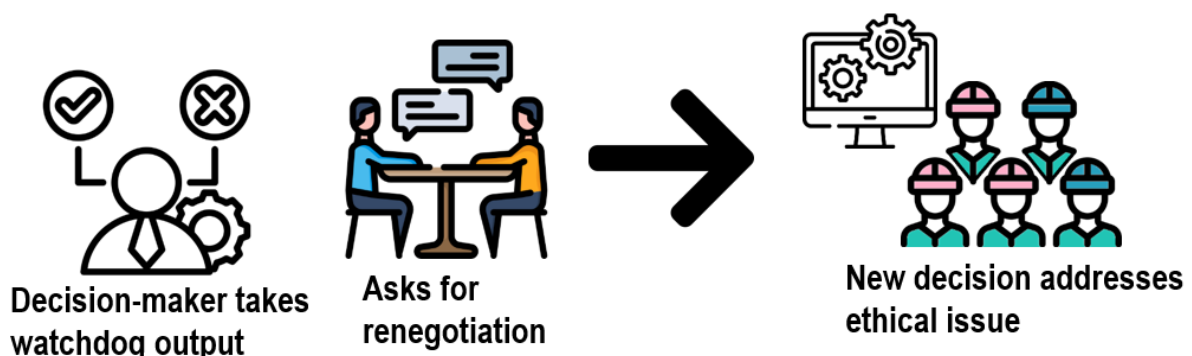


Figure 71: Renegotiation to Comply with the Watchdog

The final result shows that three workers of female gender were recommended for allocation, which is in line with the ethical standards previously established. One can see the UML sequence diagram in Figure 72.

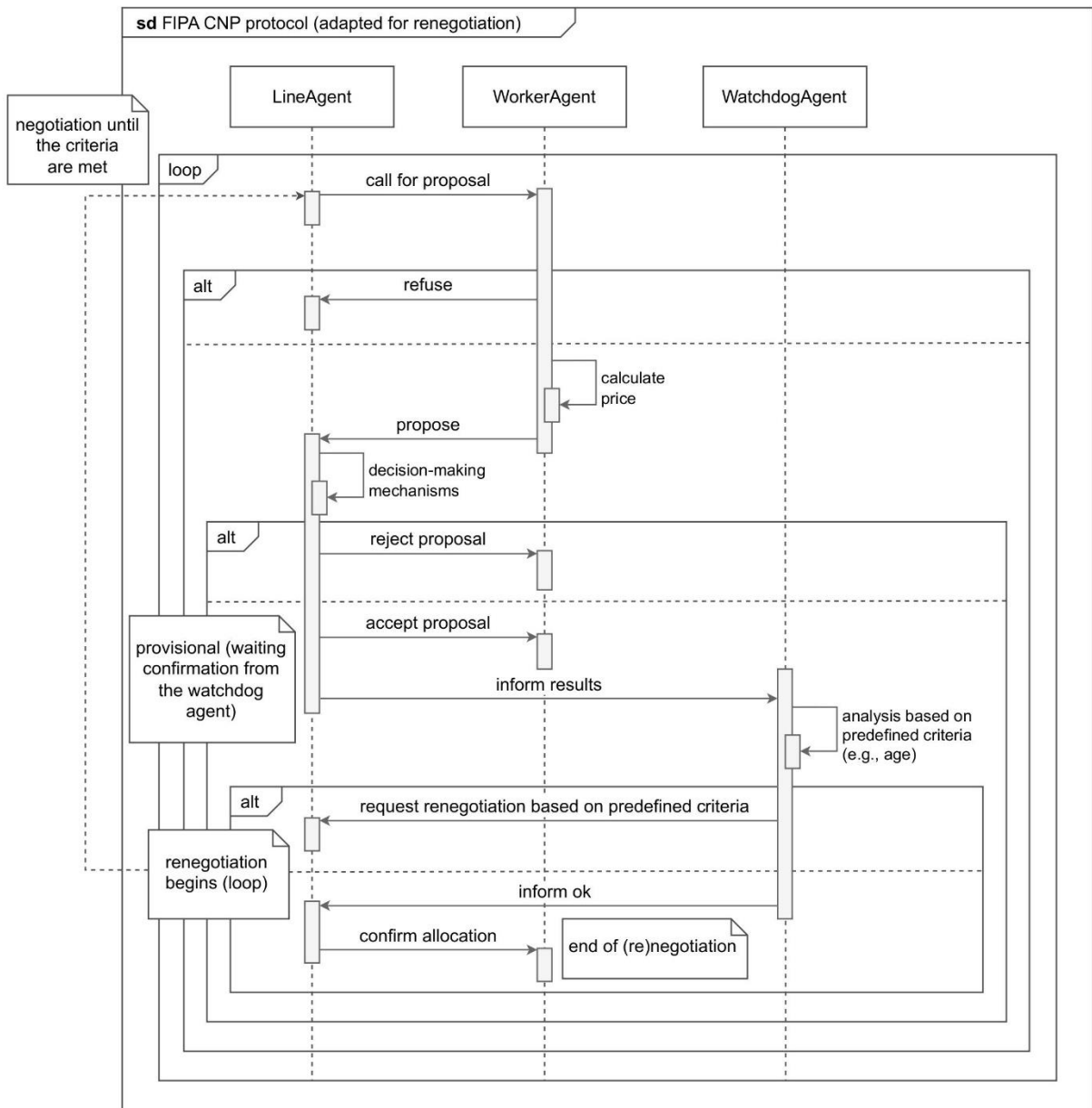
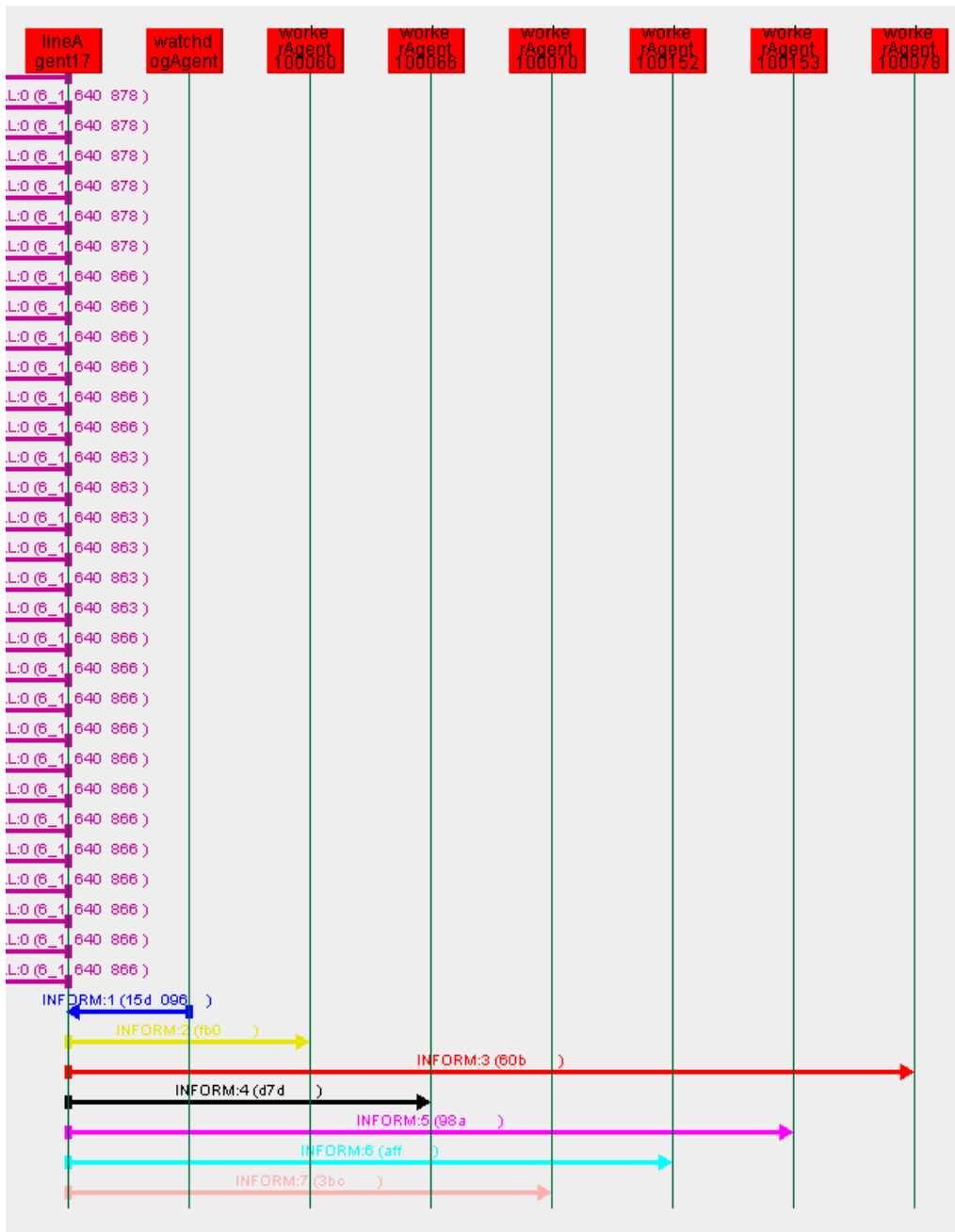


Figure 72: UML Sequence Diagram Updated for Renegotiation with the Watchdog Agent



**Figure 73: Watchdog Message Exchange after Renegotiation**

In Figure 74, one can see the result of the decision process and the WatchdogAgent in action. For Line 17, the result of the first negotiation violated the age criteria (the age average for workers must be under 30 years old) established for the ethical matter in this scenario. Therefore, the WatchdogAgent signaled that this criterion was not met. The decision-maker then has the option of asking for a renegotiation that includes the obligation to meet this criterion, or they can choose to maintain the original allocation recommendation. In this example, the decision was to trigger a renegotiation in order to choose the best workers for the job, but who also meet both ethical criteria (minimum average age and gender distribution). In the field below the first negotiation, you can find the result of the renegotiation and see that both ethical criteria have been met. The same applies to Line 18. For Line 20, no ethical criteria are met in the first negotiation, since it only includes the resilience and preference factors of all the workers. Therefore, when the decision-maker opts for renegotiation, the best available workers who jointly have an average age of less than 30 and at least half women are recommended for allocation.

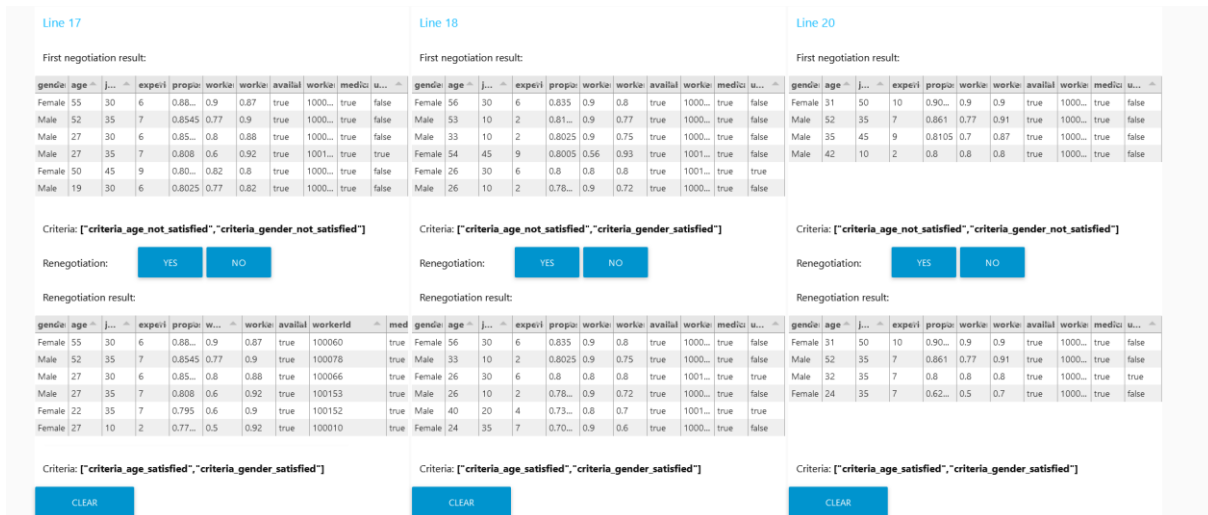


Figure 74: Interface for the Agent-based Service

In this way, an instrument is made available to support the decision maker not only with data relating to production and preference issues, such as the data relating to human resilience and preference, but also to offer an analysis of the ethical factors that result from that recommendation, always aligned with the factors of interest to be observed in each specific use case.

### 3.5.6.3 Outlook

The development of a Multi-Agent-based service prototype within the DAI-DSS framework demonstrates the significant potential of integrating human-centric considerations into industrial decision-making through a decentralized and agentic perspective. Factors such as worker resilience and preferences play a decisive role in representing human integration and representation in the decision-making process. It showcases a scalable, decentralized, and adaptable system capable of managing workforce allocation to production lines highlighting a human-centric resource allocation, and contributing to future advancements in decision-support systems in industrial contexts.

Looking ahead, the integration of MAS with LLMs in resource allocation offers an exciting frontier for MAS-based applications. LLMs can enhance MAS capabilities by providing advanced natural language understanding, enabling richer and more intuitive interactions between agents and stakeholders. This integration will help the incorporation of complex, context-aware human inputs and further refine decision-making processes in an agentic approach. This agentification allows for more efficient communication through the division of tasks in processing the demands of LLMs. It is an advance towards more autonomous, intelligent, and human-aligned industrial systems that further embody the principles of Industry 5.0.

## 3.5.7 Truck Loading Service

The truck loading service is an experiment that aims to optimize container loading on trucks in terms of filling the volume of the existing trucks of different sizes and therefore, to use the least amount of trucks and assist in decisions regarding these containers' shipment. This service aims to address the shipping process by streamlining the decision-making steps involved in container placement and truck selection. The correlating container dimensions with truck capacities, the service devises loading strategies that maximize space utilization.

The truck loading service takes the shipping plan and the relevant logistical data as inputs to determine the optimal container placement and truck allocation. It begins by evaluating each shipment's due date and the specific requirements of the contained products. Once the shipment is deemed ready, containers are loaded into trucks, and the system analyzes truck saturation to identify any free capacity. If a truck is underutilized, alternative truck

options are considered, or additional product geometries may be incorporated to better fill the available space. In cases where neither adjustment achieves the desired efficiency, the service flags the shipment for further review, potentially prompting communication with the customer to consider schedule adjustments. This decision-making process considers container compatibility, truck capacity, and dynamic shipping constraints to optimize resource allocation.

At its core, the truck loading problem can be modeled as a variant of the classic bin packing problem. In this context, containers, each with distinct dimensions and loading requirements, are analogous to items that must be packed into trucks, which represent bins of fixed capacity. The primary objective is to minimize the number of trucks used while accommodating the varying sizes and geometries of the containers. Given that the bin packing problem is NP-hard, achieving an optimal solution in every case is computationally challenging. Nonetheless, the field of bin packing offers valuable insights that can be adapted to enhance truck loading efficiency.

Heuristic and approximation methods provide practical strategies for addressing the truck loading challenge. Sorting containers based on size or other relevant attributes and then sequentially placing each container into the first available truck that can accommodate it strike a balance between computational efficiency and effective space utilization. Although these methods do not always guarantee an optimal solution, they have proven effective in managing shipping demands and dynamic loading scenarios.

#### **3.5.7.1 Reliability**

The truck loading service is reliable because it follows a systematic approach. This means it matches container dimensions with truck capacities. This ensures consistent optimisation of space utilisation and minimises the risk of underutilised vehicles. The service also has the additional capability of flagging shipments that fall outside of optimal efficiency parameters. It provides reliable feedback and allows for proactive intervention. This ensures that potential issues are identified and addressed before impacting delivery schedules.

#### **3.5.7.2 Outlook**

Currently, the truck loading service is designed to optimize container loading across a fleet of trucks with varying capacities, with a focus on minimizing the number of trucks required while maximizing shipment efficiency. Its modular design allows for easy adaptation to different shipping environments and logistical challenges. In future iterations, the system could integrate additional variables such as traffic conditions, weather patterns, and driver schedules to further refine its loading strategies. As data quality and algorithmic sophistication continue to improve, the service is expected to become an integral part of comprehensive shipping and logistics management, enhancing operational efficiency and customer satisfaction across the board.

### **3.5.8 Support Machine Maintenance using RAG and LLM**

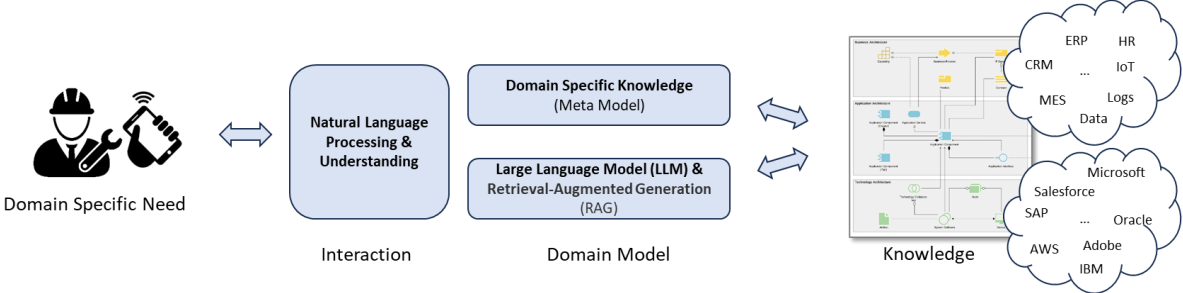
The idea is to use a chatbot to support the maintenance of machines via natural language interaction. To enable maintenance workers to query information about machine maintenance by using natural language for instance on a mobile device providing audio/voice responses, the following four aspects are considered relevant. First, structured information can be collected via ADOIT<sup>15</sup> used for modelling IT infrastructure for example the different databases for maintenance information as well as instances of (enterprise) architecture-relevant concepts. Second, not only structured information, but also unstructured information such as the maintenance history or relevant notes stored in documents containing text and images, must be handled. Here, LLMs are used to interpret these documents, identify patterns, establish a structure and get chunks of the domain or use case-specific information in the LLM. Third, natural language querying is expected to improve the interaction of the maintenance workers with the AI by using edge devices such as mobile phones to ask queries on the status of specific machines. Here,

---

<sup>15</sup> <https://www.boc-group.com/en/adoit/> (accessed 03.02.2025)

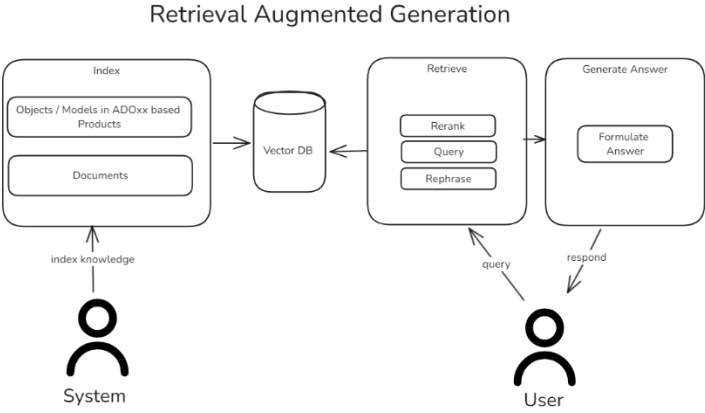


the prototype processes these queries and returns the machine maintenance status information. Fourth, via a mobile or desktop, it is possible to ask *queries via voice or text* and also to receive the response in audio or written format. The LLM accesses various information sources to deliver meaningful answers. When using the desktop version, the answers are longer, contain links to the corresponding documents, and include figures, while the mobile version provides shorter answers. The overall concept is depicted in Figure 75.



**Figure 75: Concept for Machine Maintenance**

The prototype uses the principle of RAG patterns to improve the quality of the LLM’s response by including external knowledge stored and retrieved from vector databases. The knowledge about maintenance is stored in 1) documents and manuals and potentially includes information stored in models such as definitions of the data sources, their architectural and infrastructure aspects, and 2) what is available on the Internet. Therefore, RAG is used to forward the LLM to retrieve the relevant information from predefined knowledge sources not used for its training. In this case, it guides the LLM to use in addition to its trained knowledge, the information of the FLEX maintenance documents so that answers are based on the more precise domain knowledge.



**Figure 76: RAG Concept for External Knowledge**

Two major building blocks are relevant to creating the machine maintenance prototype, which are the application configuration and the wiring. The configuration covers all relevant information and definitions such as client components or client functions and defines the wiring to handle endpoints, connectors, and flows. The wiring manages the steps that are executed to transform the uploaded file into suitable input for the vector database model by bringing together the following endpoints:

- **doc-extractor:** The endpoint can interpret different document types (doc, docx, pdf, txt, ...) and extract textual/image content.
- **semantic-splitter-chunking:** The endpoint splits the extracted text into smaller chunks adhering to a certain length and semantics.
- **excel-chunking:** The endpoint creates a chunk for each Excel sheet row.

- **image-to-text:** The endpoint extracts text from an image.
- **transform-image-desc-to-text:** The endpoint transforms the extracted image description from “image-to-text” into chunks.
- **transform-chunks-for-queue:** The endpoint transforms the created chunks (e.g. from Excel rows) into the format that is required to execute the request-queueer service.
- **request-queueer-sender:** The endpoint sends the formatted chunks to a queue service that processes and stores the chunks in the vector database.
- **vector-service-search-maintenance:** The endpoint uses the capabilities of a "vector-service-search" lambda function and returns a valid user response based on the "maintenance" index.

For chunking file content to be uploaded in the vector database there are two options, either the input file is an Excel or another file type. In case the file that was extracted by the “doc-extractor” is an Excel, chunks for each row are created using “excel-chunking”. The pieces are prepared by transforming the chunks via “transform-chunk-for-queue” and sent to the queue service with “request-queueer-sender” to be stored in the vector database. In case the extracted file was a PDF or another file type, the chunks are created using “semantic-splitter-chunking”. The pieces are prepared by transforming the chunks via “transform-chunk-for-queue” and sent to the queue service with “request-queueer-sender” to be stored in the vector database. In parallel to the chunk splitting, each picture that was extracted by the “doc-extractor” has to be transformed as well. For this, the text from a picture is extracted via “image-to-text” and afterwards the image text is transformed into appropriate chunks using “transform-chunks-for-queue”. As with all other chunks, they are sent to the queue and the database as a final step.

The sample flow of uploading the data (chunks) into the vector database (used by the LLM) is as follows: (1) uploading the documents in the prototype interface in docx, pdf or xlsx format, (2) “submit” the upload, and (3) check the number of chunks that have been created and that were stored in the vector database.

The retrieval of information is based on the RAG concept using the Agentic Knowledge Retrieval approach with LLM-based agents. For this, an agent pattern is created in LangChain that aims to fulfill the goal of answering maintenance questions from the user. Usually, an agent does this by defining sub-goals, decomposition, reflection, and refinement, however, in this use case the goal is known and a fixed pattern for how to solve the problem can be provided. The agent can access its short-term memory enabling in-context learning and being aware of the interactions that happened before. This is important for the maintenance use case as dialogue can occur between the agent and the user asking questions. Also, the agent could access its long-term memory which is composed of information that is available to the agent stored over extended periods, in most cases in the form of vector stores. However, in this prototype, the maintenance vector store and its querying are seen rather as tool usage. Tools allow the agent to call external APIs to solve the goal. This can be for example a search on the internet. In the case of maintenance use case querying the maintenance database, ADOIT, the internet or the knowledge of an LLM could be done with the agent since external APIs are invoked.

As an LLM, the general purpose LLM model "GPT-4o" from OpenAI is used to differentiate between tools by using an agent approach and to initialize data into the vector database. Different LLMs may be configured to receive better results for targeted requests. The OLIVE framework connects the individual endpoints ranging from the uploading of the document to the model creation. It is used to improve the prompts and select suitable LLM models for individual endpoints. Also, OLIVE supports the creation of an application that can be considered a UI.

### 3.5.8.1 Reliability

For enhancing AI reliability the proposed model-based framework of D3.3, on the one hand, the idea of using workflows and conceptual models as AI steering technologies to LLMs is applied, on the other hand, the usage of RAG targets to increase factual accuracy through reducing hallucinations (Figure 77). Each time a question is submitted by the user, an agent pattern defines which tool to use based on the user input. The different tools have

a description of the capabilities and an overview of the stored data elements. Different tools can be configured in the agent config which is defined in the “agent-maintenance-showcase” step under “toolEndpoints”. Additionally, the agent is guided with a specific prompt applying prompting techniques to enhance it with context information to the different data sources. In the maintenance case, the agent is accessing information based on the RAG principle and accesses the “vector-service-search-maintenance” where the domain-specific documents of FLEX maintenance are stored. To enhance the reliability of the provided information and maintenance instructions, the prototype includes links to the source documents in the UI where the provided information by the LLM-based Chatbot can be double-checked.

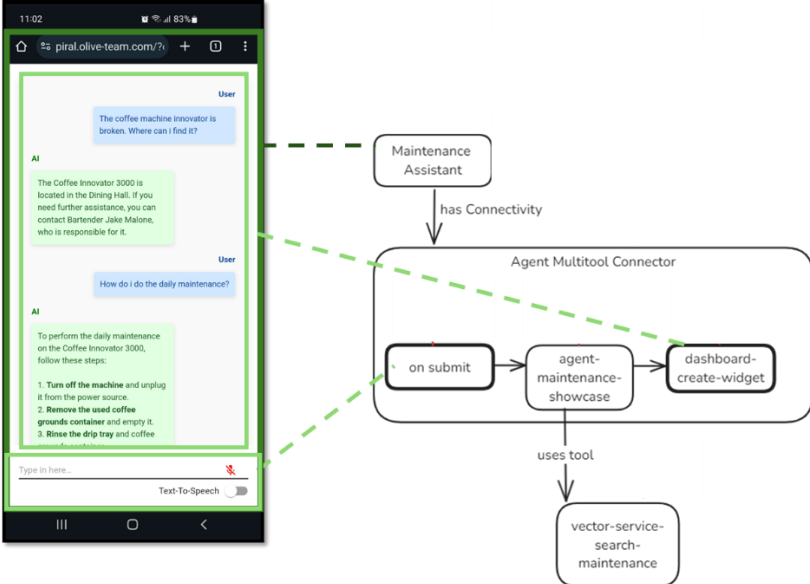


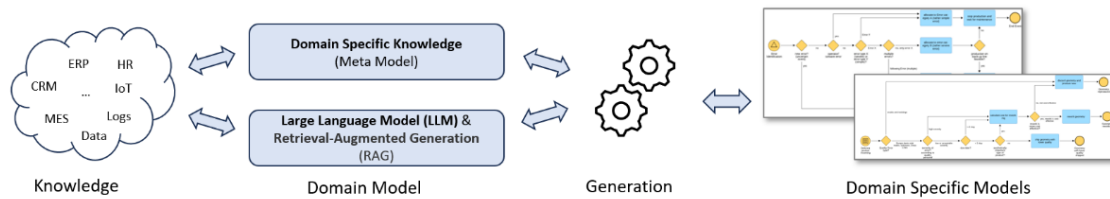
Figure 77: Workflow for Machine Maintenance Prototype

3.5.8.2 Outlook

Currently, the prototype provides correct answers and references the right documents stored in the vector database. Therefore, the AI scope is limited to certain very specific use cases, where RAG patterns are used to improve the quality of the LLM response including the usage of external sources of knowledge. However, the questions must be very precise so that the chatbot gives back the right answer. Here advancements could include fields in the UI that can better guide the prototype to the answers. Additionally, a general improvement in the natural language understanding should be made. It provides good answers for English or German input but with background noise or dialect the chatbot faces difficulties.

3.5.9 Document Transformation using LLM

AI – specifically LLMs – can be applied to convert unstructured documents into structured business process models (BPMN). AI support during the transformation is of high interest e.g. FLEX who has already a lot of instructions documented but in unstructured ways to reduce time and manual effort. In the conversion process, the role of AI is three-fold. First, the content of existing documents is interpreted by using large LLMs to understand the concepts and identify relationships. Here, LLMs may consider domain-specific knowledge to interpret the document context. Second, AI is used to handle different media types in documents such as plain text, titles, section headings, numbers, listings, images and the like. Third, LLMs can be used for lay-outing and positioning of the elements in the BPMN model. The output of the text-to-model AI services consists of structured process models describing the document content. Human experts are still needed to check the AI-generated processes and finally approve them. The conceptual overview is illustrated in Figure 78.



**Figure 78: Document Transformation Concept**

The models are created with the same process modelling tool ADONIS that was already used for creating the decision processes of FLEX (described in FAIRWork D5.1). As an LLM, the general purpose LLM model "GPT-4o" from OpenAI is used to interpret the document, create a model JSON file out of it, and lay out the model. Different LLMs may be configured to receive better results for targeted conversion (e.g. requiring specific domain knowledge, etc.) requests. The OLIVE framework connects the individual endpoints ranging from the uploading of the document to the model creation and is used to improve the prompts and select suitable LLM models for individual endpoints. Also, OLIVE supports the creation of an application that can be considered a UI.

As for the machine maintenance prototype, the application configuration and wiring are again two major building blocks. The configuration covers all relevant information and definitions such as client components or client functions and defines the wiring to handle endpoints, connectors, and flows. The wiring manages the steps that are executed to transform the uploaded .docx file into an ADONIS model by bringing together the following endpoints:

- **doc-extractor:** The endpoint can interpret different document types (doc, docx, pdf, txt, ...) and extract textual/image content.
- **transform-text-prompt, output-to-prompt and model-json-to-doc-list:** These endpoints transform JSON data so that the output of a node fits the input for the next node.
- **process-text-to-model-json:** The endpoint interprets the text to identify nodes and edges of the model and creates a JSON out of it using the LLM "GPT-4o" provided by "OpenAI".
- **layout-model-prompt:** The endpoint adds position information to the nodes to provide a layout for the model by using the LLM "GPT-4o" provided by "OpenAI".
- **insert-image-to-model:** The microservice endpoint takes the image URLs from the doc-extractor and adds to the model for each image a note element (C\_NOTE) depicting the image.
- **create-model-adoxx:** The microservice endpoint communicates with a REST interface of an OLIVE extension allowing the creation of a model.

The sample flow of document conversion is as follows: (1) uploading the documents in the prototype interface in docx, pdf or xlsx format with "submit", (2) a link to the generated model in ADONIS will pop up, (3) the model contains the process tasks extracted from the uploaded document plus images as note objects, and (4) document details are included in the relevant process tasks. An example output for the FLEX document transformation is shown in Figure 79.

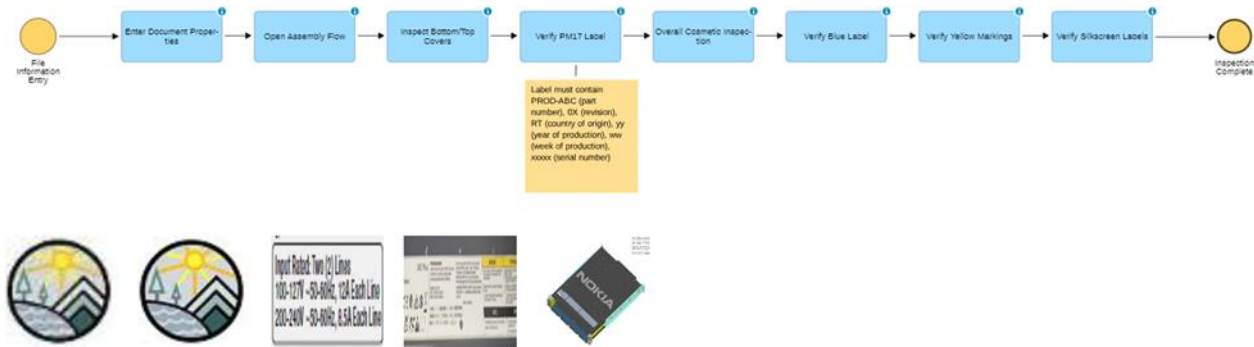


Figure 79: Sample Output of Document Transformation

### 3.5.9.1 Reliability

For enhancing AI reliability the proposed model-based concept of FAIRWork D3.3 and the idea of using workflows and models as AI steering technologies is applied. Figure 80 illustrates this for the FLEX prototype “document to the business process model”.

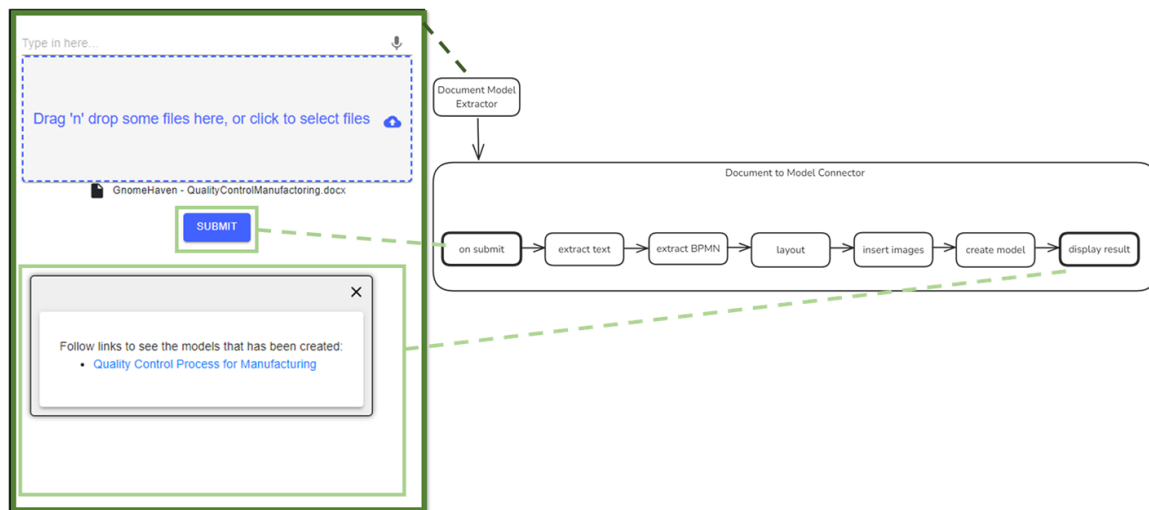


Figure 80: Model-based workflow for Document Transformation Prototype

Each time a document is uploaded the LLMs come to different results, hence, the workflow-based principle aims to guide, create transparency, and support reliability. For each of the steps, certain prompts and requirements are defined that must be considered or complied with. For, example, when uploading the document and pressing the “submit” button, different components and services of the application are triggered. In the component “extract text” a service is used to interpret different types of documents like docx, pdf, txt etc., and extracts the textual content and the contained images. Under “extract BPMN” a prompt for interpreting text and generating a JSON supported by the LLM is used to analyze the data and create the corresponding model. Additionally, the “layout” step includes an engineered system prompt to provide concrete information to the LLM on laying out the BPMN model. The prompt includes a generic description of the target diagram and BPMN foundations. E.g. information that it contains nodes and edges, the number of pixels needed for the nodes, the flow direction and positioning, or some information on the JSON schema. Then there are multiple other steps defined in the workflow to guide the AI to the final result. Additionally, multiple prompting techniques are used to specify the required output. These include e.g. Instruction-Based Prompting, Structured Output Prompting, Contextual Prompting, Conditional Logic in Prompting, and Summarization Prompting

### 3.5.9.2 Outlook

Two aspects need to be considered for this prototype. First, the result may differ each time the service is executed although the input document is the same. Generative AI is not deterministic, since the AI is trained on huge data sets serving as an input for the requested answer. The AI reacts flexibly and creatively when choosing the relevant data to produce an answer resulting in different answers. From a more technical perspective, generative AI is not deterministic, as it is based on probabilistic models. It uses probabilities (learned from the huge training data sets) to run through neuronal networks to forecast the solution evolution. Thus, although the LLM can help with analyzing and structuring information, the variability must be taken into account otherwise it is not feasible to use it.

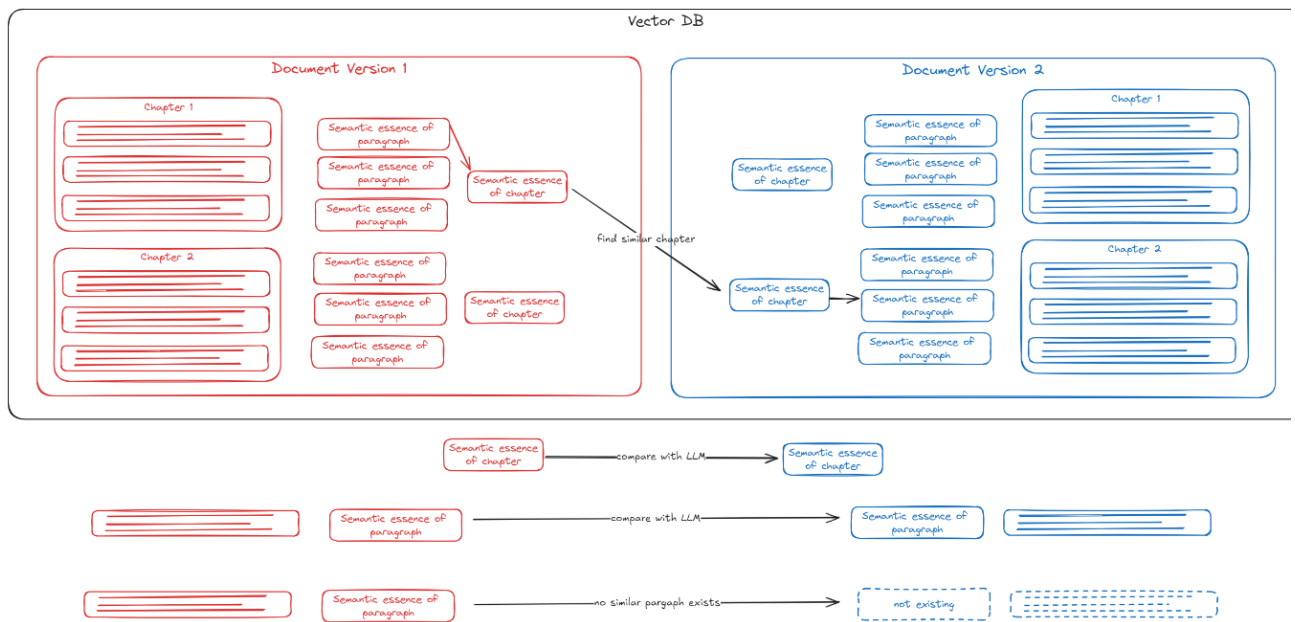
Second, depending on the document content, data privacy concerns may arise. At the moment, services from OpenAI (a third-party AI provider) are applied, which could require compliance checks to ensure that users' critical production documents are kept confidential. A workaround for these privacy concerns is to host the LLM model within the use case company or a trusted partner, still, external providers (e.g. AWS, Azure, etc.) for the hosting infrastructure may be needed.

### 3.5.10 Support Compliance for Clean Room using RAG and LLM

For compliance for FLEX clean rooms, the customer wants to compare entirely different documents or different versions of one document to support the employees in having an overview of the latest relevant compliance regulations. For example, periodically, new versions of a regulation become effective in the European Union or one of its member states and to get a rough overview of the changes and if it is relevant for which country, an AI solution that provides a rough summary of the changes without having to go through the entire full document. Thus, AI can support gaining an initial overview and provide inputs of what is relevant and applicable to a company.

The rough general concept for “compliance document comparison for Clean Rooms” covers four main steps and applies the RAG concept. First, LLMs, “GPT4-o”, are used to chunk the document into smaller pieces and store information about the contents of those pieces in a vector database. For this step, the uploaded documents are split into chapters which are again separated into paragraphs. Then for each paragraph, a semantic summary is created using the LLM. All paragraph summaries are utilized to create an overall chapter summary and the paragraph as well as the chapter summaries are stored in the vector database. The second step targets the search for similar contents between the two documents by using the vector database. For the comparison, each chapter of the document is analyzed to find a semantically similar chapter in the other document. If a resembling chapter is identified, in the third step, the smaller document chunks, the paragraphs, are searched and compared for similar information with the help of LLM. Additionally, any paragraph not contained in the other document is remembered. In the last step, based on the complete list of all found document differences between chapters and paragraphs, the LLM is used to categorize them and create an overall summary. The overall process is depicted in Figure 81.





**Figure 81: Document Comparison Concept using Vectors**

The execution of the AI solution is triggered by entering the document names in the corresponding lines of the UI and pressing the submit button. Then depending on the document size it usually takes about two minutes to complete the comparison check and a summary of all differences is provided in the output. For large documents, it can take up to 15 minutes. A benefit of the applied concept of extracting the semantic context of various documents and transforming it into vectors is that it is also possible to compare the context to other forms of data like repository data.

### 3.5.10.1 Reliability

For the compliance document comparison application, a workflow that includes the integration of the vector database and RAG, as well as for some steps of sequential LLMs usage to generate the outputs. The LLM is guided via precise prompts to receive the desired output and enhance their reliability. The prompts include the description of the received input format of the previous step of the workflow, a detailed description of its task and the specification of the output format JSON. A combination of multiple prompting techniques are used and include Instruction-Based Prompting, Structured Output Prompting, Contextual Prompting, Conditional Logic in Prompting and Summarization Prompting.

### 3.5.10.2 Outlook

The document comparison works best with similar documents and document structure. The higher the variability of the documents to compare, the harder it is to sum up all the important differences. For example, the comparison with LLMs of document versions with similar structures performs better in detecting differences, than entirely different documents or ones with completely different structures such as comparing information in text form with information in table form.

Potential advancements of this AI application include an improvement in the parsing of tables. This will lead to general improvements in detecting differences, as a high ratio of the FLEX compliance documents describes the compliance standards in this way. Secondly, the handling of subchapters can be improved. The idea is to parse subchapters instead of root chapters to enable the detection of similar chapters across different document structures.

### 3.5.11 Calibration Certification Service

After a device is manufactured, it must undergo validation to ensure its accuracy and reliability. Calibration issuers are responsible for testing the device and issuing a certificate containing all the essential details about the tested device and the calibration test cases. FLEX processes approximately 3,000 such certificates annually. Currently, these certificates are manually reviewed for formal errors and test results, a process that can take upwards of 10 minutes per certificate depending on the document's length. This time-intensive task involves data comparisons from various sources to ensure that only certificates with accurate information are deemed valid. By enhancing the efficiency of this process, the certificate validator significantly streamlines FLEX's workflow.

The format of a calibration certificate varies depending on the issuing laboratory. Presently, the majority of certificates FLEX receives are issued by the laboratories MicroPrecision and TESI. Despite differences in layout formatting, the core information contained in the certificates remains consistent and typically includes:

- Details about the device undergoing testing

```

MPC Control #: 3062
Asset ID: 3062 URS 6.0
Gage Type: MULTIMETER URS 2.0
Manufacturer: HEWLETT PACKARD URS 3.0
Model Number: 34401A URS 4.0
Size: N/A
Temp/RH: 21.1°C / 48.9 %
Location: Calibration performed at Customer's facility
    
```

- Information on the calibration equipment used

The manufacturer procedure was applied (Document PN 34401-90013, Ch.4). Standards used to calibrate inspection, measuring and test equipment are traceable to national and international standards. The following reference standards have been used for calibration:

Asset Number	Serial Number	Certificate	Due date
TES1288	3674901	374498	2024-03-09
TES0494	2823A21136	M737_2023_ACCR_EO	2024-03-16

- Confirmation that all calibration tests have been passed

REFERENCE INSTRUMENT	INSTRUMENT UNDER TEST				RESULT
	APPLIED VALUE	LOWER LIMIT	READING	UPPER LIMIT	
	(N)	(N)	(N)	(N)	
	20,84	18,8	21,1	22,9	pass
	30,44	27,4	30,8	33,5	pass
	40,72	36,6	40,9	44,8	pass
	50,50	45,5	50,7	55,6	pass
	60,41	54,4	61,1	66,5	pass
	70,36	63,3	71,4	77,4	pass
	81,21	73,1	81,7	89,3	pass

- Signatures of authorized personnel
- Verification of certificate completeness



Key details about both the tested and calibration devices are accessible through the database.

Key Components:

- **Configuration File:** Allows the end user to modify paths for data, configuration, and output folders.
- **Layout Configurator:** Provides layout structure information for individual laboratories.
- **PDF Parser:** Extracts tables, plain text, and images from the provided PDF. Cleans the data by removing unnecessary whitespaces, adjusting capitalization, and refining table cells. Utilizes two Python libraries, including Fitz.
- **Validator:** Compares the extracted information against the guidelines defined by FLEX.
- **Viewer:** Generates a user-friendly overview of certificates that do not comply with the guidelines.

Currently, the system is deployed as an executable file (EXE) that runs locally. Users simply place the documents to be validated into the folder specified in the *document\_path* configuration and start the EXE. The service processes each document sequentially, scanning for tables, keywords, and images. Validation results are stored in log files, and upon completion, a webpage summarizing all failed documents is automatically displayed, which can be seen in Figure 18.

### 3.5.11.1 Reliability

The service utilizes a rule-based machine learning approach, which has been extensively tested on over 1,000 TESI-issued documents. As a result, it offers a high level of reliability in extracting and validating information. In cases where the service is unable to process a document, it will be automatically flagged, and an error message will prompt the user to review the document manually. This ensures that no document goes unchecked and maintains the accuracy of the validation process.

### 3.5.11.2 Outlook

Table and text extraction capabilities will be further enhanced through the integration of additional test data, allowing for more accurate and reliable results. Additionally, a web-based interface will be developed to improve accessibility and usability, enabling users to interact with the service more intuitively. Future updates may also include support for a wider range of document layouts, ensuring greater flexibility and compatibility with diverse document structures.

## 3.6 Real World Data Providers

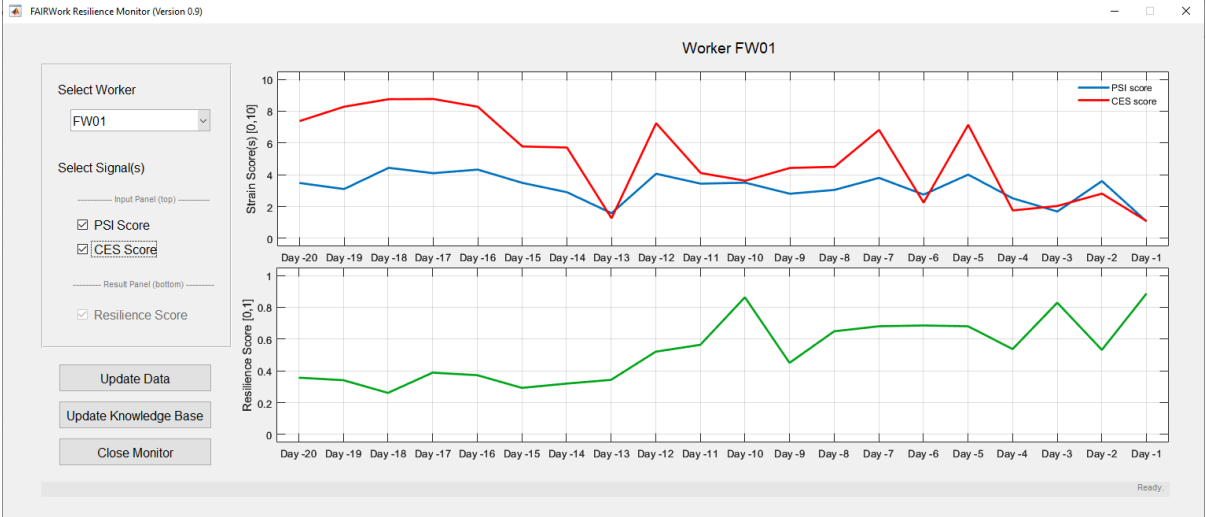
### 3.6.1 Intelligent Sensor Boxes

The Intelligent Sensor Box (ISB) as real world data provider enables the measurement of a worker's physiological and psychological stress while doing tasks and finally provides information about the worker's estimated resilience. In general, it is configured based on a set of wearable and also stationary sensors together with AI-based analytics for assessment and optimization functions. Its application provides a means to assess the human factors as well as the ergonomics of industrial training and work environments.

The ISB component of the DAI-DSS system architecture primarily collects daily bio-signals from the wearables that are worn by the workers and connected to the Local Workplace Sensor Network records. A local decision support system generates the first analytics on the worker's human factors from the Human Factors Intelligent Services that provides state-of-the-art in terms of relevant metadata via a prepared UI to the worker. The ISB also enables the research assistant to view the worker's biosignals, human factors parameters and current system state via an expert's external dashboard. The DAI-DSS Knowledge Base finally has access to necessary data from the ISB via the dedicated REST API. To ensure data protection, all provided data is anonymized or at least pseudonymized by

the Data Anonymization component. The psychophysiological strain of the worker accumulates when (job) demands, such as time pressure or physical workload are appraised as a threat due to inefficient available resources to adaptively cope with them. Details of related methods are described in FAIRWork D3.2 and D3.3.

For the computation of the resilience score, in the first stage, the Daily Strain Score (DSC) is calculated. This score integrates contributions from the Physiological Strain Index PSI\* as well as from the Cognitive-Emotional Stress (CES) score into DSC(n) of day n, squashed by the Sigmoid function to always be within the interval [0,1]. In a further step, the DSC(n) components of a pre-defined time window – for example, 20 working days – are integrated in a further expression that downscales more distant DSC(n) with an exponential decay function with  $\tau$  being another time window for having larger weights for the recent 10 working days. A weighted and normalized sum would then represent an equivalent of a score for potential aspects of mental exhaustion, or, the need for recovery (i.e., NFR). Finally, the overall current resilience score RS is computed. Figure 82 provides a sample number of estimations of physiological and cognitive-emotional strain during a time course of 20 days of a potential worker together with the integrated resilience score RS. The most recent estimate is uploaded to the DAI-DSS Knowledge Base for further consideration.



**Figure 82: FAIRWork Resilience Monitor, with a Sample Number of Estimations of Physiological (blue) and Cognitive-emotional (red) Strain during a Time Course of 20 Days of a Potential Worker**

This resilient score provides then the data for further computational objectives, see section 3.4 or further rationale.

**3.6.1.1 Reliability**

Explainable AI and fairness of AI services in the context of socio-technical environments are key to enabling future, ethically approved applications of AI for the optimization of production services with human-machine interaction. Fair algorithms will prevent decisions from reflecting discriminatory behavior. The aim is to gain a better understanding of collective decision-making processes to tackle new socio-technological challenges where aspects of decision-making and fairness are important. We need to ensure that people in similar situations are treated equally and not discriminated against. Examples of unfair decisions are situations where people are discriminated against on the basis of protected characteristics, such as race, gender, or age.

The computation of the resilience score is based on a computation of the Physiological Strain Index which in turn is based on an AI-driven estimate of the core body temperature. FAIRWork D3.2 (section 3.3.2; AI-based Physiological Strain Estimation) describes in detail the methodology of this integrated AI-based service. In general, multiple models were proposed for the estimation, several of them being represented by regression trees that implicitly provide full transparency to the human expert. The best of these models was the model with bagged trees.

Bagging trees is an ensemble technique that consists of combining several regression trees fitted on different bootstrap samples of the training set. The main advantage of regression trees is their human-readability. Regression trees not only predict attribute values of targets, but they also explain which attributes are used and how the attributes are used to reach the predictions.

### **3.6.1.2 Outlook**

A further component of the resilience scoring will be integrated on the basis of the Recovery-Stress state (RSS) of the worker. A measure of the current RSS includes the frequency and amplitude of stress symptoms as well as the amplitude and frequency of recovery-associated activities. Furthermore, the dimension of bouncing-back effects is measured based on the concrete responses to the stress symptom data.

Future extensions of this work-in-progress are planned to focus on (i) gaining wearable data from the work at the manufacturing company, analyzing and evaluating it from the point-of-view of the resilience framework, (ii) introducing fairness and transparency in methodologies, applying visualization methodologies for intuitive insight into the analytics' behavior, and (iii) applying optimization methodology on the data in order to provide best economical solutions from a long-term monitoring and assessment perspective.

### 3.7 Summary of the DAI-DSS Building Blocks

This section aims to provide a comprehensive overview of section 3 summarising all prototypes and building blocks of the DAI-DSS. The below Table 3 provides a summary of the building blocks along with their corresponding implemented services and prototypes. This aims to enhance the understanding of the main picture of the FAIRWork DAI-DSS.

DAI-DSS Building Blocks	Implemented Services and Prototypes
<b>DAI-DSS Configurator</b>	<ul style="list-style-type: none"> <li>• Multi-Agent Orchestrator Configuration</li> <li>• Configuration Framework</li> <li>• Configuration Integration Framework</li> </ul>
<b>DAI-DSS User Interface</b>	<ul style="list-style-type: none"> <li>• Order Overview UI Component</li> <li>• Worker Overview UI Component</li> <li>• Allocation Proposal Through AI Resource Allocation Service UI Component</li> <li>• Production Planning Service UI Component</li> <li>• Truck Loading UI Component</li> <li>• Document Transformation UI Component</li> <li>• Machine Maintenance UI Components</li> <li>• Documents Compliance UI Components</li> <li>• Calibration Document Validation UI Component</li> </ul>
<b>DAI-DSS Orchestrator</b>	<ul style="list-style-type: none"> <li>• Workflow-based Orchestration</li> <li>• Multi-Agent-Based Orchestration</li> </ul>
<b>DAI-DSS Knowledge Base</b>	<ul style="list-style-type: none"> <li>• Knowledge Base: ISO 10103-Based Data Repository</li> <li>• Intelligent Sensor Boxes</li> </ul>
<b>DAI-DSS AI Enrichment</b>	<ul style="list-style-type: none"> <li>• Support Understanding of Decisions through Conceptual Modelling</li> <li>• Decision Support through Decision Tree</li> <li>• Resource Allocation using Neural Networks</li> <li>• Resource Allocation using Linear Sum Assignment Solver</li> <li>• Production Planning Service with a Hybrid Approach</li> <li>• Resource Allocation MAS-based</li> <li>• Truck Loading Service</li> <li>• Support Machine Maintenance using RAG and LLM</li> <li>• Document Transformation using LLM</li> <li>• Support Compliance for Clean Room using RAG and LLM</li> <li>• Calibration Certification Service</li> </ul>

**Table 3: Summary of DAI-DSS Building Blocks**

In addition, a summary of all developed AI services is given in Table 4. They are categorized by their maturity level, the type of AI, and their reliability. The maturity levels of the AI prototypes can range from:

- **Initial:** This stage corresponds to a conceptual definition of the prototype meaning that the idea is defined, but no implementation exists. This was not the case for the included AI services within this document.
- **Basic:** This stage corresponds to an experimental implementation meaning that a simple prototype is built to test core feasibility, but with limited functionality.
- **Intermediate:** This stage corresponds to a functional status, where a working prototype with key features is implemented but may lack scalability, robustness, or integration.
- **Advanced:** This stage refers to an optimized status, where a well-developed version of the prototype with performance improvements or partial automation is implemented, and real-world testing and demonstration by the use case partners are given.

AI-service	Maturity Level	Type of AI	Reliability
Support Understanding of Decisions through Conceptual Modelling	Basic	Rule-based	High
Decision Support through Decision Tree	Basic	Supervised Machine Learning	High
Resource Allocation using Neural Networks	Intermediate	Artificial Neural Network	Medium-High
Resource Allocation using Linear Sum Assignment Solver	Intermediate	Optimization Algorithm	High
Production Planning Service with a Hybrid Approach	Advanced	Reinforcement Learning, Constraint Programming	Medium-High
Resource Allocation MAS-based	Intermediate	Multi-Agent System	Medium-High
Truck Loading Service	Basic	Optimization Algorithm	Medium
Support Machine Maintenance using RAG and LLM	Advanced	Large Language Model, RAG	Medium-High
Document Transformation using LLM	Intermediate	Large Language Model	Medium
Support Compliance for Clean Room using RAG and LLM	Intermediate	Large Language Model, RAG	Medium-High
Calibration Certification Service	Advanced	Rule-based, Algorithm	High
Resilience Score Service	Intermediate	Rule-based, Algorithm	High

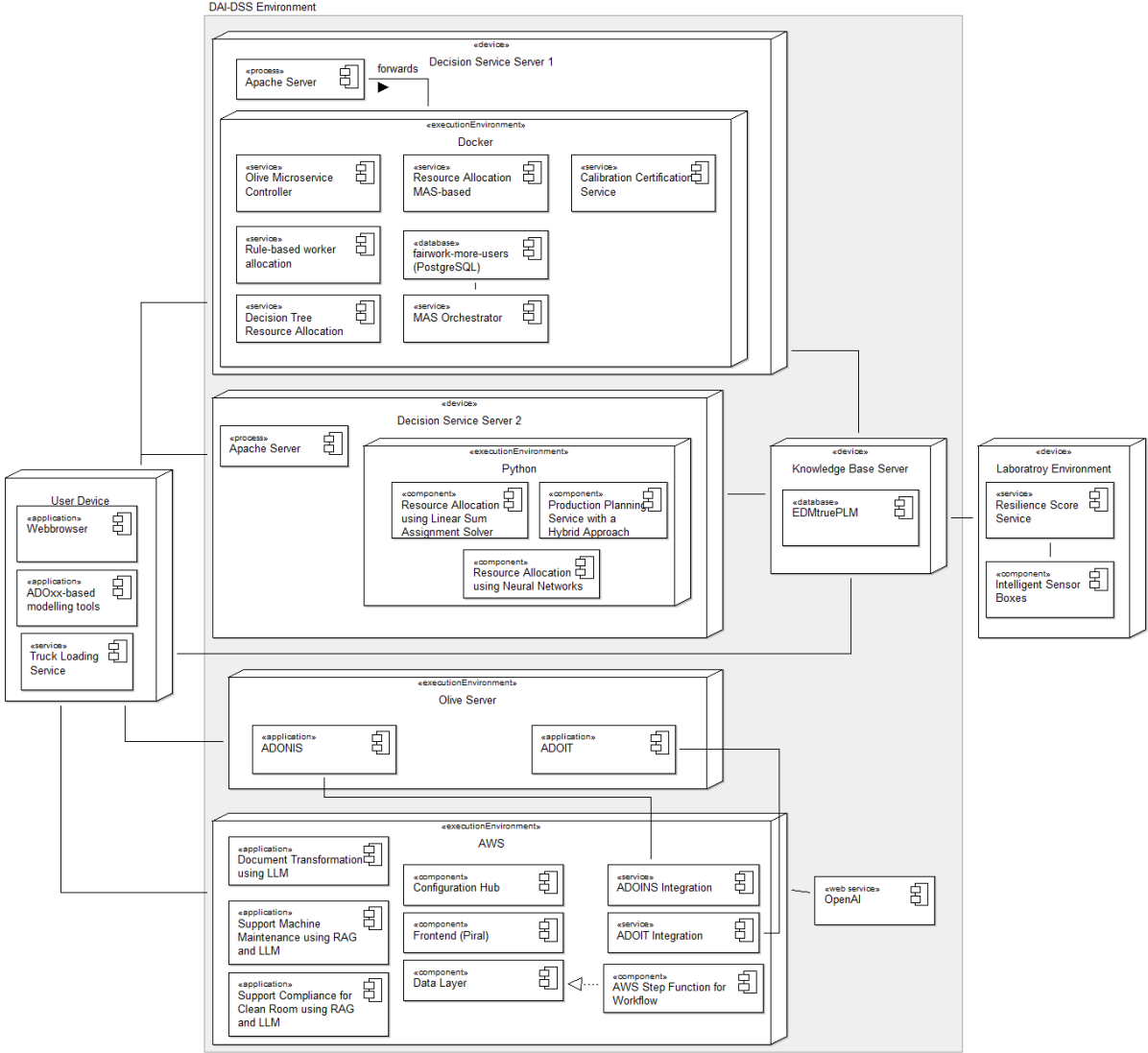
**Table 4: Summary of the AI Services**

The reliability differs depending on the type of AI. For rule-based AI it is indicated as a high, as it is deterministic and reliable for well-defined problems. Supervised ML is also attributed to High as it is reliable in controlled settings with high-quality data, however, it can degrade with poor data. ANN relates to Medium-High as its performance

depends on data quality, model architecture, and training. The optimization algorithm is defined as High as it uses deterministic approaches such as the Linear sum Assignment solver finding always the mathematically optimal solution. The Hybrid approach using RL and CP is defined as Medium-High as CP is very reliable for problems with well-defined constraints, while RL can have high adaptability but is less predictable and can struggle with generalization. MAS approaches also correspond to Medium-High as their performance depends on coordination mechanisms and the environment's complexity. LLMs are attributed to Medium reliability as they are prone to hallucinations and lack verification. In combination with RAG, their reliability is improved through retrieval but still depends on source quality.

# 4 PROTOTYPE DEPLOYMENT

This section describes the methodologies for deploying the individual components of the final DAI-DSS prototype. It presents a comprehensive understanding of the deployment lifecycle, from installation to execution. The deployment diagram in Figure 83 provides a high-level overview of how the different components and parts of the prototype are deployed. The deployment can be separated roughly into two parts (1) client devices running the interfaces and (2) the DAI-DSS prototype environment consisting of multiple servers, applications, and services working together.



**Figure 83: Deployment Diagram DAI-DSS Prototype**

The Knowledge Base is deployed on a server provided by JOTNE, which runs an instance of their EDMtruePLM software. It is accessible over an endpoint, allowing to read and save data. These endpoints are then used to integrate the Knowledge Base and its data with the other components of the DAI-DSS.

JR provides two servers on which decision services and the MAS Orchestrator were deployed. Both servers are publicly accessible and can therefore interact with other components of the DAI-DSS.

The Decision Service Server 1 runs some of the decision services, the MAS Orchestrator, and supporting services like the OLIVE Microservice controller and a PostgreSQL database. The components are run as Docker containers, easing the deployment as each service can have its own technology stack. Additionally, to support accessing the services over the internet, an Apache server was installed to manage forwarding the calls to the right service. The Apache server was used to have a common place to support HTTPS and allow to use of the standard ports from the outside. The calls are then forwarded to the internal service.

On the second decision service sever, decision services are directly started in a Python environment, using dedicated ports. The Python environment was set up and shared by the deployed services.

BOC offers its OLIVE Server and the Amazon Web Services (AWS) environment, where it was deployed on its own services and additional applications. On the OLIVE server, instances of the ADONIS and ADOIT modelling tools are deployed, which are used within their components. The AWS environment contains the implementation of the concrete logic for the services developed to tackle the use cases. Additionally, the components for the UI and the Configuration components are also deployed here. Last but not least, the applications in the AWS environment use an OpenAI endpoint, which is run on the environment of OpenAI and called from the services.

The user devices, shown on the left side of Figure 83 is an abstraction of all the devices users can use. Here they mainly will use the Web browser or ADOxx-based modelling tool to interact with the other deployed components. The True Loading Services is also implemented to run on user devices as a standalone tool.

Last but not least, the Laboratory Environment is set up in the infrastructure of JR, containing the services and components to run and use the Intelligent Sensor Boxes. This laboratory environment is set up locally to not send all private information to the central components, but only the processed and aggregated data is sent to the Knowledge Base.

Besides the interaction of the deployed components, different Software (SW), Hardware (HW), and Technologies are required and used. This is summarized in Table 5 for each single deployable component.

(Single deployable) Component	SW /HW Requirements	Technology
Configuration Environment	Hardware requirement AWS SaaS: None Hardware requirement Local: <ul style="list-style-type: none"> <li>RAM: &gt; 16 GB</li> <li>CPU: &gt; 4</li> <li>HDD: &gt; 64Gb</li> </ul> Software requirement: <ul style="list-style-type: none"> <li>Node.js</li> <li>Python v3.11</li> </ul>	AWS OpenAI
OLIVE Microservice Controller (as docker and direct installation)	Hardware requirement, tested on: <ul style="list-style-type: none"> <li>Processor: Intel i7-8550U (1,8 GHz)</li> <li>RAM: 16 GB</li> </ul> Direct installation: <ul style="list-style-type: none"> <li>Software Requiement: <ul style="list-style-type: none"> <li>Java 8</li> <li>Tomcat 8.5</li> </ul> </li> </ul> Docker installation: <ul style="list-style-type: none"> <li>Docker 27.1.1</li> </ul>	Maven DockerImage: tomcat:8.5-jdk8
ADOxx based modelling tools (Scene2Model, Bee-Up)	Hardware Requirement, tested on: <ul style="list-style-type: none"> <li>Processor: Intel i7-8550U (1,8 GHz)</li> <li>RAM: 16 GB</li> <li>Storage: about 500 MB</li> </ul> Software requirements: <ul style="list-style-type: none"> <li>Everything is installed with the tool</li> </ul>	ADOxx SQLite



Knowledge Base	<p>Hardware Requirement:</p> <ul style="list-style-type: none"> <li>Windows server OS</li> <li>Minimum Cores 10</li> <li>RAM: minimum 16 GB</li> <li>Storage: about 500 GB</li> </ul> <p>Software requirements:</p> <ul style="list-style-type: none"> <li>Tomcat 10.1</li> <li>Java JRE or JDK version SE 17+ 64 bit</li> <li>.NET Framework 4.5+</li> </ul>	Windows x64 Installer
MAS-Orchestrator	<p>Software requirements:</p> <ul style="list-style-type: none"> <li>Python v3.11</li> </ul>	Docker
MAS-Allocation service	<p>Software requirements:</p> <ul style="list-style-type: none"> <li>Java version 8</li> </ul>	Docker
Calibration Certification Service	<p>Software requirements:</p> <ul style="list-style-type: none"> <li>Python v3.9 or later</li> </ul>	Based on Open Source Libraries
Lin-Sum-Solver for Worker Allocation	<p>Hardware:</p> <ul style="list-style-type: none"> <li>CPU: At least one core with a clock speed of 2 GHz or higher.</li> <li>Memory: A minimum of 8 GB RAM.</li> </ul> <p>Software:</p> <ul style="list-style-type: none"> <li>Python: Version 3.9 or later.</li> </ul> <p>Package Repository Access:</p> <ul style="list-style-type: none"> <li>Access to PyPI (Python Package Index) or a similar repository for installing Python packages.</li> </ul>	Google OR-Tools
Production Planning with Hybrid Approach	<p>Software:</p> <ul style="list-style-type: none"> <li>Python: Version 3.9 or later.</li> <li>Package Repository Access: Access to PyPI (Python Package Index) or a similar repository for installing Python packages.</li> </ul> <p>Hardware:</p> <ul style="list-style-type: none"> <li>CPU: At least one core with a clock speed of 2 GHz or higher.</li> <li>Memory: A minimum of 8 GB RAM.</li> </ul>	Google OR-Tools
Rule-based worker allocation (as docker or standalone)	<p>Hardware requirement, tested on:</p> <ul style="list-style-type: none"> <li>Processor: Intel i7-8550U (1,8 GHz)</li> <li>RAM: 16 GB</li> </ul> <p>Software:</p> <ul style="list-style-type: none"> <li>Python: 3.9 (packages are provided in the projects as requirements.txt)</li> </ul> <p>Docker installation</p> <ul style="list-style-type: none"> <li>Docker 27.1.1</li> </ul>	Swagger
Decision Tree Resource Allocation (as docker or standalone)	<p>Hardware requirement, tested on:</p> <ul style="list-style-type: none"> <li>Processor: Intel i7-8550U (1,8 GHz)</li> <li>RAM: 16 GB</li> </ul> <p>Software:</p> <ul style="list-style-type: none"> <li>Python: 3.10 (packages are provided in the projects as requirements.txt)</li> </ul> <p>Docker installation</p> <ul style="list-style-type: none"> <li>Docker 27.1.1</li> </ul>	Scikit-learn (Python Library) Swagger

**Table 5: Overview of the Individual Deployable Components**

## 4.1 Deployment of Configuration Integration Environment, Workflow Engine and User Interfaces

Both the configuration and integration environment and the workflow engine are deployed in AWS as a combination of serverless AWS Lambda functions and AWS EC2 instances and rely on AWS DynamoDB for file storage and on AWS API Gateway for the exposure of the APIs. The UI components are deployed in an AWS Bucket and distributed through AWS CloudFront CDN.

In the context of the project, other local deployment options have been explored in order to not be bound to the AWS environment. In the following Table 6: Deployment Components of Configuration Environment are listed as the required components used for AWS deployment and Local Docker-based deployment.

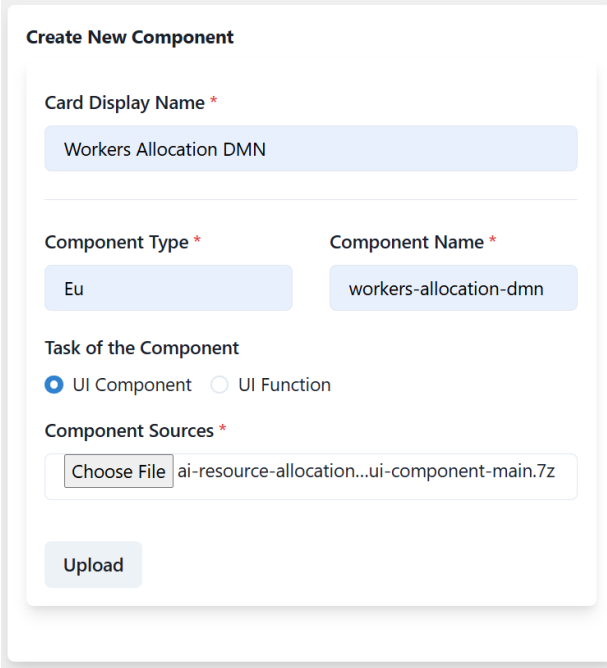
Components	AWS Serverless Architecture	Local Docker Compose
Function Execution	<a href="#">Lambda</a> ( <a href="#">Node.js</a> execution)	<a href="#">Express</a> with <a href="#">Node.js</a> ( <a href="#">Docker Image</a> ) <a href="#">OpenFaaS</a>
Function Lookup	<a href="#">AWS Lambda Environment</a>	Own <a href="#">Express</a> Service
Virtual Machine	<a href="#">AWS EC2</a>	<a href="#">Docker</a>
System Maintenance	<a href="#">AWS shared responsibility model</a>	manual
NoSQL DB	<a href="#">DynamoDB</a>	<a href="#">CouchDB</a> ( <a href="#">Docker Image</a> )
Object Storage	<a href="#">S3</a>	<a href="#">MinIO</a> ( <a href="#">Docker Image</a> )
Backup and Recovery Service	<a href="#">AWS Backup</a>	-
Secrets Management	<a href="#">AWS Secrets Manager</a>	Own (insecure) <a href="#">Express</a> Service
Parameter Store	<a href="#">AWS SSM</a>	Own <a href="#">Express</a> Service and/or global environment variables
DNS Management and Routing Service	<a href="#">AWS Route 53</a>	<a href="#">Cloudflare</a>
Content Delivery Network (CDN)	<a href="#">AWS CloudFront</a>	Own file-storage-service that can also deliver files.
API Gateway	<a href="#">AWS API Gateway</a>	<a href="#">Traefik</a> ( <a href="#">Docker Image</a> )
Resource and Provisioning Management	<a href="#">AWS Cloudformation</a> , OLIVE Deployment Script	Local Docker Image Registry, OLIVE Deployment Script

**Table 6: Deployment Components of Configuration Environment**

In order to automate the deployment of the configuration environment, specific deployment scripts have been created for both the AWS and the local deployment environment. In this way, no manual operations are performed and the complete deployment process is fully integrated into the CI/CD pipeline.

The configuration environment allows the user also to automate the deployment and registration of the different UI components in the UI configuration environment (Figure 84) when such components follow a specific format. In this way, the user has to only provide the name of the UI component, the category, and the label to fit in the UI configuration environment, and the zip package of the UI source code, which should be React-based, and the

system automatically build, deploy, and assign the component to a new entry in the UI builder part of the configuration environment.



The screenshot shows a 'Create New Component' form with the following fields and values:

- Card Display Name \***: Workers Allocation DMN
- Component Type \***: Eu
- Component Name \***: workers-allocation-dmn
- Task of the Component**:  UI Component,  UI Function
- Component Sources \***: Choose File ai-resource-allocation...ui-component-main.7z

An 'Upload' button is located at the bottom of the form.

**Figure 84: UI Deployment and Registration**

This functionality is accessible from the “Manage UI Components” tile from the Configuration environment.

## 4.2 AI-services Deployment

### 4.2.1 Decision Service Sever 1 and 2

The deployment of AI services within the project involves hosting and accessibility through dedicated servers provided by Joanneum Research. We have two servers: *Decision Service Server 1* and *Decision Service Server 2*. Both of the servers have an Apache<sup>16</sup> installed to be used as a reverse proxy, with an SSL certificate to enable HTTPS. The open ports for HTTP requests are 80 and 443 and port 20 is open to connect via SSH and manage the server itself.

On *Decision Service Server 1* a Docker<sup>17</sup> environment was set up, where the individual services and additional components are run in Docker containers. Additional components are for example the OLIVE microservice controller or the user database used by MORE. The containers can then be internally called by other components or made accessible to the outside by adding forwarding rules to the Apache2 server.

On *Decision Service Server 2*, an Anaconda environment management system is used, with dedicated Python environments for each deployed service. Each service runs within its own tmux session for better process management and isolation. The services are hosted as Web Server Gateway Interface (WSGI) servers, enabling Python-based Flask web applications to interact seamlessly with the Apache web server in a standardized manner. The Apache server acts as a reverse proxy, forwarding incoming traffic to the respective services.

<sup>16</sup> <https://httpd.apache.org/> (accessed: 8.1.2025)

<sup>17</sup> <https://www.docker.com/> (accessed: 8.1.2025)

This setup was chosen to separate the runtime environments of the different services and allow each of them to use the best-fitting technology stack. The containers can also be configured to restart automatically if a problem occurs and if need be, the service can be started multiple times to scale the application.

#### 4.2.1.1 Calibration Certification Service Deployment

The service is developed in Python 3.12 and comprises multiple components:

- **PDF Parser:** Utilizes fitz, Camelot, and pandas for efficient document processing.
- **Validator:** Built using standard Python libraries.
- **Viewer:** Leverages webbrowser and yattag for content display.

#### 4.2.1.2 Deployment of Model-based Decision Experiments for the Rule-based Resource Allocation and Decision-Tree Service

The service, making the decision, and offering the endpoint are deployed on *Decision Service Server 1*, with the environment as described there. This subsection contains additional, specific information about the Model-Based Decision Experiments, like where the code can be found or where the modelling tool and the extensions for the rule-based and decision tree experiment can be downloaded.

The modelling tool component of the decision tree experiment is unchanged and information on how to use it can be found here:

- <https://code.omilab.org/research-projects/fairwork/decision-services/decision-tree-resource-mapping>

The modelling tool prototype for the rule-based approach has two versions, where one contains everything including the functionality to automatically instantiate a decision service in a running OLIVE controller instance directly from the modelling tool and the other the functionality to export everything to manually instantiate a decision service. The difference is that the functionality for the manual creation can be imported into an installed Bee-Up instance and the automated instantiation needs an ADOxx instance which must be configured, needed more effort to set-up.

A description on how both of this prototype versions can be used, is available on its GitLab repository:

- <https://code.omilab.org/research-projects/fairwork/decision-services/bee-up-dmn-extension>

#### 4.2.1.3 Deployment of Worker Allocation Services and Production Planning Service

The worker allocation services with the neural networks approach (see section 3.5.3) and the LinSmSovler (see section 3.5.4), as well as the Production Planning Service with the hybrid approach (see section 3.5.5) are deployed on *Decision Service Server 2*. The deployment setup is the same as before. However, the production planning service was added and tested.

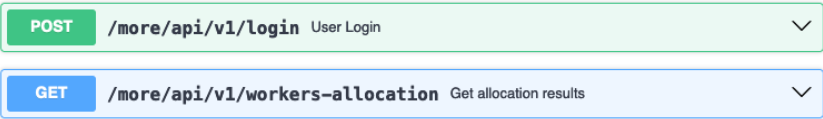
The code and setup information for the services are published online on GitHub:

- <https://github.com/Alexander-Nasuta/FAIRWork-AI-Service-Catalog>

#### 4.2.1.4 MAS-based Service Deployment

A RESTful API was developed to provide access to the results of the Workload Balance use case scenario using the Multi-Agent approach. This is a secured API developed in Python through the Flask framework and provides two endpoints to the users, as shown in Figure 85. The API also has administrative endpoints that are only accessible to the administrator and uses JWT – JSON Web Tokens – to protect the endpoint of the worker allocation

results. To do so, it is necessary to make a successful login by providing a valid username/password in the login endpoint. When that happens, a JWT token is created to be used in the workers' allocation endpoint.



**Figure 85: Available Endpoints to Registered Users**

For the production environment, a Web Server Gateway Interface (WSGI) has to be used: in this case, Waitress, operating on local port 5050. This server, running the web app, does not have direct connectivity with the outside world. Instead, it uses an Apache proxy server configured on the machine that can serve communications through port 443. All incoming and outgoing connections go through this Apache server, adding an extra layer of security to the communications.

Deployment-wise, a set of docker services was defined to establish the API with secure connectivity to achieve service integration within the FAIRWork platform. This includes the creation of a Docker image for the API, hosted in Docker Hub, and the necessary database to manage credentials information. The containerization of the web app allows the API to run continuously on the server without introducing any conflicts with other services running on the same machine.

The API is already running on the server, using SSL certification through the Apache server to encrypt/protect the communications.

### 4.2.2 AWS-deployed AI-services

The three AI-services Support Machine Maintenance using RAG and LLM, Document Transformation using LLM , Support Compliance for Clean Room using RAG and LLM are deployed in the AWS environment which corresponds to the description of deployment in the section 4.1.

## 4.3 Cost Factors for LLM Deployment

For some services where LLMs are integrated, it involves LLM deployment costs. These depend on factors like model size and type, API pricing, hardware, and the utilization of add-ons such as vector databases and RAG. When the LLM is accessed via API there is no need for hardware and can be easily integrated or scalable but the cost scales with usage which usually LLMs (e.g., OpenAI, Google Gemini, Anthropic) charge per token (input + output). For example, GPT-4-turbo API costs \$0.01 per 1K input tokens and \$0.03 per 1K output tokens. On the other hand, Self-Hosting and locally deployed alternatives include costs arising from the hardware, electricity, and maintenance. This refers mainly to running open-source LLMs like LLaMA, Falcon, or Mistral enabling full control and privacy over the LLM. Additional costs emerge when using optimization techniques, fine-tuning, or RAG. RAG can add further monthly costs for the vector database storage and retrieval depending on the provider. For example, Pinecone is free for 1M vectors, then \$0.096 per hour per pod while the open-source ChromaDB removes cloud fees but requires server maintenance as it is self-hosted.

Currently, the LLM is based on Cloud APIs by using OpenAI and GPT-4o. The API key is provided by the AI service provider resulting in costs for input tokens of \$0.15 per 1 million tokens and output tokens of \$0.60 per 1 million tokens. For the utilized vector database, AWS OpenSearch is calculated with 0,05\$ per hour per instance and 0,1\$ per month per GB. Also, AWS infrastructure costs including computing resources, storage and data transfer arise. Future steps and decisions can reflect whether a changing to a locally hosted LLM by the use case partner or using API key provided by the use case partners themselves.

## 4.4 Knowledge Base Deployment

The Knowledge Base (EDMtruePLM) web application contains a front-end, developed in JavaScript with the VUE.js framework, and a back-end, developed in Java with the Spring Boot framework. Front-end and back-end communicate with each other through the REST API of the repository.

Each call from the front end goes through the back end to the EDMserver. The backend may use several calls to EDM for one REST API call. The backend and the EDMserver communicate with each other through an internal TCP protocol, that is, the Java implementation EDMconnect. The web application, that is, the PLM application GUI is compatible with most modern web browsers. All the required components to run the knowledge Base were deployed on a standalone virtual machine running with Windows server operating system.

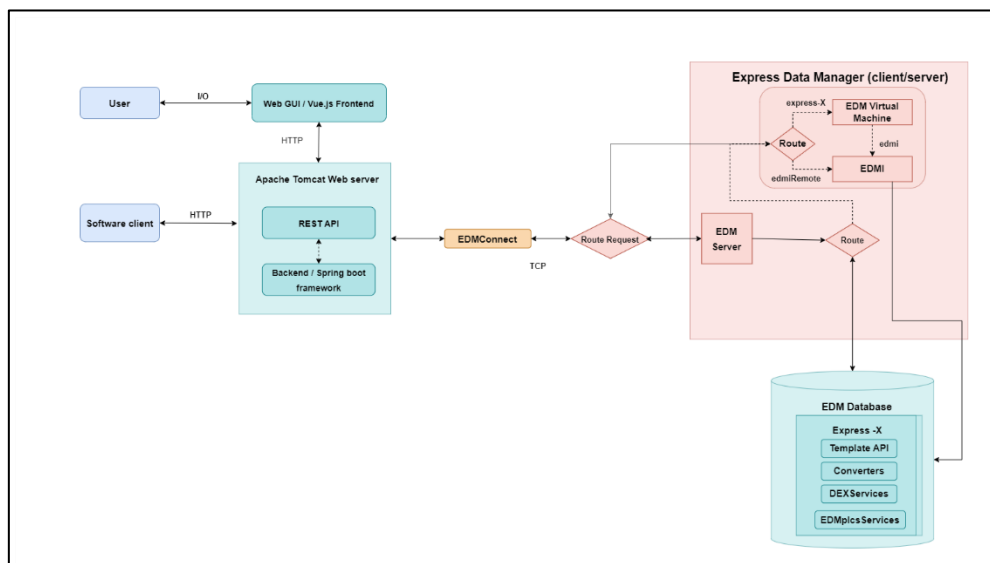


Figure 86: Knowledge Base Internal Architecture

# 5 EXTENDING DAI-DSS FOR NEW USE CASE SCENARIOS

---

## 5.1 Extending Workflows and User Interfaces

The modular architecture of the FAIRWork DAI-DSS enables it to support multiple use cases. Examples of this modularity include the ability to add and extend workflows and UI elements as needed, as well as the adaptable functionality to add and remove tiles from the configuration platform. In general, the configurator component should speed up and help with the system's (re-)configuration process. As outlined in the section 3.2.1, new workflows can be added to the workflow engine, or tasks can be added or removed to modify workflows that have already been created. Additionally, other forms of orchestration besides the MAS Orchestrator and workflow-based orchestration such as agentic workflows must be considered. To integrate services into workflows, tasks involving the calling of services may require input or output adaptations in addition to having a public endpoint. To access them in a configurable web application, existing UI components can be updated, or new UI components can be created and added to the UI component pipeline.

## 5.2 Extending Decision Services through Conceptual Modelling

One perspective during the design of the DAI-DSS was the ability to adapt it to new decision scenarios, increasing the flexibility of the system. In FAIRWork we chose a model-based configuration approach to support this flexibility, which is covered by the *Configuration* component and uses the corresponding design methodology (D2.1) combined with the three-layered modelling framework introduced in D3.2.

Conceptual, diagrammatic models are used to capture knowledge in a human and machine-understandable way, which are then used as input for the configuration, either the models can be directly used within the service, or they support the human users in understanding the decision scenario and then configure the DAI-DSS accordingly. Therefore, the models are either used as design artifacts to support the understanding of the users or they are directly used within the service as input for the decision services.

To enable the DAI-DSS to support a new decision scenario, first, the scenario must be understood. Therefore, a physical workshop is held, where domain experts with various backgrounds come together and discuss the idea on a high-abstraction level and identify the parts of and influences on the decision that must be considered, e.g., the parameters considered in the decision, which people are involved or where possible problems with the need for information or decision may lie.

Haptic figures are used within the workshops to foster creativity and co-creation between the participants. The created haptic models are then automatically digitalized and enriched to capture the knowledge and establish a representation of the common understanding of the decision scenario, which then later is used as input for creating more detailed models, describing the decision more precisely and allowing its integration in the DAI-DSS. In FAIRWork we used BPMN and DMN modelling languages. These models belong to the identification layer of the three-layered approach.

These models are then enriched, or new models are created to specify the concrete conditions for the decision, including technical details, needed for establishing the decision service within the DAI-DSS. Here the models from the identification layer are used as the basis, to reuse the already captured knowledge and establish a link between the models on different abstraction levels. This link is used to find models that belong together enabling to provide them to stakeholders, facilitating their understanding.



In the last layer, the collected knowledge is then used to configure the DAI-DSS to offer decision support for a concrete scenario. Depending on the service, the modelled information is directly used as input or manually translated by human users.

For example, the experiment introduced in the section 3.5.1 uses DMN models to describe the decision logic in the form of its structure and the rules defined in decision tables. The Bee-Up modelling tool, used for creating the models, was extended to allow for saving technical information, enabling to directly establish a decision service in a connected OLIVE controller. Through this new decision services based on rules can be created, without the need for excessive technical knowledge, but by focusing on the decision that should be made.

For the machine maintenance use case and the corresponding service, as introduced in the section 3.5.8. Here models created with ADOIT are connected to a concrete service, which uses modelled information to create answers to asked questions in natural language. To extend the answers that can be provided the modelled information must be changed or enhanced. Or a new service with new connected models can be created, enabling the DAI-DSS to provide answers for different scenarios.

Therefore, the extension of the DAI-DSS using conceptual modelling is based on two parts that must fit together. On one hand, the modelling method and the corresponding tool must be able to capture the needed information. On the other hand, the DAI-DSS must contain configurable services, that correspond to the concepts from the modelling tool and can be configured based on the captured knowledge. Or that services go a step further and can understand the models directly to use them to produce fitting answers to asked questions.

### 5.3 Extending DAI-DSS Capabilities through AI-Enrichment Services

Extending DAI-DSS capabilities through AI-Enrichment services The services are designed for specific scenarios. Depending on the specification depth, the existing services can be adapted to new scenarios with or less effort. Worker allocation or production plans for different plants must be analyzed. In case the planning works with similar variations and constraints, the services might be used almost exactly as they are now. However, plants can work differently and have different demands. The adaptations can be more severe if the differences are bigger.

As the connection to the different components of the FAIRWork prototype are standardized, it is possible to connect existing or separately developed algorithms as services to the orchestrator and with that to the UI and the knowledge base. Therefore, the algorithm must get a http-wrapper that allows the algorithm to be triggered via an http-request. The orchestrator must know of the algorithm as a new service in the system and must know about the necessary data. With that, the orchestrator gets the ability to create the json that includes the necessary data for this algorithm. With that, the algorithm becomes a new service in the system.

Numerous AI services adhere to the Gymnasium standard for Reinforcement Learning (RL), a framework for solving various tasks. This approach allows the underlying algorithms to be easily trained on new use cases without requiring changes to the learning algorithm itself. The only necessary adjustment is the environment, which refers to the formal model of the specific use case. In addition to RL, we provide a library compatible with the Gymnasium API that facilitates Monte Carlo Tree Search (MCTS). The library includes examples and test cases for well-known RL environments such as "FrozenLake" and "CartPole," as well as resource allocation use cases from the project. These implementations demonstrate that the algorithm is versatile and capable of finding solutions to a wide range of use cases and environments.



## 5.4 Extending Real-World Data Provisioning

Connecting a sensor box to sensors is straightforward and intuitive. The sensors are added to the sensor box with a little configuration. It works like a simple 'plug-and-play' system that allows new sensors with the appropriate data interface to be added.

## 5.5 Extending the Knowledge Base

The Knowledge Base is flexible for allowing the addition of new classification types with their respective attributes using the Reference data definition from the dedicated Graphical UI. Depending on the use cases, the project's breakdown structure can be expanded to include additional elements for storing necessary data to support a use case. Since the Knowledge Base offers standardized REST API services with swagger documentation for CRUD operations, querying the data from other DAI-DSS components requires minimal adjustments to the path and query parameters, ensuring seamless access to the updated information.

## 6 SUMMARY, CONCLUSION AND OUTLOOK

---

The documentation outlines the foundational elements and architecture of the system, as established in “D4.1 – DAI-DSS Architecture and Initial Documentation and Test Report” and “D4.2 – Initial DAI-DSS Prototype”. The development and deployment of the Final DAI-DSS Prototype have integrated various AI-driven decision support capabilities, addressing multiple industrial use cases. The final prototype extends the initial state, demonstrating flexibility across applications such as worker allocation, production planning, machine maintenance, and compliance documentation and validation. The structured approach based on the DAI-DSS High Level Architecture ensures modularity, and adaptability, which are critical for practical industrial implementation. This document, first, provides a comprehensive overview of the current problem settings and our DAI-DSS solution suggestions. Then the building blocks, their general structure and integral role within the system, and the individual prototypes for each block are described.

The DAI-DSS UI serves as the primary interaction layer, providing decision-makers with visualizations and control mechanisms. It integrates multiple components, including order overviews, worker allocation illustrations, chat interfaces, and validation tracking dashboards, ensuring understandability and accessibility by the users.

The DAI-DSS Orchestrator plays a central role in managing workflows and microservices. It enables seamless communication between various AI components, ensuring efficient execution of decision-support tasks. The workflow-based orchestration and multi-agent orchestration provide scalability and flexibility in handling complex industrial scenarios and demonstrate two distinct approaches to orchestrate.

The DAI-DSS Configurator allows the configuration of the UI and Orchestrator supporting the system's adaptability and integration with external systems.

The DAI-DSS Knowledge Base acts as a structured repository. The Knowledge Base stores essential data, including worker profiles, production records, production line details and maintenance documents. It enables the Orchestrator and AI services to access and process relevant information enhancing decision accuracy.

The DAI-DSS AI Enrichment covers a list of developed AI prototypes that have been incorporated into this building block. It includes many different services with different maturity, types of AI, and reliability. Neural Networks, Multi-Agent Systems, Constraint Programming, and Decision Trees are used for resource allocation and production planning. RAG and LLMs are applied for document transformation, information access to support compliance with cleanroom regulations, and maintenance support. Rule-based approaches are used for document verification or worker allocation.

The FAIRWork Final DAI-DSS Prototype demonstrates how effectively these building blocks are integrated into a cohesive system. Key results include the proposal of scalable decision support system based on the modular architecture to ensure adaptability across different industrial applications. Different AI integrations demonstrate a hybrid approach combining rule-based logic, machine learning, and agent approaches and the reflection of their reliability target to enhance transparency for decision makers. Also, the system's microservices-based structure allows for extensions, ensuring interoperability and extendibility for future adaptability to new AI models and datasets.

The DAI-DSS demonstrates its capability as a decision support system by integrating AI techniques, data and UIs for decision making problems to bring humans and AI together. The modularity of the architecture allows for continued evolution, ensuring scalability and applicability across various industrial domains. In particular, the established foundation and the covered problem settings and use cases target to benefit arising robotic use case challenges in manufacturing e.g. using the machine maintenance prototype to support the maintenance of robots,

assist decisions to optimize the robot-to-task and line allocation while considering human robot interactions, check compliance regulations for robots and their different types, support the configuration of robots etc.

The structured approach to data handling, workflow orchestration, and AI-driven decision-making has set a foundation for future advancements in AI-based decision support systems. While ongoing enhancements in data integration and AI reliability are researched, the prototype marks a step toward intelligent, explainable, and scalable decision-support solutions for industrial applications. Also, for future outlook, the FAIRWork DAI-DSS prototype aims to mark one pathway and contribution to an overall European AI Reference Architecture to raise competitiveness and technological sovereignty. The DAI-DSS architecture illustrates one way to implement different architectural elements by a set of European organizations.

## 7 REFERENCES

---

References are included as footnotes within the text.