INITIAL DAI-DSS PROTOTYPE

D4.2

Editor Name	Gustavo Vieira (MORE)
Submission Date	December 29, 2023
Version	1.0
State	Final
Confidentially Level	PU

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

EXECUTIVE SUMMARY

This deliverable, “D4.2 – Initial DAI-DSS Prototype”, is the first reference implementation focusing on the synthesis of fundamental components to establish a shared comprehension of the operational principles behind DAI-DSS. It aims to describe how the different DAI-DSS components fit together, forming a complete and cohesive structure in which the functionalities of the blocks are well-defined, the complementarity of the structure is explored, and the technological component is described in terms of fulfilling its purposes and integrating with the neighbouring structures that are part of the architecture.

The first part of this documentation deals with the description of the building blocks that form part of the architecture described in deliverable “D4.1 – DAI-DSS Architecture and Initial Documentation and Test Report“. Initially, a quick overview of what the building blocks are and a brief explanation of their general structure are given. We then move on to the integration of the DAI-DSS User Interface (UI) into the system. Taking Industrial Partner CRF's Workload Balance Use Case Scenario as a reference, an overview of the UI in this setting is provided. It describes how the UI offers the possibility of visualising the important components directly linked to decision-making in this example, as well as the added value offered by this building block. The UI is essential for efficient decision-making. It provides the appropriate interface between the decision-maker and the system in order to enable a clear visualisation of the decision-making process. Some images provide an introductory view of the UI in this initial prototype.

Next is the DAI-DSS Orchestrator. The Orchestrator plays a very important and central role in the architecture of this system. Its purpose is to provide coordination in the exchange of information and decisions between the various parts of the system. Like the conductor in an orchestra, the DAI-DSS carries out the decision-making process, and the involved individual services retrieve the relevant data from the knowledge database. This currently takes place in two parallel ways. At a more advanced stage is the Workflow-based Orchestrator. In this orchestrator, a workflow engine offers the possibility of executing workflows, i.e., the execution of tasks that follow a well-defined path leading to the triggering or realisation of an AI service that offers a recommendation for decision-making on issues identified in the business process for which the system is built. For demonstration purposes, a workflow was built in which worker data is retrieved, and the order of a part is requested in the Workload Balance Use Case Scenario. The step-by-step execution of the orchestration is documented in order to elucidate how the orchestration of the building blocks can take place in a real, concrete case from the manufacturing industry in this example.

It presents the DAI-DSS Configurator, a tool designed to enhance decision support systems through efficient configuration and integration frameworks. It explains the configurator's components: the Configuration Framework, which assists in creating decision models and strategies, and the Configuration Integration Framework, which generates system configurations from these models. The report details the prototype's configuration steps, illustrated for clarity. It also allows for microservices and workflow configuration, featuring a user-friendly 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 report covers the integration of AI services using REST-API endpoints, emphasising ease of access and industry-standard practices like Swagger documentation. It explores various algorithms, including Neural Networks and Decision Trees, and the integration of human factors data for assessing worker resilience. Deployment details of the AI services are discussed, focusing on hosting, management, and future enhancements like data catalogues and decision pattern services for resource allocation and predictive maintenance.

Finally, the report addresses the extension of the DAI-DSS with new services for various industrial use cases. It discusses customising services through conceptual modelling and the advantages of rule-based approaches over machine learning methods. The services aim to optimise resource allocation, improve productivity, and address human factors, thereby enhancing the system's decision-making capabilities in complex industrial scenarios.

PROJECT CONTEXT

Workpackage	WP4: Development of DAI-DSS
Task	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
Dependencies	WP2, WP3, WP5

Contributors and Reviewers

Contributors	Reviewers
Herwig Zeiner, Lucas Paletta, Michael Schneeberger (JRS) Gustavo Vieira (MORE), Higor Rosse (MORE), Rui Fernandes (MORE) Sylwia Olbrych, Alexander Nasuta (RWTH Aachen) Magdalena Dienstl (BOC) Christian Muck (OMILAB) Rishyank Chevuri (JOTNE)	Herwig Zeiner (JRS) Khadija Sabiri (MORE) Tord Kaasa (JOTNE)

Approved by:

Version History

Version	Date	Authors	Sections Affected
1.0	December 29, 2023	Gustavo Vieira	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
Under the following conditions:	
	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 CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Purpose of the Document

This document provides an initial implementation of the DAI-DSS architecture prototype of the FAIRWork project. Taking „D4.1 – DAI-DSS Architecture and Initial Documentation and Test Report“ as a base, this deliverable documents the technical implementation of the building blocks contained in the proposed architecture, highlighting how the technologies explored in the project are integrated to create a cohesive decision-support system.

Its primary objective is to outline the methodologies and processes involved in developing and implementing user interfaces, orchestration, configuration, access to knowledge base, and AI services within the DAI-DSS framework. The document aims to serve as a detailed guide for understanding the system's operation, highlighting the integration of AI services, the use of conceptual modelling for decision-making, and the deployment strategies for prototypes. It is intended for stakeholders who seek a deeper comprehension of the DAI-DSS's functionality and application in industrial contexts and decision-making processes.

1.2 Document Structure

The document is organised into several key sections, each focusing on distinct aspects of the DAI-DSS initial prototype development.

In Chapter 2, the implementation of the architecture’s building blocks is approached. Starting with the user interfaces, this section details the development and deployment of user interfaces, describing the creation and combination of UI components to support managerial roles in production units. The implementation of the DAI-DSS Orchestrator, Configurator, Knowledge Base, and AI services is tackled in this chapter.

In Chapter 3, relevant documentation on deploying various prototypes, including AI services and user interfaces, is given. This section covers the deployment of a modelling tool, experimental services, and the management and hosting of AI services on servers.

In Chapter 4, some ideas for extending DAI-DSS capabilities are discussed, delving into the potential extensions of DAI-DSS for additional use cases, applying conceptual modelling for various scenarios, focusing on intricate resource allocation challenges, and proposing significant approaches.

In Chapter 5, we conclude the current developments and give an outlook on the next steps in prototype development.

2 BUILDING BLOCKS AND THEIR INTEGRATION

This chapter aims to describe how the building blocks within the FAIRWork architecture were implemented. It provides a detailed overview of each component, explaining its purpose, design considerations, and integration into the overall system.

2.1 Overview of the building blocks

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, described, and their relevant features are presented. A detailed description of the initial architecture is given in Deliverable D4.1, including the technical implementation of the basic core services or application-specific services.

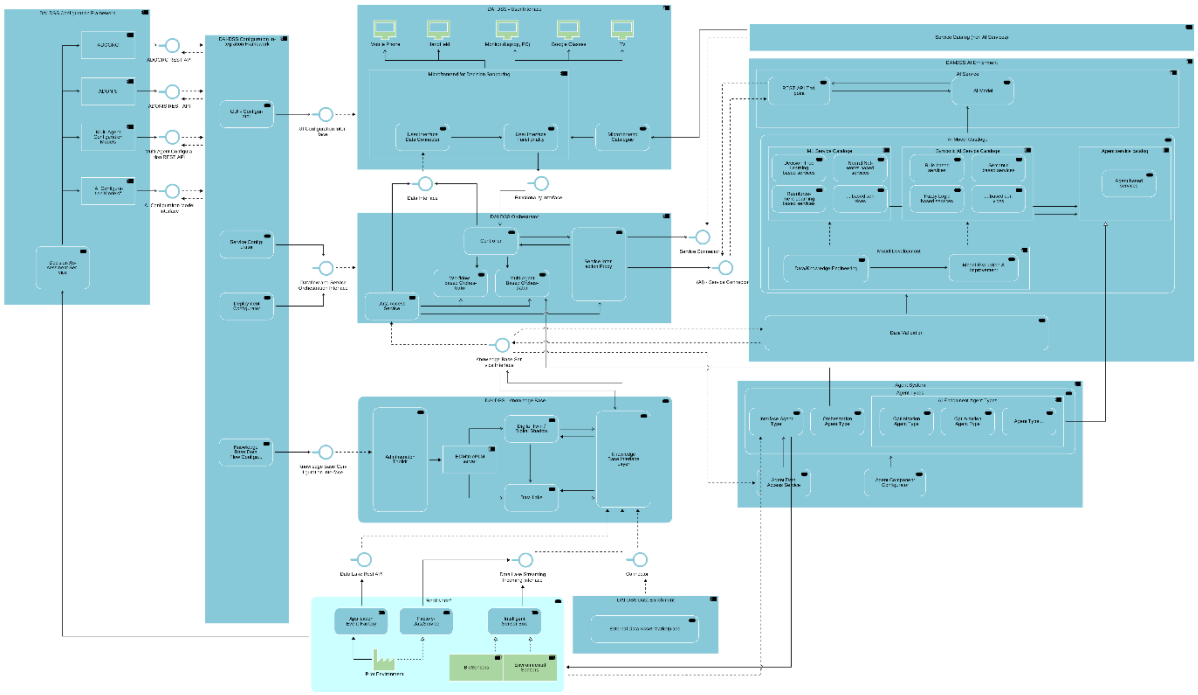

Figure 1: Overview of high-level architecture (HQ image at Annex B.18).

User Interface: This component is essential in the decision-making process as it must provide decision-makers with a clear understanding of the potential alternatives and possibilities available to them in a supportive environment. These interfaces can be displayed on various devices as per the user's needs, depending on the working conditions and circumstances. The main purposes of the interfaces are to present significant KPIs or other information and to allow the decision-maker to interact with the user. An instance where presenting KPIs could be beneficial is overseeing various statuses in a production hall.

AI Enrichment Service: This service comprises AI algorithms and models that aid decision-making. Its primary objective is to offer a compilation of data-driven models through AI enrichment in the environment. The approach involves using explainable, conventional AI algorithms and more advanced solutions for agent representation. As every agent is distinctive, the selection and development of the model are handled separately. Each model or optimisation algorithm contributes to the decision-making process, making the decentralised and fair approach more attainable.

Besides this, we have some relevant components which enable decision marking:

Configurator: This component connects different model environments to the user interfaces and orchestrator element of the DAI-DSS system, allowing model awareness. The model environments can include business process models, dashboard models, and models for Multi-Agent Systems and AI configuration. The configurator consolidates various model types and produces decision models that integrate the configuration data for other system components.

Orchestrator: The Orchestrator plays a crucial role in the decision support system as it manages and coordinates the configured services and workflows necessary for providing decision-making options for the end user. The services can either be standalone or linked in the form of a workflow.

Knowledge Base: The Knowledge Base is a pivotal component of the infrastructure, serving as a central storage for most of the data entailed in decision-making configurations. Such configurations demand data from diverse sources (e.g. humans, machines, processes) and yield outcomes. The purpose of the Knowledge Base is to gather all required data sets and streams, organise them, potentially integrate them, save them, and then present an interface layer for other DAI-DSS components to access this data. To ensure that AI algorithms can learn from previous decisions and that processes can use past results, these decisions and results are likewise recorded in the Knowledge Base.

2.2 Integration of User Interfaces

Over the course of this project, different user interfaces for the various participants in the decision process should be developed and deployed on different devices. This should be realised by providing various UI components that can be reused and combined if needed for different use case scenarios. As a first version, we chose to create three different UI components that can be combined to create a web application for supporting the manager of a production unit in one of our use case scenarios. The manager is in charge of allocating the workers to different production lines while considering various decision parameters. Demonstration data was created based on our use case partner's decision models and example data.

2.2.1 Order Overview Component

This UI component consists of a button and a table that displays all orders that are considered in the example (see Figure 2). When the button on the top is clicked, the current order information is retrieved from the knowledge base. For this example, we retrieve data from the knowledge base for three orders. Each order consists of a geometry that is produced on a certain production line. Additionally, the production priority, how many days are left until the due date, and the required workers to handle the geometry are displayed.

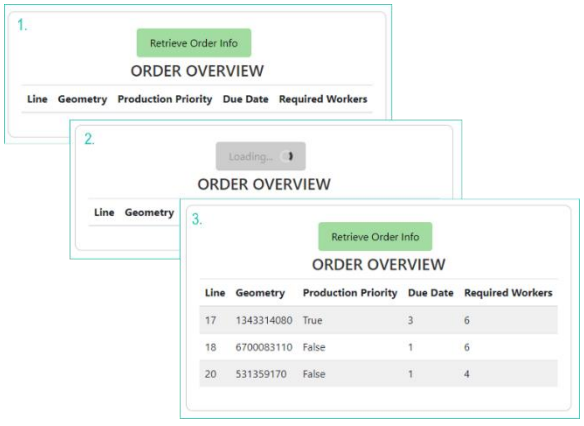

Figure 2: Order overview UI component.

2.2.2 Worker Overview with Manual Allocation Component

This UI component (Figure 3) 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. Before starting with the allocation, the user needs to retrieve the necessary data from the knowledge base by clicking on the corresponding button on the top. After the data is loaded, a table for each production line is available where all the workers and their characteristics are listed. The user can switch between production lines by using the tabs at the top of the table. Not all workers are allowed to work on each line. The column on the right side of the table, named *"Allocation to line X",* contains buttons to indicate the worker's suitability. Only if a worker is available and fulfils the medical condition to work with the geometry the button is green and can be clicked to allocate a worker to the corresponding line. Otherwise, the button is red and disabled. If workers are allowed to work on the line but are already allocated somewhere else, this is indicated by an orange disabled button. In the line overview section below the table, three cards representing the production lines are visualised. The currently proposed workers are displayed on these cards. A counter at the bottom of the card indicates how many workers are needed on the line.

Figure 3: Worker Overview UI Component.

2.2.3 Allocation proposal through AI resource allocation service component

This component shows the name of an AI service and a short description. Under the description, the user can find two buttons, one for triggering the AI service and one for clearing the results. When the user clicks on the “Trigger AI service” button, a request for calling the corresponding workflow is made. While the AI service is executed and the data for displaying the result is fetched, the label “Loading” is shown on the button. When the response is successful, the result is displayed below in the form of three cards. Each card represents one production line, and the workers suggested for this line are listed on it. Besides the ID of the workers, relevant characteristics are listed to give the decision-maker more insights into the suggested workers (see Figure 4). In the example, the results of the experimental rule-based service (see section 2.6.1) are presented, but this component can be configured for other allocation services as well. The configuration is realised by specifying the properties of the component, which are the name of the service, its short description, and the workflow's name triggering the service.

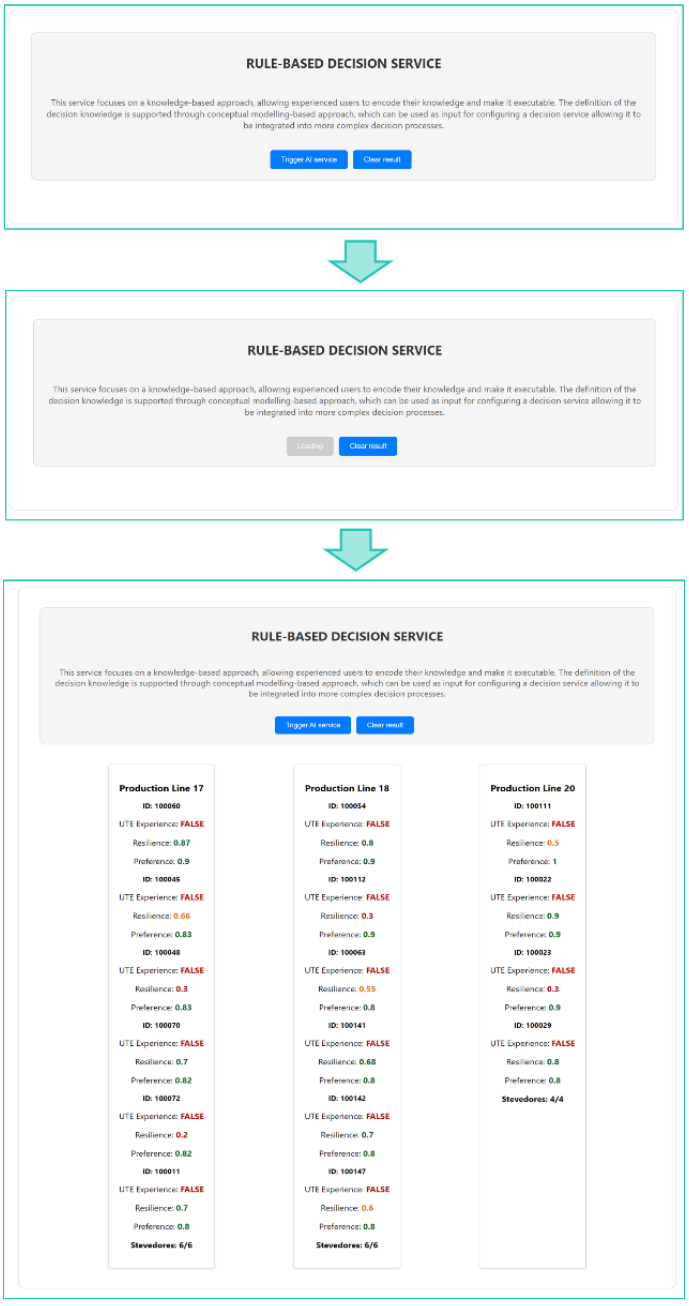

Figure 4: AI resource allocation service component.

Available UI components can be put together and deployed as web applications. The UI components are currently created with React[1]. If a component is finalised, it is uploaded to the OLIVE component pipeline. Components that are uploaded to the pipeline in the form of zip files can be accessed via the configuration environment. The configuration steps will be explained in more detail in section 2.4.2.3.

2.3 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.

2.3.1 Workflow-based Orchestration

As described in Deliverable 4.1[2], the DAI-DSS Orchestrator enables workflow-based orchestration using a Workflow Engine. In the current prototype, the workflow Engine "Conductor"[3] is used, and its configuration environment can be accessed via the configurator of our architecture (see Section 2.4). Within the Conductor setup environment, users have the option to navigate between sections by utilising 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 5. 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.

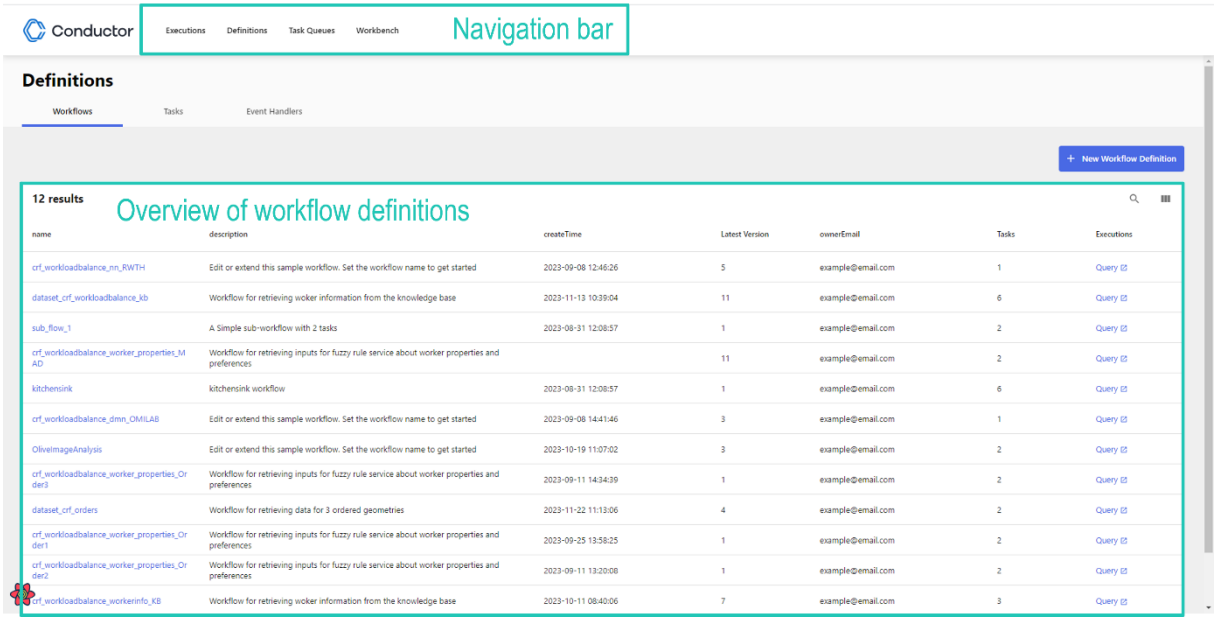

Figure 5: Configuration environment of the workflow engine.

[1] Meta Open Source. (2023). React. https://react.dev/

[2] Chevuri, R. (2023). DAI-DSS architecture and initial documentation and test report. https://fairwork-project.eu/deliverables/D4.1_DAI-DSSArchitecturev1.0a-preliminary.pdf

[3] Conductor. (2023). Basic Concepts - Conductor Documentation. https://conductor.netflix.com/devguide/concepts/index.html

2.3.1.1 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. For demonstration purposes, a workflow for retrieving data containing worker and order information was created. The required data is stored in different ways in the knowledge base. Depending on how the needed information is stored, different API calls or combinations of them are necessary to retrieve it. For each case, a service was created using the microservice controller. The instance of the microservice controller used in the FAIRWork project can be reached via the “Microservice Controller Tile” in the Configurator (see Section 2.4.2).

Public microservices can be selected in the user interface of the microservice controller. For example, the microservice “FAIRWork Knowledge Base public” contains several microservice operations for handling the token creation 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 6. 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 2.5 of this document. The so-created microservices have been further used and combined within workflows, which results in different workflow definitions.

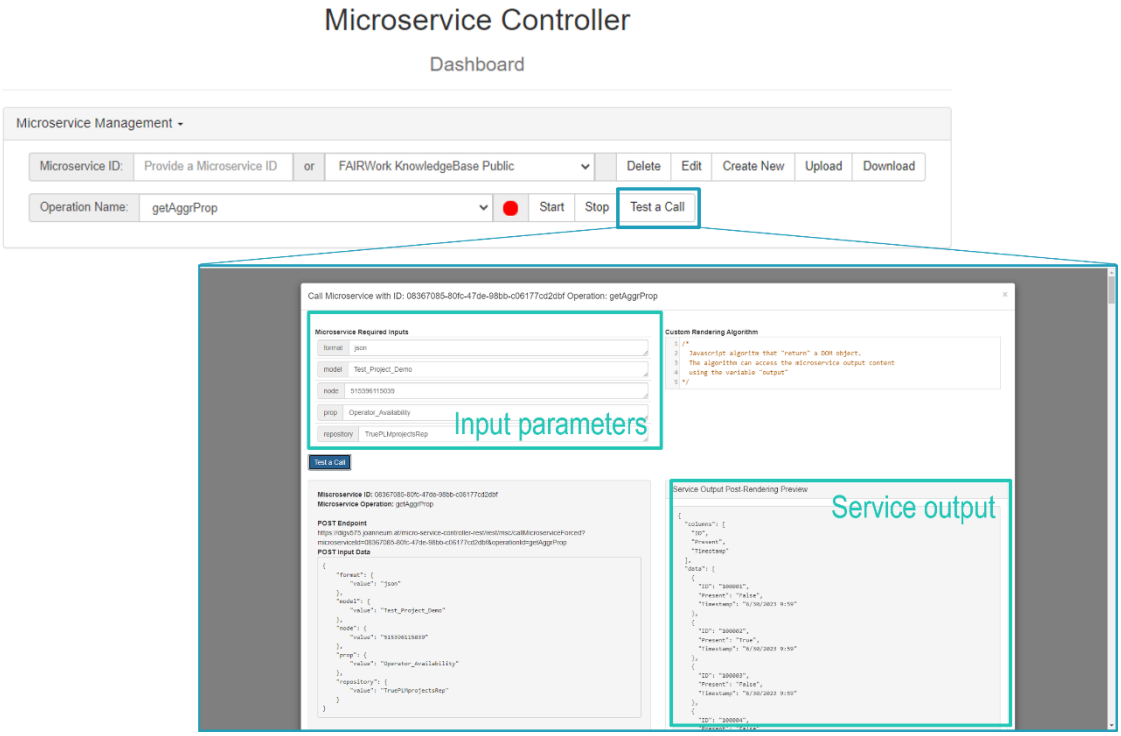

Figure 6: Test call of a microservice operation for retrieving data from the knowledge base in the microservice controller.

In the current instance of the workflow engine, workflow definitions have already been specified. All workflow definitions can be found in the corresponding section of the workflow configuration platform. Each definition can be inspected in more detail or modified by clicking on the workflow name in the list. By doing so, the user can find the workflow definition in code on the left side of the screen, while a visual representation is provided on the right side. All information required to specify how a workflow should behave is contained in the workflow definition. The tasks

property, which is an array of task configurations, is the most significant component of this definition. It can be seen as the blueprint for the workflow. There are different types of tasks available that can be referenced by the task configurations: simple tasks, system tasks, and operators[4]. For the initial prototype, we only used two types: system tasks and operators. A system task is one that entails carrying out a system-level operation, frequently interacting with external systems or services. These tasks are intended to handle interactions with external components and services that are not part of the workflow but are required for the overall execution of the workflow. An example of a system task 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 utilised to handle branching and decision-making within the workflow. They aid in controlling the flow based on the conditions or outcomes of prior tasks. Operators, unlike system tasks, do not represent a unit of work but rather manage the flow of execution based on task outputs.

An example of a workflow definition overview can be found in Figure 7. 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 tasks output. In the workflow example, the 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.

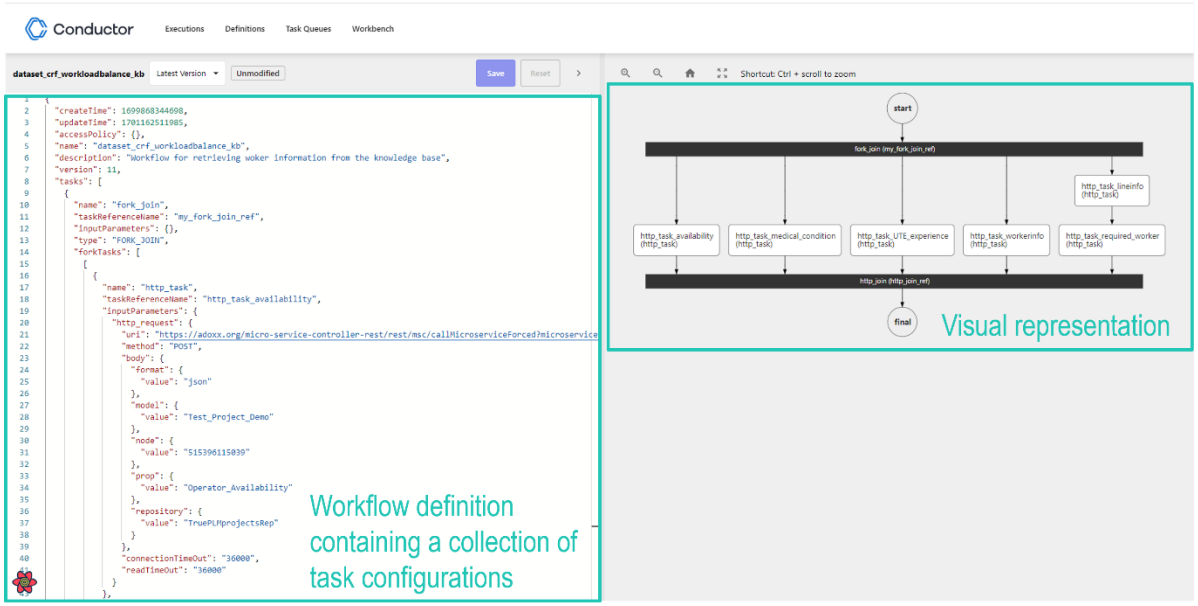

Figure 7: Example of a workflow definition for retrieving information from the knowledge base.

2.3.1.2 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

[4] Conductor. (2023). Tasks — Conductor Documentation. https://conductor.netflix.com/devguide/concepts/tasks.html

executed. The user can click on the workflow ID on the right side to inspect the running workflow in more detail (see Figure 8).

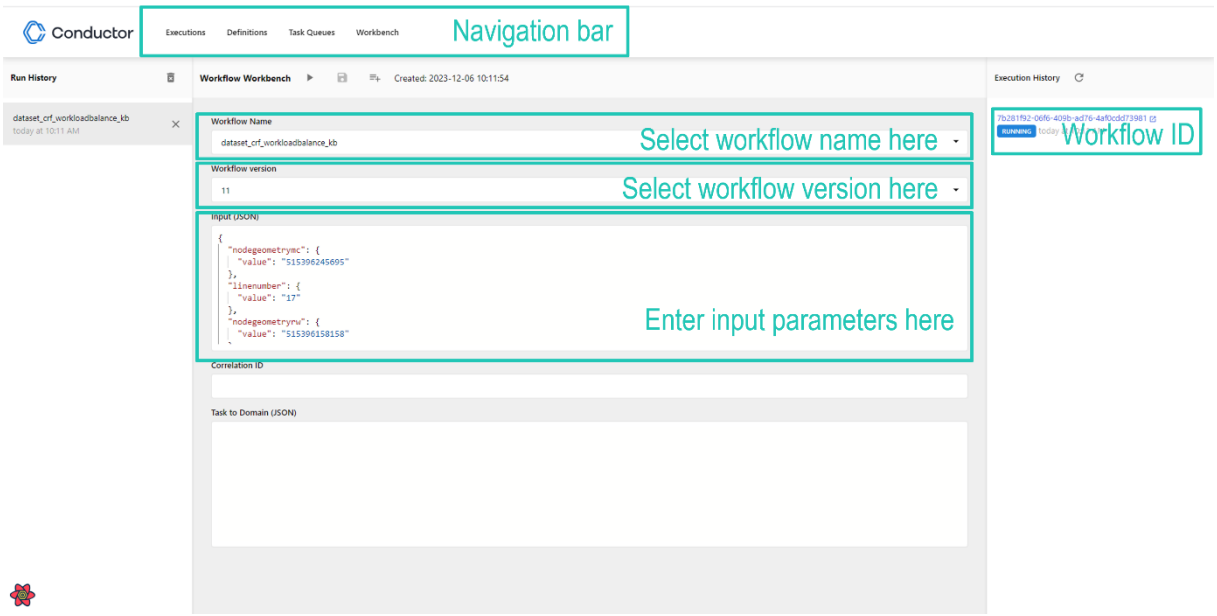

Figure 8: Workflow Workbench.

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 9).

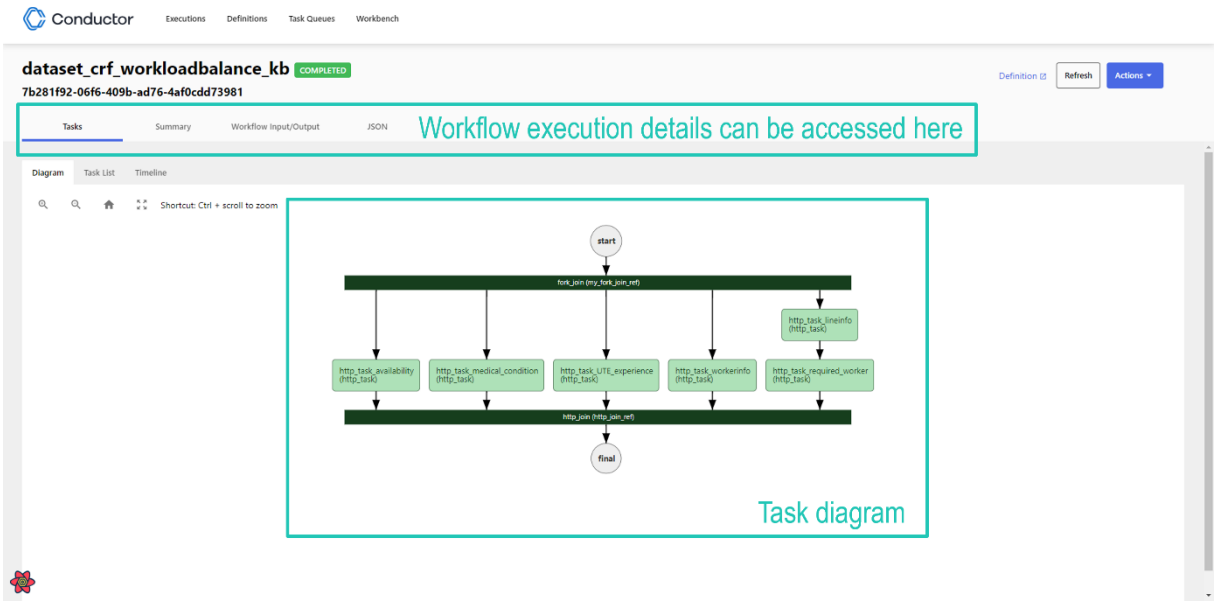

Figure 9: Workflow execution details.

2.3.1.3 Storing and Using Defined Workflows

Workflows can be further used in other components of the system, like the user interfaces or other services and workflows. This can be done using the APIs provided by the OLIVE microservice framework. 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 10, 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[5] expressions.

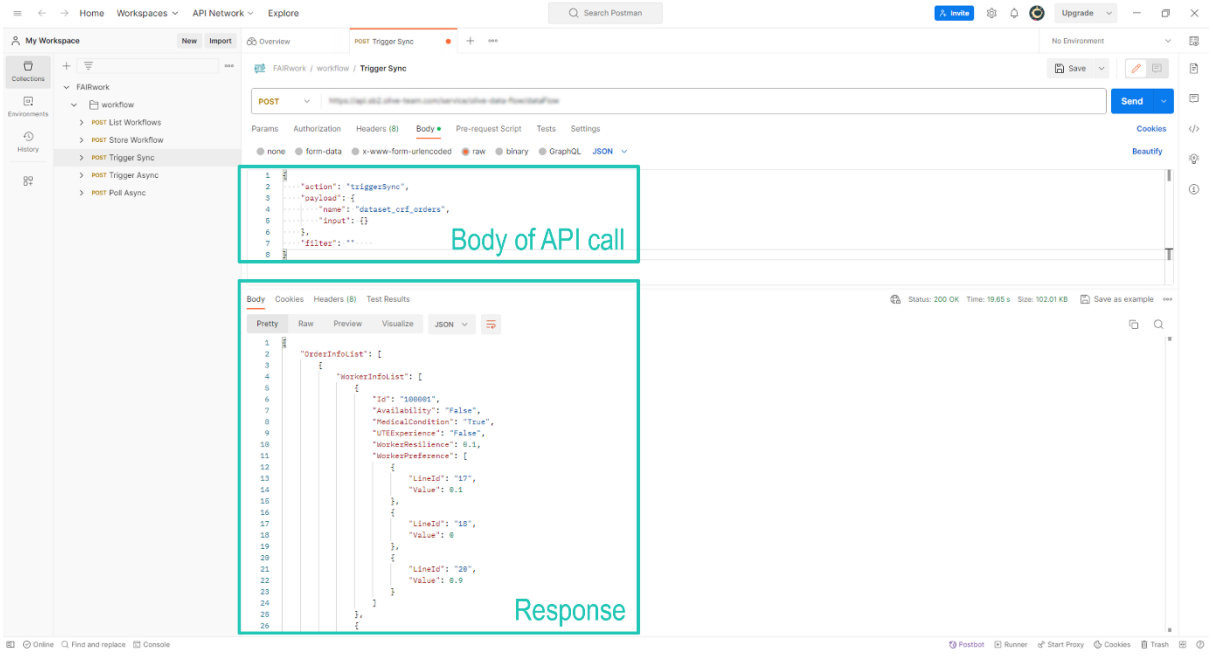

Figure 10: Example of an API call for triggering a workflow in Postman.

2.3.1.4 Demonstrator example

For the initial prototype, three workflows have been created to demonstrate a possible usage in a resource allocation scenario. The workflow engine can be accessed by clicking on the brown tile called “Workflow Engine” in OLIVE's configuration environment (see Section 2.4.2). 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. As this instance by now is shared with different instances of the OLIVE framework, there are more workflows listed that are not covered in the present example. The workflows relevant to this demonstration have the following characteristics (see Table 1) and can be examined in more detail by clicking on them:

Table 1: Workflows.

Workflow name	Workflow version	Input parameters
dataset_crf_workloadbalance_kb	11	**node_geometry_mc** – indicates the ID of the breakpoint element in the knowledge base describing the medical condition for a certain geometry. **linenumber** – indicates the ID of the line where the order will be produced.

[5] JSONata.org. (2021). JSONata Documentation. http://docs.jsonata.org/

		node_geometry_rw – indicates the ID of the breakpoint element in the knowledge base describing the characteristics of a certain geometry.
dataset_crf_orders	7	none
crf_workloadbalance_dmn_OMILAB	11	none

Additionally, the user can execute the workflow by entering and selecting the values displayed in the table above in the workbench section. 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"
}
```

The first workflow mentioned in the table retrieves necessary values from the knowledge base needed by the decision maker or an AI service 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 11.

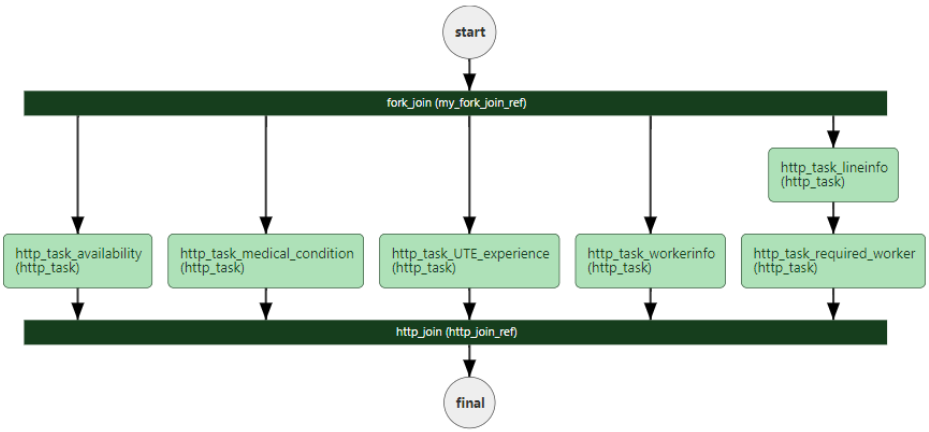

Figure 11: Execution Path of workflow "dataset_crf_workloadbalance_kb".

The workflow with the name “dataset_crf_orders” (see Figure 12) calls the workflow described above three times in parallel to retrieve the required information for three orders. Its output serves in this demonstrator example as the input for different AI services.

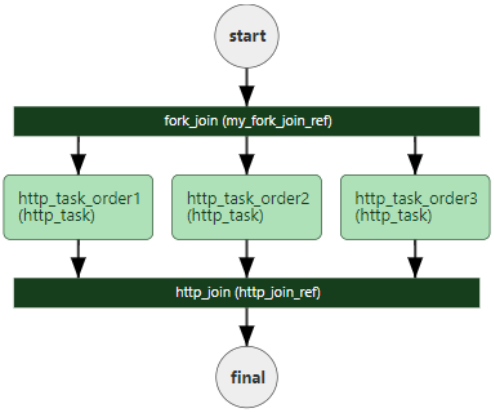

Figure 12: Execution path of workflow "dataset_crf_orders".

The last workflow example (see Figure 13) should illustrate that the above-described workflow can be extended by an AI service. So the workflow “crf_workloadbalance_dmn_OMILAB” first retrieves data from the knowledge base, and this data serves as input for a rule-based decision support service, which proposes which workers should be allocated to certain production lines. The workflow includes additional HTTP tasks to transform the input and output data as needed. More information about the decision support service can be found in section 2.6.1.

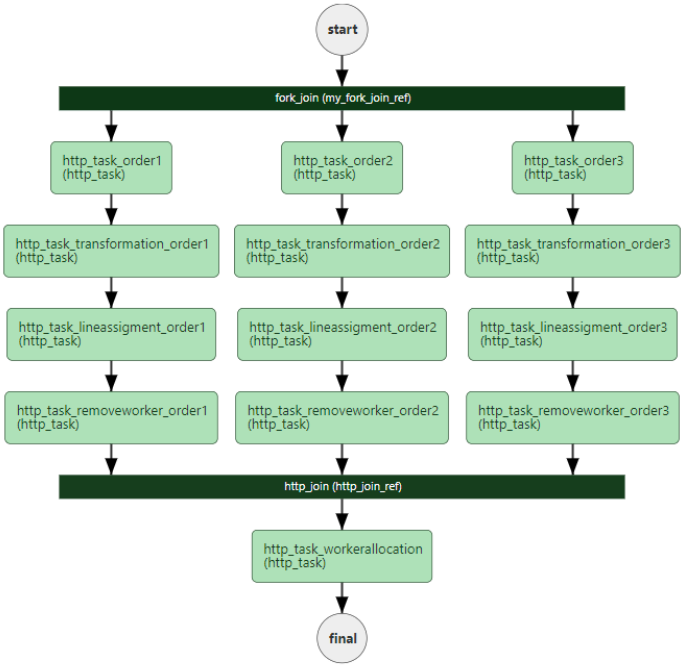

Figure 13: Execution Path of workflow "crf_workloadbalance_dmn_OMILAB".

2.3.2 Multi-Agent Orchestration

When combined with microservices architectures, Multi-Agent Systems play an important role in creating decentralised 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 decentralised 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 decentralised nature of microservices.

An orchestrator is being developed to provide access to the results of the microservices working on the Workload Balance use case scenario. This is a secured element developed in Python through the Flask framework and provides two endpoints to its users, as shown in Figure 14. 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.

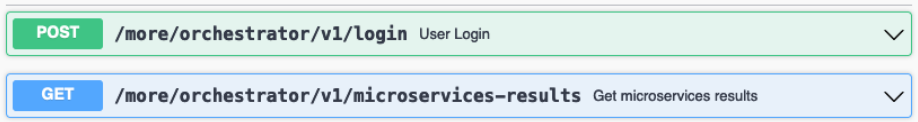

Figure 14: Available endpoints to registered users.

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, hosted in Docker Hub, and it was also used as a database to manage the orchestrator user credentials information. The containerisation 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 will run on the server, using SSL certification through the Apache server to encrypt/protect the communications.

2.3.2.1 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 15.

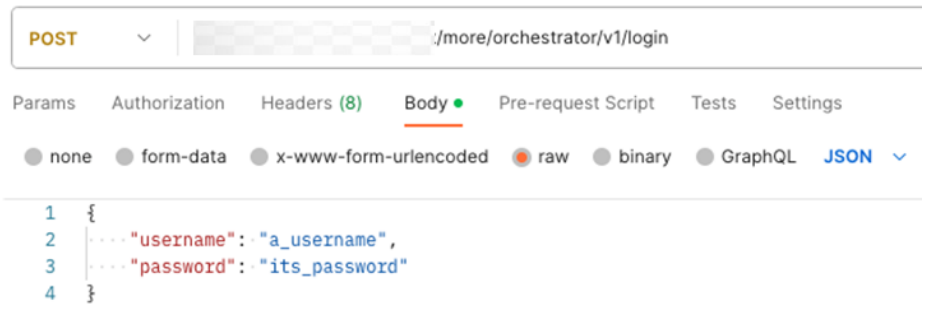

Figure 15: Login endpoint usage.

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 16. Token validity is set for 30 minutes.

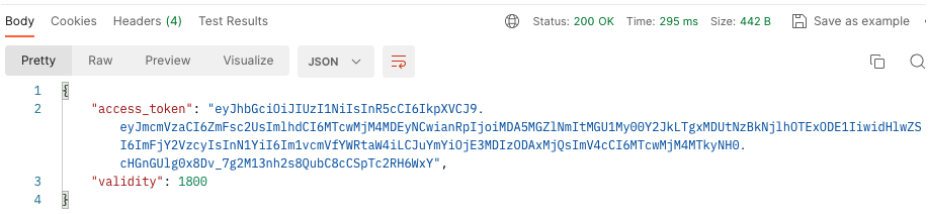

Figure 16: Successful login response.

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 17.

Figure 17: Login endpoint data models.

2.3.2.2 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 18 shows the configuration of the REST request to use this endpoint properly.

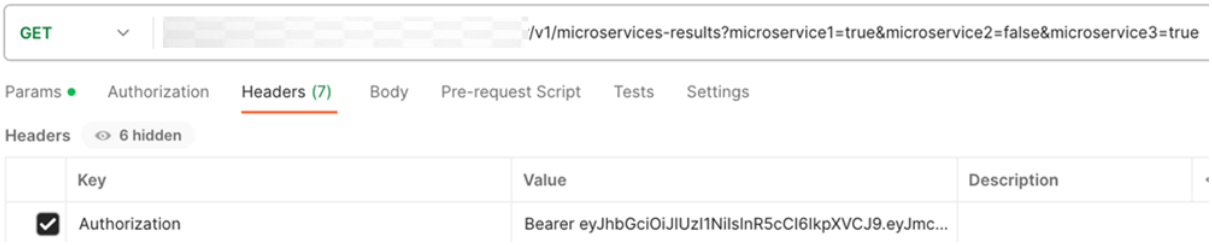

Figure 18: Microservices allocation results endpoint usage.

The endpoint accepts parameters to define which microservices should be inquired to get their results. This is presented generically in Figure 19.

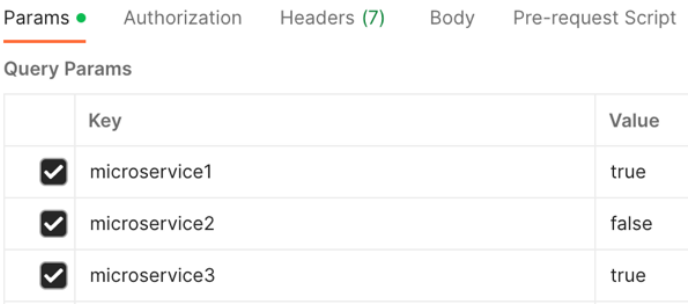

Figure 19: Generic parameters structure to be used in the GET request.

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 20 illustrates how the multi-agent orchestrator interacts with the microservices of the CRF Workload Balance use case scenario.

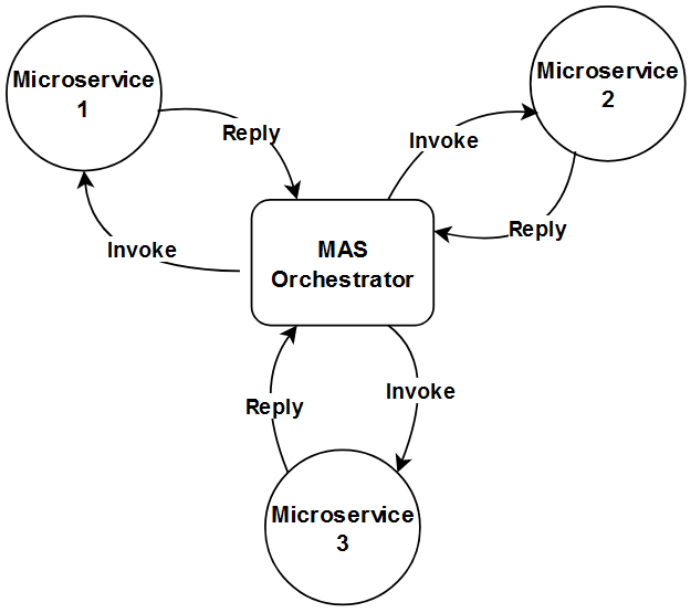

Figure 20: MAS orchestration of microservices.

2.4 Integrating Configuration

As described in Deliverable D4.1[2], the DAI-DSS Configurator consists of two parts: the Configuration Framework, and the Configuration Integration Framework. While the former enables the creation of models describing the decisions and their strategies using several modelling environments, the Configuration Integration Framework enables the generation of configurations for other components in the system derived from the previously created models. In this sense, the DAI-DSS Configurator can be seen as a set of tools to speed up the configuration of the decision support system.

2.4.1 Configuration Framework

2.4.1.1 Decision Models with BPMN

First, as also described in Deliverable D5.1, decision models to concretise 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. Currently, the cloud-based modelling platform ADONIS is used as the FAIRWork process modelling environment. It is a framework based on the ADOxx meta-modelling platform that is BMPN 2.0 compliant. To access the modelling tool, first request ADONIS evaluation credentials from faq@adoxx.org, and then navigate to https://fairwork-sp103940.boccloud.com/main.view#0 or access the login page via the OLIVE Configuration Environment by clicking on the tile with the name "BPMN Modelling". Figure 21 depicts the OLIVE Configuration Environment and the ADONIS login page. How you can navigate in the modelling tool is described in detail in Deliverable D5.1[6]. Additionally, the decision models can be found in the knowledge base under FAIRWork_Project/CRF/Decision models and FAIRWork_Project/Flex/Decision models.

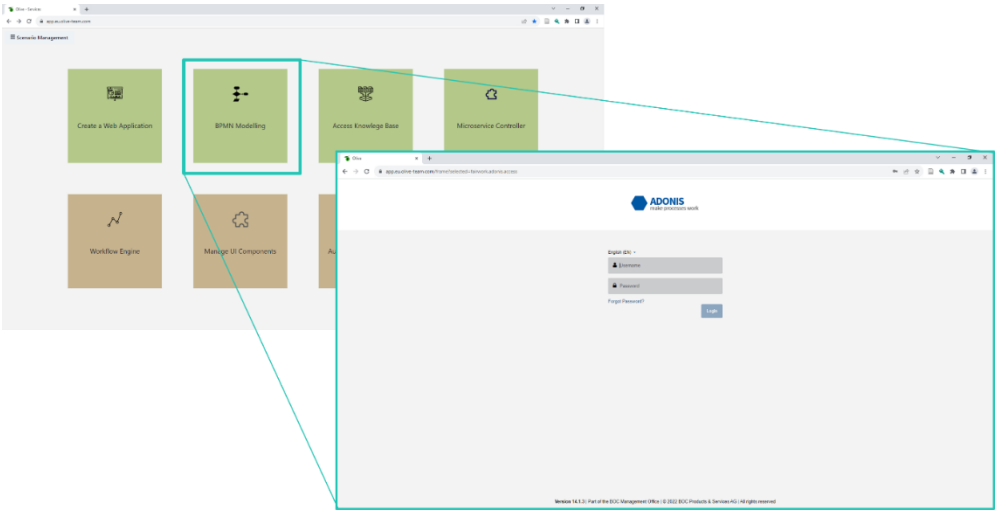

Figure 21: Accessing Login Page of Modelling Tool.

2.4.1.2 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

[6] Chevuri, R. (2023b). FAIRWork Deliverable 5.1 - Demonstrator - FAIRWork Project. https://fairwork-project.eu/deliverables/d5-1

service and how it can be configured to create tailored decision support is described in section 2.6.1. In this section, we want to discuss the procedure and environment that were used to create this experiment for the configuration.

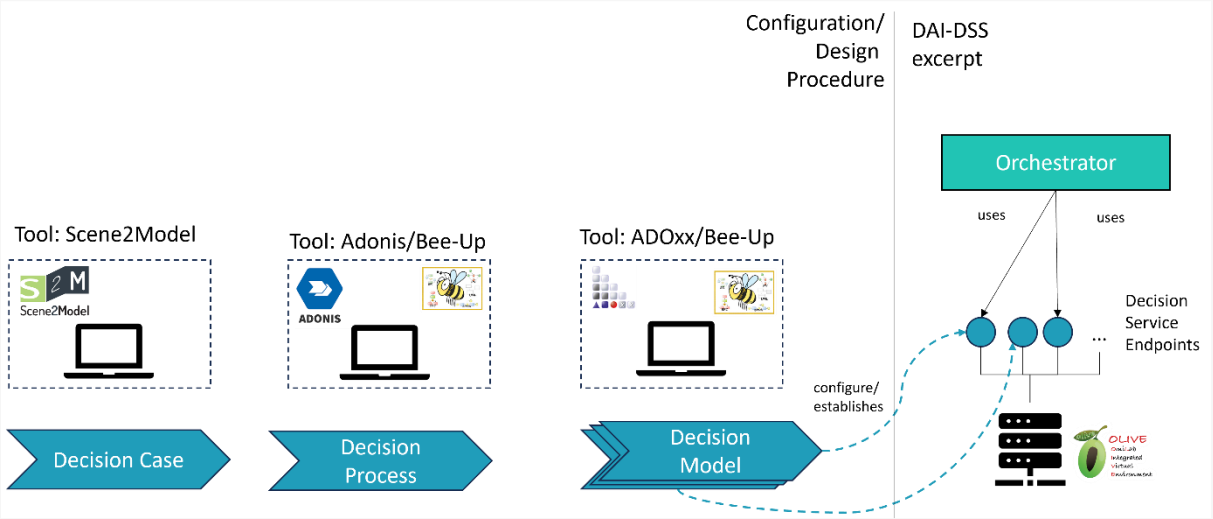

Figure 22 Overview: Experiment for Decision Service Configuration.

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 deliverable 2.1[7] for details). The procedure for this experiment and its connection to the DAI-DSS are visualised in Figure 22. 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 an 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 metamodeling platform; in our case, this was the ADOxx[8] metamodeling 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[9] tool was used, which supports physical workshops to foster the 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 created from the physical artefacts created in the workshop. The created 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 standardised modelling method to start with tested and well-known modelling methods. In particular, *Business Process Model*

[7] Zeiner, H. (2023). Specification of FAIRWork Use Case and DAI-DSS Prototype Report 2.1. https://fairwork-project.eu/deliverables/D2.1_Specification%20of%20FAIRWork-v1.0a-preliminary.pdf
[8] OMiLAB NPO. https://www.adoxx.org/ (last visited: 30-11-2023)
[9] OMiLAB NPO. https://www.omilab.org/activities/scene2model/ (last visited: 30-11-2023)

and Notation (BPMN) and *Decision Model and Notation (DMN)*. Both of these are available in the Bee-Up[10] 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 section 2.6.1.

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 OLIVE[11] microservice framework 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.

2.4.2 Configuration Integration Framework

These decision models have been either refined and converted into a configuration file for an AI service (see section 2.4.1.2) or used as a basis for deriving possible configurations of other components in the system, like the user interfaces or the orchestrator. In the first iteration of the demonstrator, a configuration environment in the form of an OLIVE instance is provided, which gives an example of a collection of different tools to speed up the configuration of the system. The current configuration environment can be seen in Figure 23. Each tile on the web page gives access to configuration wizards or platforms for defining necessary parts of the system, e.g. user interfaces or workflows. The highlighted tiles are currently relevant for the first iteration of the prototype. The tile “Create a Web Application” enables the configuration of a user interface for a web application, while the tiles “BPMN Modelling”, “Access Knowledge Base”, “Microservice Controller” and “Workflow Engine” provide access to other components and applications relevant to the whole system. As this is an initial prototype, the structure and content of this OLIVE instance may change in the course of the project.

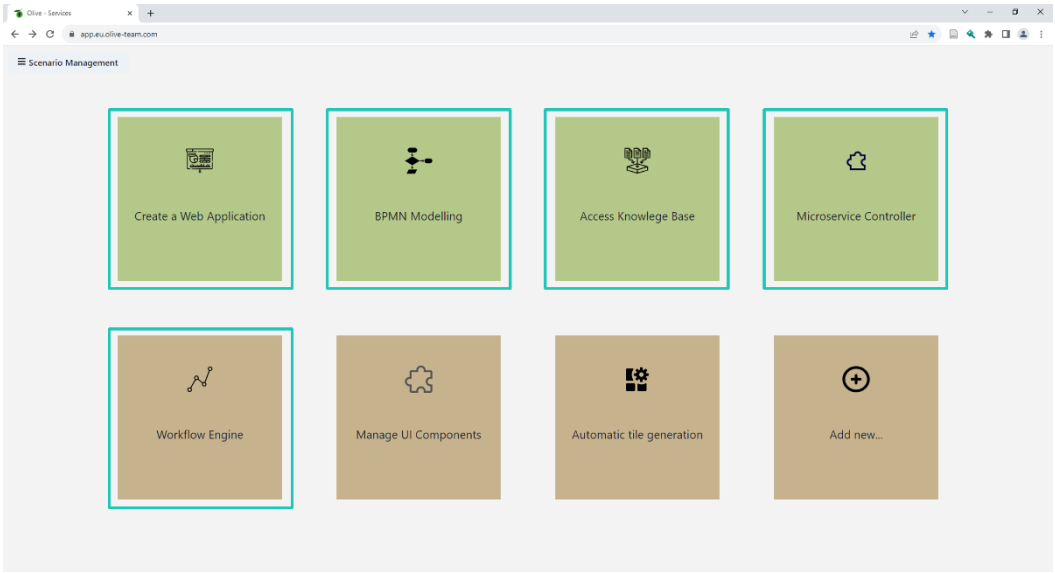

Figure 23: OLIVE instance showing the configuration environment.

[10] OMiLAB NPO. https://bee-up.omilab.org/activities/bee-up/ (last visited: 30-11-2023)
[11] Adoxx.org. https://www.adoxx.org/live/olive (last visited: 30-11-2023)

2.4.2.1 Configuration of services

In the first iteration, a tile for accessing the microservice controller was created in the OLIVE instance. The microservice controller allows to define and manage microservices following a configuration approach. Microservice operations can be defined through the configuration of so-called “connectors”. There are different types of connectors available. For the current version, we used 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. How such a microservice definition can look like can be seen by accessing the microservice controller instance by clicking on the corresponding tile. In the microservice controller dashboard, the user can choose a predefined microservice and click on “Edit”. Now, all currently defined microservice operations and their definitions can be inspected. Depending on the selected connector, different configuration options are available (see Figure 24). 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 2.6.1.

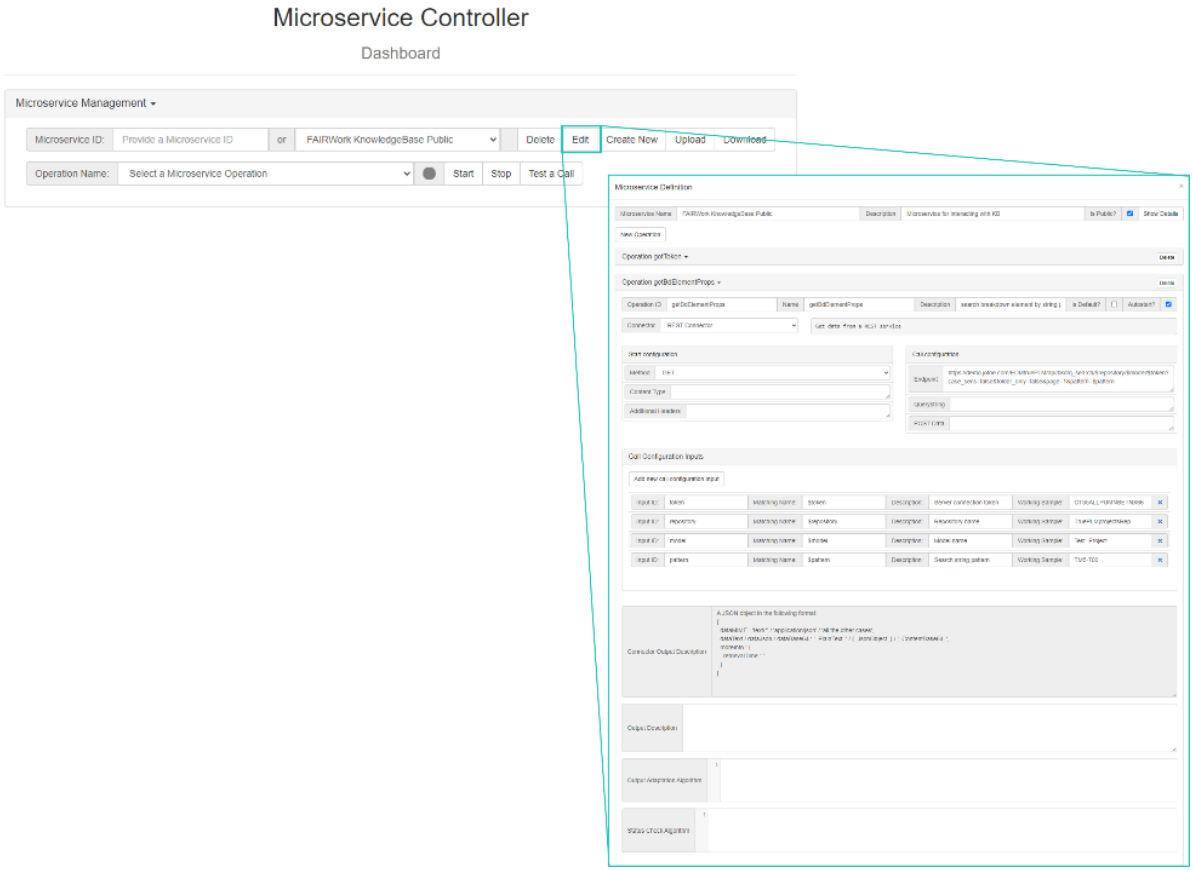

Figure 24: Example of microservice definition.

2.4.2.2 Configuration of workflows

For now, OLIVE allows the management of workflow by providing access to the workflow engine Conductor either through manually accessing it by clicking on the tile named “Workflow Engine” or by using APIs provided by OLIVE. How such a workflow definition can look is described in section 2.3.1 in more detail.

2.4.2.3 Configuration of user interfaces

Besides handling microservices and workflows, OLIVE enables the configuration of user interfaces 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 when required. 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 available components have been uploaded and published first into a component pipeline and are now visible to the user on the left side of the screen. The UI components relevant to this prototype can be found in the component group named “Eu”. Currently, there are three UI components available: “Manual-Allocation”, “Order-Overview,” and “Allocation-Service”. In the initial prototype, the UI components are not dependent on each other; therefore, no wiring between them is needed. If the user is satisfied with the placement of the components, they 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 25 shows the different configuration steps.

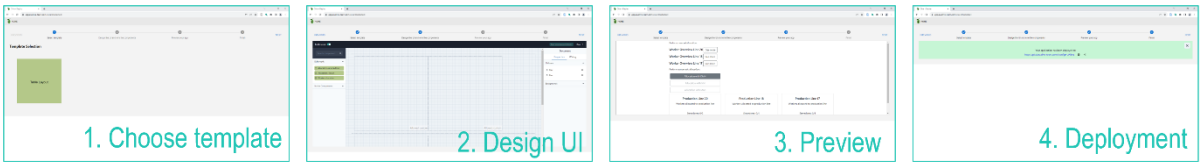

Figure 25: Configuration steps of creating a web application.

In the second iteration, this environment will be extended by providing additional configuration possibilities.

2.5 Integrating the Knowledge Base and Data Flow in the DAI-DSS Architecture

This section provides information on the integration of Knowledge Base with various components in the DAI-DSS Architecture.

The role of the Knowledge Base in the DAI-DSS framework is to store and provide access to relevant data and decision models that are used in the DAI decision support system. It serves as a repository for digital representations of physical assets and stores user-defined properties, sensor data, aggregate data, machine learning results, and other relevant information. The knowledge base is accessed by various components of the DAI-DSS components, such as the DAI-DSS Configurator, to store and retrieve configuration files, decision models, and other necessary data. It also supports the storage and provision of data for training AI models and facilitates the decision-making process by providing relevant information and data to the decision-makers.

Data Storage in the Knowledge Base:

The data required for each scenario is collected and stored in the Knowledge Base based on the decision process models that were described for each use case scenario in deliverable “**D5.1 DAI-DSS FAIRWork Knowledge Base”.** Table 2 shows the data requirements for the CRF use case scenario workload balance below.

Table 2: Required data for the Workload Balance Use Case Scenario.

S.no	Data required	Data source	Format	Frequency	File
1	**Capabilities of operators**	Software (SGA)	XLSX	Weekly	2_Stevedore suitability.xlsx

2	**Availability of operators**	Software (SIPERT)	CSV	Daily (on shift starting)	1_Presence_of operators.csv
3	**Part unload requirement**	Software (SGPP,Siemens)	XLSX	Fixed values	7_Production Cycles.xls, (+ Legende L.X.xls)
4	**Production plan requirements**	Software (SGPP,Siemens)	XLSX	Daily	6_Scheduled orders.xls (+ table_02.xls)
5	**Logistic information regarding (containers, forklifts, storage area)**	Software (SGPP,Siemens)	XLSX	Daily	3_List of Stamping Parts.xlsx, 5_Inventory of container.pdf, 4_Inventory of raw and finished materials.xls

This data is available in the Knowledge Base and can be accessed using REST API methods. Below, Figure 26 and Figure 27 show the snapshot of data stored in the private Knowledge Base and the API method for retrieving the data and corresponding response in JSON format.

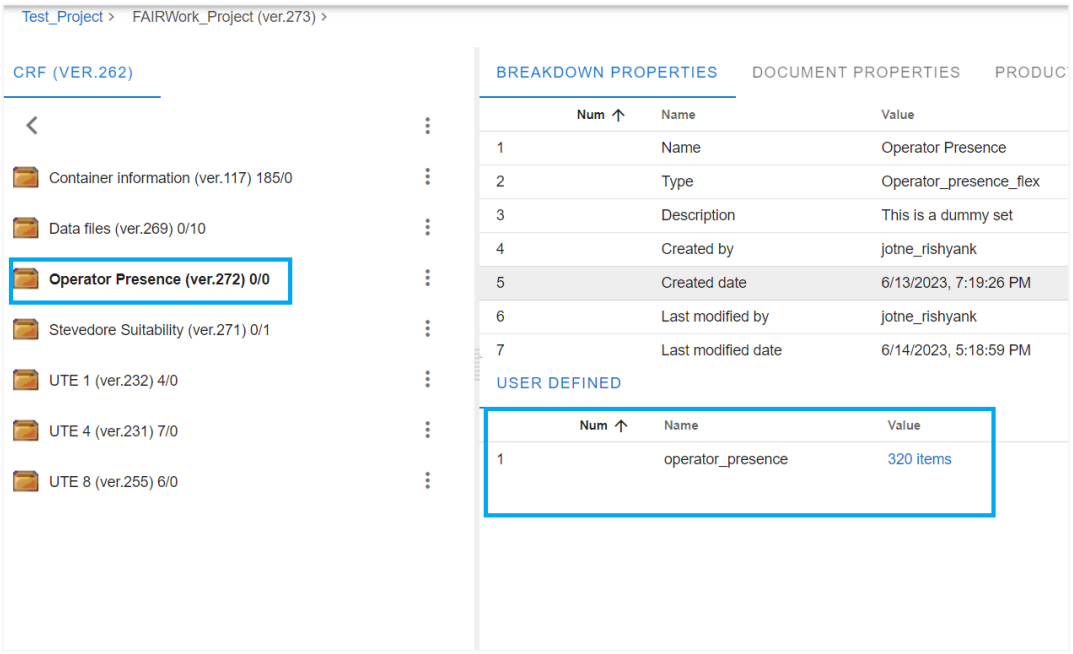

Figure 26: Operator Presence information stored in the Knowledge Base.

```
Performing request: https://fairwork.jotne.com/EDMtruePLM//api/bkd/aggr_exp_dt/TruePLMprojectsRep/Test_Project/261993276944/urn:rdl:Test_Project:operator_presence/6O9XN289G3FGX1OK0P
        Request (status code = 200)
[{'Index': '100000001', 'Timestamp': '2023-04-11 06:31:04', 'ID': '10001'}, {'Index': '100000002', 'Timestamp': '2023-04-12 06:31:04', 'ID': '10002'}, {'Index': '100000003', 'Timestamp': '2023-04-13 06:31:04', 'ID': '10003'}, {'Index': '100000004', 'Timestamp': '2023-04-14 06:31:04', 'ID': '10004'}, {'Index': '100000005', 'Timestamp': '2023-04-15 06:31:04', 'ID': '10005'}, {'Index': '100000006', 'Timestamp': '2023-04-16 06:31:04', 'ID': '10006'}, {'Index': '100000007', 'Timestamp': '2023-04-17 06:31:04', 'ID': '10007'}, {'Index': '100000008', 'Timestamp': '2023-04-18 06:31:04', 'ID': '10008'}, {'Index': '100000009', 'Timestamp': '2023-04-11 06:31:04', 'ID': '10009'}, {'Index': '100000010', 'Timestamp': '2023-04-11 06:31:04', 'ID': '10010'}, {'Index': '100000011', 'Timestamp': '2023-04-11 06:31:04', 'ID': '10011'}, {'Index': '100000012', 'Timestamp': '2023-04-11 06:31:04', 'ID': '10012'}, {'Index': '100000013', 'Timestamp': '2023-04-11 06:31:04', 'ID': '10013'}, {'Index': '100000014', 'Timestamp': '2023-04-11 06:31:04', 'ID': '10014'}, {'Index': '100000015', 'Timestamp': '2023-04-11 06:31:04', 'ID':
```


Figure 27: REST API JSON Response of Operator Presence.

The data from the Knowledge Base is made available using the DAI-DSS Configurator, which utilises the OLIVE Microservice Framework. The OLIVE Framework allows the creation of model-aware web applications by configuring existing components, effectively serving as a bridge between the Knowledge Base and other DAI-DSS components. It uses the Knowledge Base's REST API functions to extract required information. A key component of the OLIVE Framework is the Microservice Controller, which enables the definition of backend microservices. These defined services can then be used within the configuration environment to incorporate data from the Knowledge Base into web applications. Based on the requirements, these configured services are then integrated into the DAI-DSS Orchestrator as a standalone service or workflow-based orchestration coupled with a series of microservices.

2.5.1 Data flow in DAI-DSS

In this subsection, the key components of the DAI-DSS architecture are depicted (see Figure 28), depicting the flow of data and decision support for the use case scenario. The central element in this process is the configurator, which initiates the sequence. It uses decision process models for aligning user interfaces and AI services with the necessary data from the Knowledge Base. This configured information is then sent to the Orchestrator as a workflow. The user interface plays a crucial role by providing means to invoke the required AI services. When a user submits an input request, the corresponding workflow is executed. This workflow, in turn, triggers the execution of AI services or multi-agent-based services, essential for decision support.

Moreover, if the invoked AI services require additional data from the Knowledge Base, this data is obtained via REST API services provided by the Knowledge Base. Based on this data, the AI services generate suggestions crucial for user decision-making. The results are then presented back to the user through interfaces, often accompanied by appropriate graphics. Throughout this entire process, interactions between components occur through REST API and JSON format, ensuring a standardised flow of data.

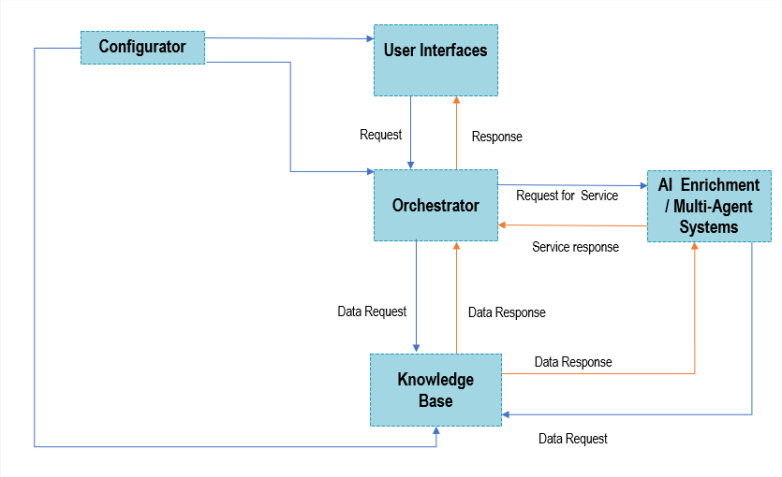

Figure 28: Data Flow in DAI-DSS Architecture.

Figure 29 presents the other part of DAI-DSS architecture, which presents the data flow from the end-user environment and the Intelligent Sensor Box into the Knowledge Base.

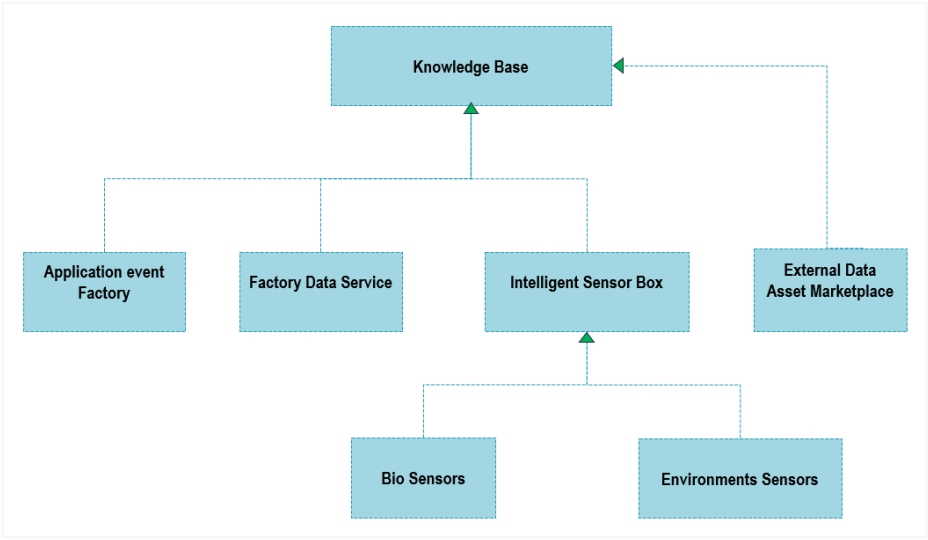

Figure 29: Data Flow in DAI-DSS Architecture.

2.6 Integrating AI-Services

To facilitate the seamless incorporation of AI capabilities, all developed AI services are made accessible through REST-API endpoints, which are currently considered an industry standard. This practice ensures that various clients can effortlessly tap into the functionality of these services through HTTP requests.

Adherence to the Swagger documentation standard for defining the API endpoints is a pivotal component of this standardised integration. Swagger provides comprehensive and human-readable documentation of the available services and establishes a consistent framework for developers to understand the functionalities offered by each endpoint. The significance of this standardisation extends to the format of the HTTP requests themselves. JSON-formatted data in these requests ensures a well-defined structure for exchanging information between clients and AI services. With its simplicity and widespread support, JSON is a defacto standard for communication in large software systems, enhancing interoperability and easing the burden on developers.

Furthermore, the integration process can be enhanced by employing code generation tools such as Swagger Editor. These tools empower developers by automatically generating client-side code tailored to the AI services' API specifications. This expedites the development process and mitigates the likelihood of errors that may arise from manual coding, fostering a more robust and reliable integration.

Overall, integrating AI services via REST-API endpoints, following industry standards and Swagger documentation, coupled with JSON-formatted data and code generation tools, represents a harmonised and streamlined approach. This not only ensures the accessibility and comprehensibility of AI functionalities but also simplifies the development process, making AI integration more accessible and efficient for a diverse range of clients and applications.

2.6.1 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 worker allocation use case, which belongs to the resource mapping decision problem, which is introduced in FAIRWork's deliverable 2.1[7].

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 section 2.4.1, 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[11] 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.

2.6.1.1 Prerequisite

To prepare a new decision endpoint within this experiment, three parts must be prepared. One is the Bee-Up modeling 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 first 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 in section 2.6.1.3).

To fully use the implemented decision service for this use case, an additional service is needed, which is called *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 the resource allocation.

2.6.1.2 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 workflow orchestrator described in section 2.3.1 and the user interface described in section 2.2.

An example of the OLIVE user interface can be seen in Figure 30. 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.

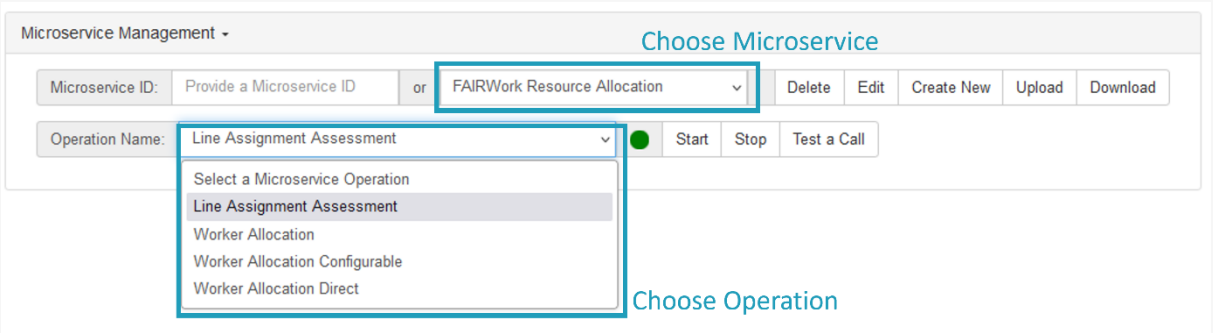

Figure 30: OLIVE User Interface for Resource Allocation Experiment.

For testing the service defined with the model, the *Line Assignment Assessment* operation must be chosen (see Figure 31). 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
- UTEExperience: "True" and "False"

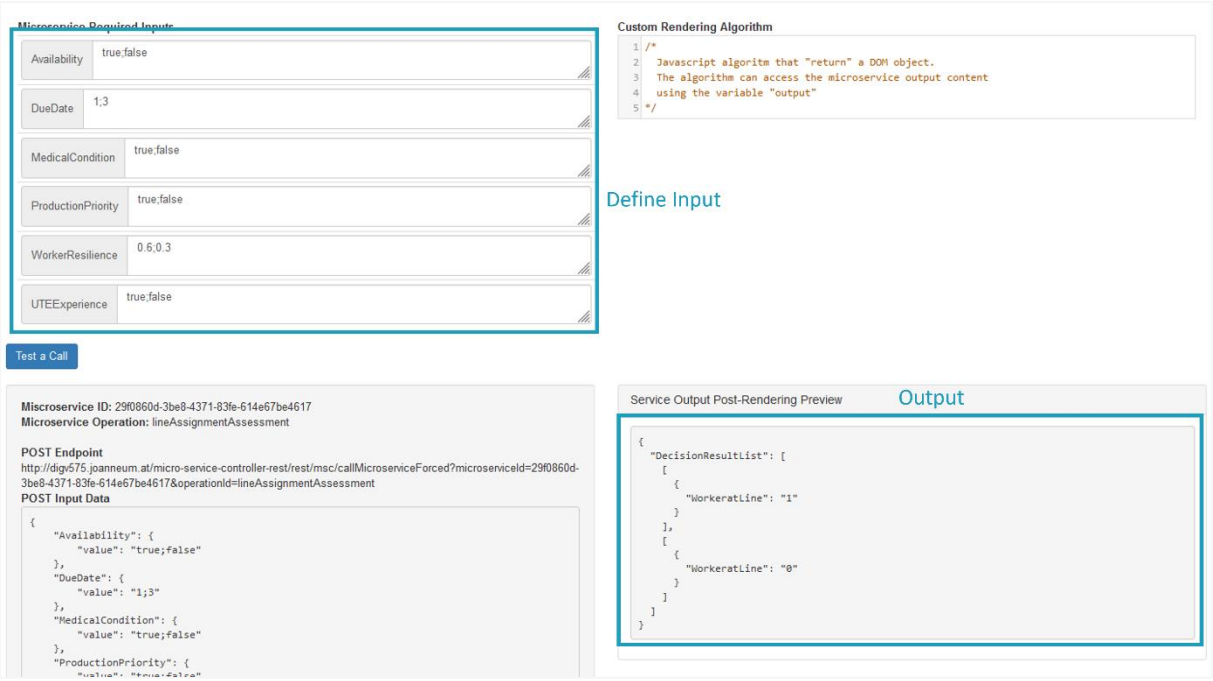

Figure 31: OLIVE Interface for Testing the Line Assignment Assessment Operation.

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.

2.6.1.3 Creating a New Decision Endpoint with Conceptual Modeling

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 make the decision 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 32. 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.

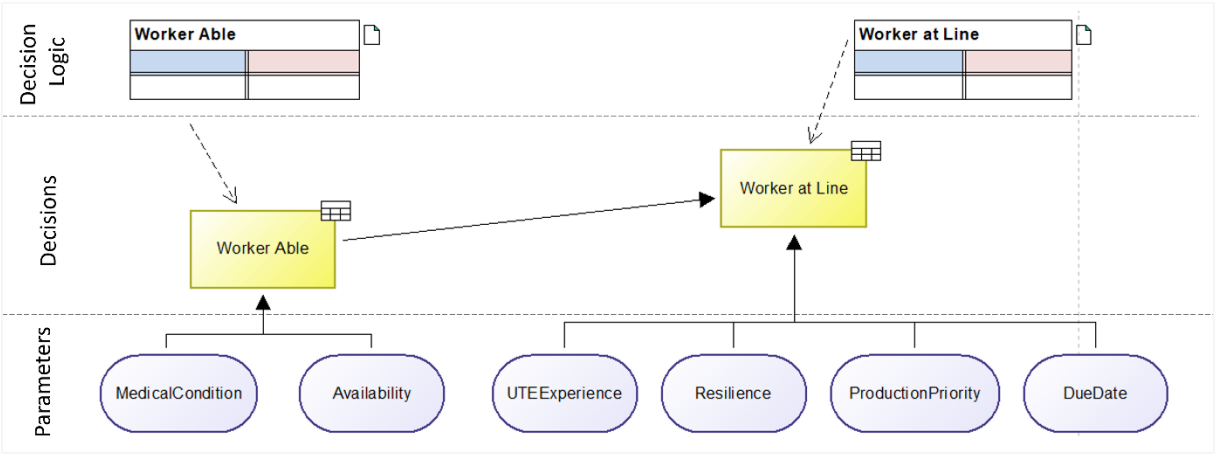

Figure 32: DMN Model for the Worker Allocation Use Case.

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 32), 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[12], 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. Afterwards, the decision rules can be added; an example is shown Figure 33. 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 *[<star>..<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	UTEExperience	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 33: Example decision table.

[12] Object Management Group. (2023). Decision Model and Notation (DMN).

After the model is completely defined, one can create the DMN file, which can be used as the input for creating the decision endpoint. Therefore, 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 34. 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.

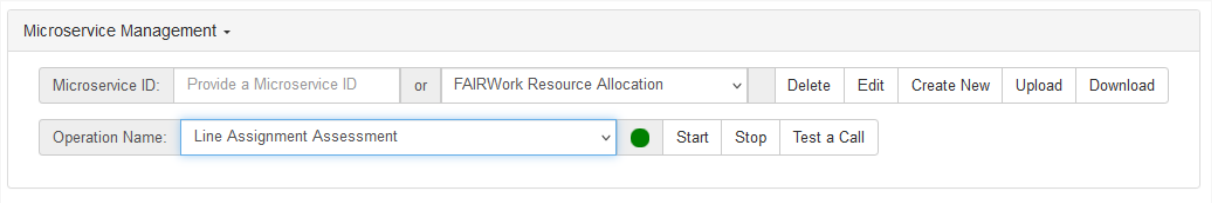

Figure 34: OLIVE web interface.

The interface for adding a decision endpoint can be seen in Figure 35Figure 35. 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.* Afterwards, this configuration must be closed by using the *Continue* button on the end, and then the rest of the endpoint can be used.

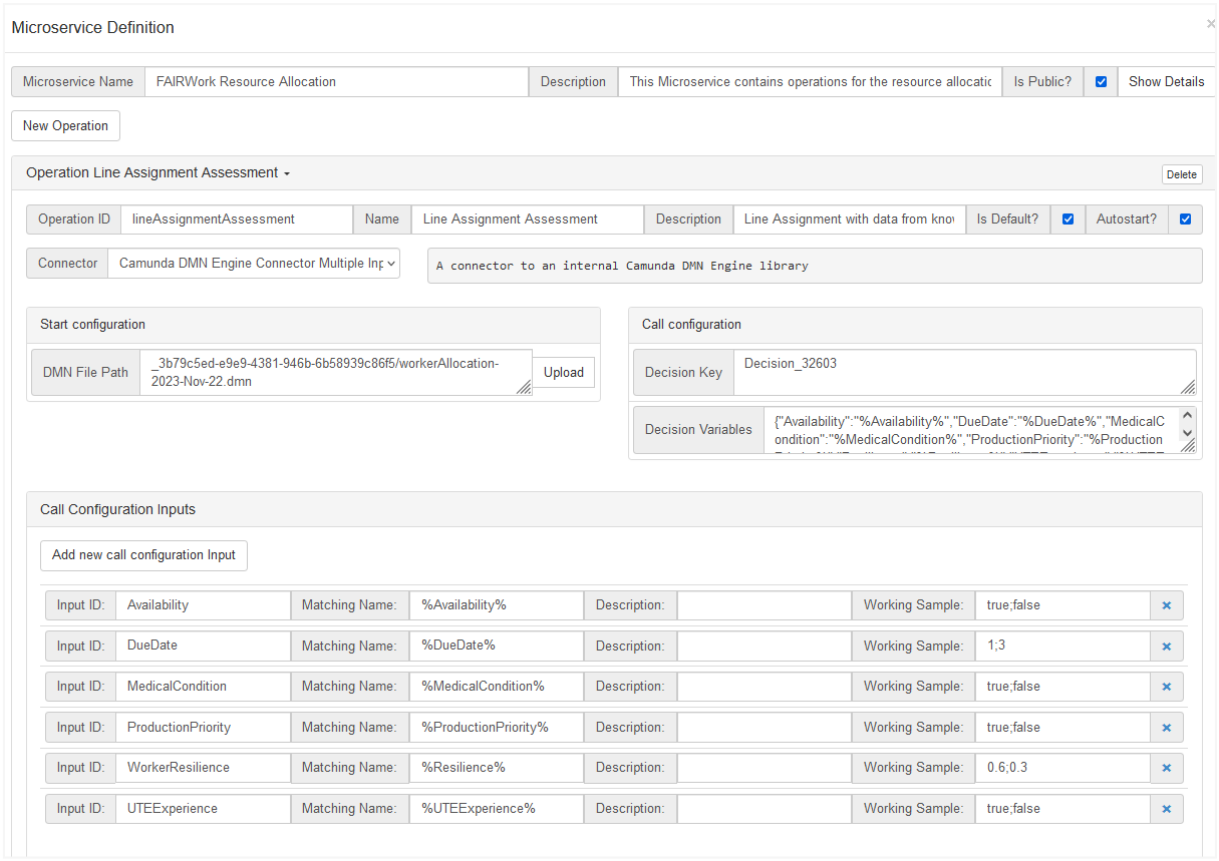

Figure 35: OLIVE Microservice Configuration Interface.

The OLIVE interface also has a test interface, which can be opened by choosing your microservice and your operation in the corresponding drop-down menus of the basic interface (see Figure 36Figure 34), and then click *Test a Call.*

This opens the windows shown in Figure 35. Here, values for the defined parameters can be defined, and with the *Text 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.

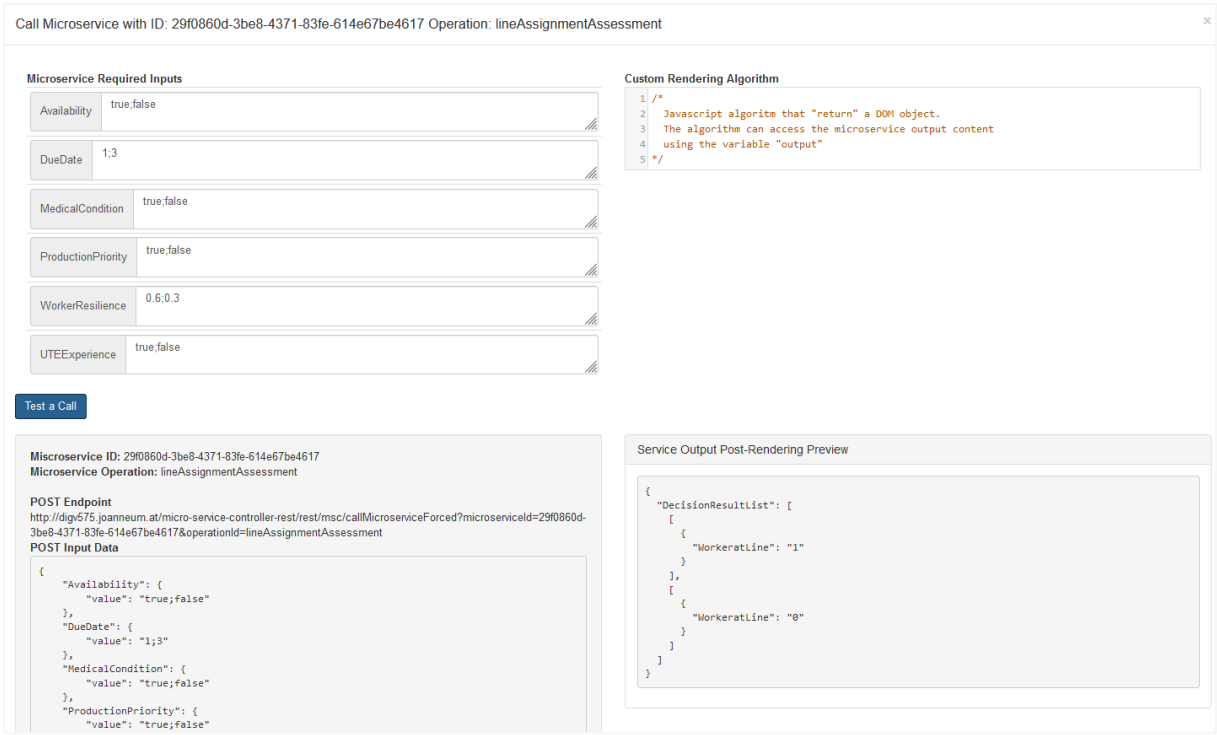

Figure 36: OLIVE Test Interface.

2.6.2 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 2.4.1. The difference to the service introduced in section 2.6.1 (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 comprehendible 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 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[10] 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. This extension 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 37 provides the steps on how to trigger the functionality.

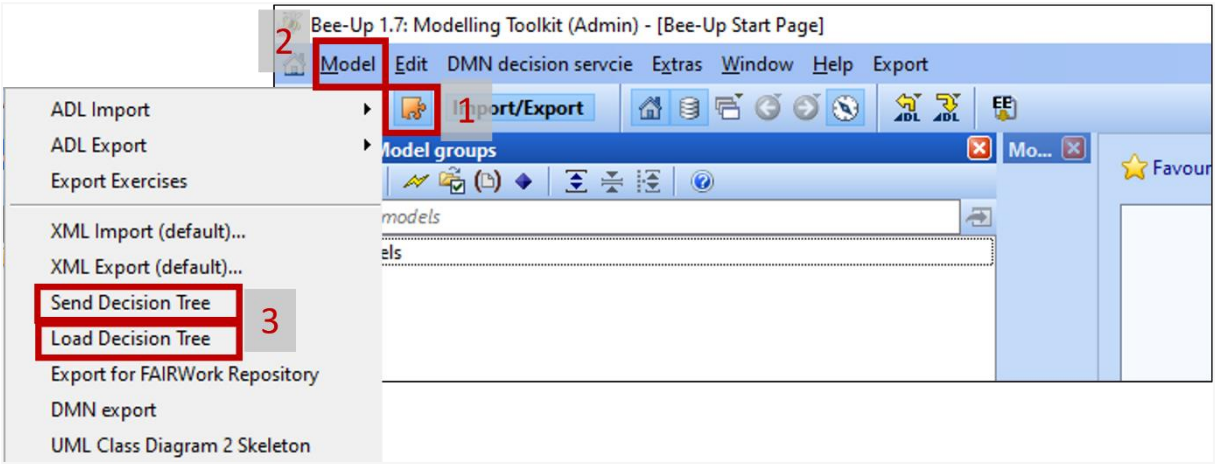

Figure 37: Triggering Decision Tree Functionality in the Modeling Tool.

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 38. 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 2.4.1, 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 38 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 2.6.1 can be used.

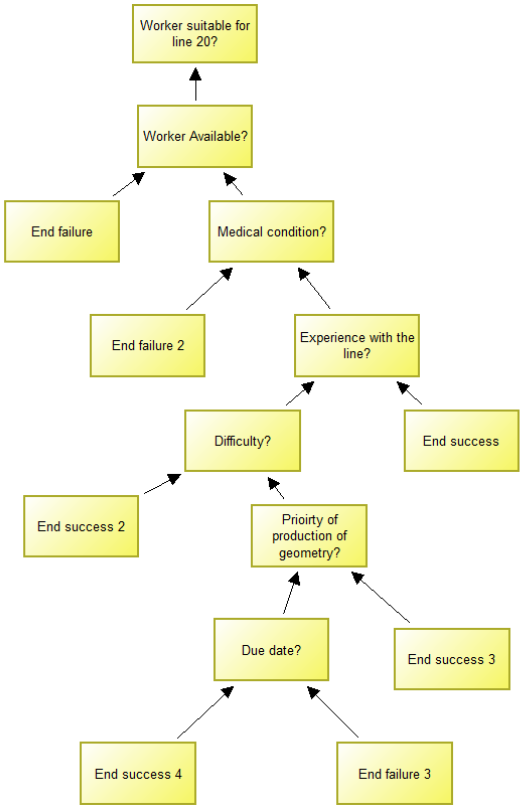

Figure 38: Example of Decision Tree in the Modelling Tool.

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

2.6.3 Resource Allocation using Neural Networks

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 optimising 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 39 and Figure 40). 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.

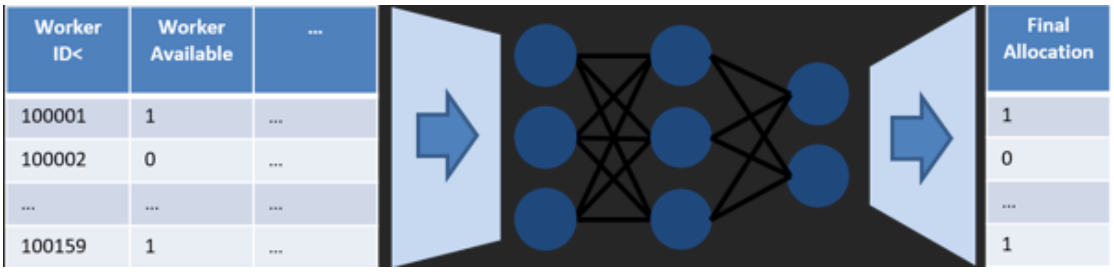

Figure 39: Neural Network Resource Allocation.

The data structure used in the demonstrator can visualized in tabular format (see Figure 40). 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 user interface 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.

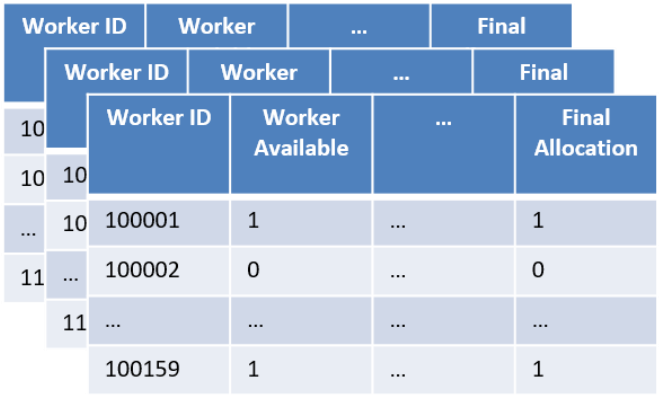

Figure 40: Historic data.

The training process involves adjusting the network's weights and biases to minimise the disparity between the predicted "final allocation" and the actual values in the training set.

Implementing a neural network for worker-machine allocation offers an automated and efficient alternative to manual allocation methods. By leveraging historical data, the neural network can learn and replicate the decision-

making strategies of experienced employees, providing a valuable tool for optimising the allocation process in manufacturing industries.

2.6.4 Resource Allocation using Linear Sum Assignment Solver

In essence, the underlying problem covered by the demonstrator is, at its core, an assignment problem. 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 minimise the total cost of the assignment.

Assignment Problems are a generic class of Problems, and Optimization Libraries like Google OR-Tools have specialised 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 visualised in Figure 41.

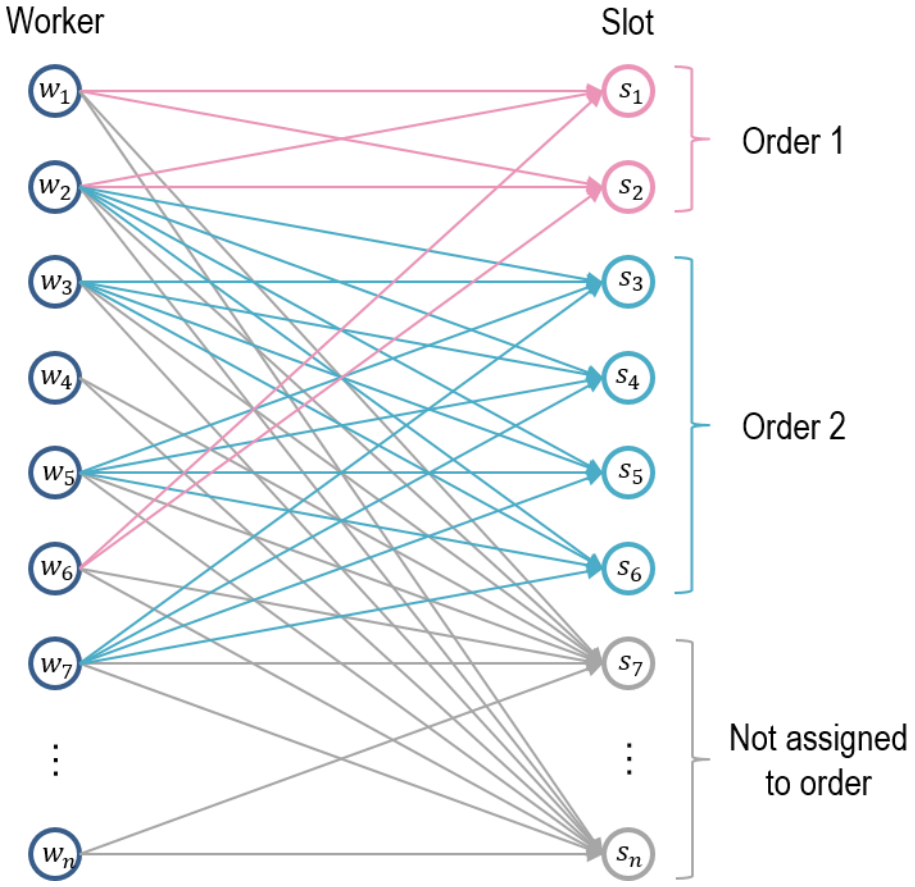

Figure 41: Example Problem Instance in Linear Sum Assigment Graph Representation.

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 41, order one introduces two slots; order two introduces 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, optimising the cost still corresponds to optimising the assignment. The not-assigned slot weight offsets the total cost, which does not influence the cost optimisation. The cost of assigning a Worker to a slot can be organised in a matrix. The example instance results in a matrix (see Table 3).

Table 3: Cost matrix.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	$\cdots$	s_n
w_1	$f_c(w_1, s_1)$	$f_c(w_1, s_2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$
w_2	$f_c(w_2, s_1)$	$f_c(w_2, s_2)$	$f_c(w_2, s_3)$	$f_c(w_2, s_4)$	$f_c(w_2, s_5)$	$f_c(w_2, s_6)$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$
w_3	$\emptyset$	$\emptyset$	$f_c(w_3, s_3)$	$f_c(w_3, s_4)$	$f_c(w_3, s_5)$	$f_c(w_3, s_6)$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$
w_4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$
w_5	$\emptyset$	$\emptyset$	$f_c(w_5, s_3)$	$f_c(w_5, s_4)$	$f_c(w_5, s_5)$	$f_c(w_5, s_6)$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$
w_6	$f_c(w_6, s_1)$	$f_c(w_6, s_2)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$
w_7	$\emptyset$	$\emptyset$	$f_c(w_7, s_3)$	$f_c(w_7, s_4)$	$f_c(w_7, s_5)$	$f_c(w_7, s_6)$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
w_n	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\sup(f_c)$	$\cdots$	$\sup(f_c)$

This matrix represents an Adjacency matrix, essentially encoding the cost associated with assigning a Worker to a specific slot. If the entry is 0, it signifies the absence of a connection between the Worker and the slot.

In the provided example, Worker w1 is constrained to working only on Order 1, specifically on slots s1 and s2, and is not available for Order 2 (slots s3 to s6). Conversely, for Worker w3, the situation is reversed; they are available for both Order 1 and Order 2. Worker w4 is not present in the example, indicating a lack of availability for both Order 1 and Order 2, and consequently, no edges exist in the matrix for this worker.

Internally, the Linear Assignment Solver uses an algorithm by Goldberg and Kennedy presented in the paper "An efficient cost scaling algorithm for the assignment problem" (1995). The run time of this algorithm is O(n*m*log(nC)), where n is the number of nodes, m is the number of edges, and C is the largest magnitude of an edge cost. An initial implementation of the Linear Sum Assignment Solver is provided on GitHub. As a next step, it will be wrapped as a REST-API and deployed on a server to be integrated into the overall system.

2.6.5 Resource Allocation MAS-based

In the evolving landscape of the manufacturing industry, the advent of Multi-Agent Systems (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 this initial 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.

This solution was modelled on the basis of a business model that identified the stakeholders. In the Multi-Agent System, these stakeholders were taken into account, and an interaction strategy was chosen based on the overall objective of this application. The system was developed using the JADE[13] 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.

During the modelling, 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 42). 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, 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 43 and Figure 44).

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 to the particular line.

This application of Multi-Agent Systems 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 prioritises factors like worker resilience and personal preferences, ensuring a more human-centric approach to industrial operations.

Key to this system's functionality is the business model, which effectively identifies and incorporates the roles and requirements of different stakeholders. Utilising the JADE framework for its development, the system leverages the power of Object-Oriented Programming in Java, making it highly adaptable for use in a variety of industrial settings. The choice of this framework is particularly beneficial due to its capabilities to facilitate communication and interaction between various agents, a feature essential for the dynamic and often complex landscape of manufacturing industries.

When it comes to the practical application of this system, its focus is primarily on two main stakeholders: the workers and the production lines. The system considers a range of parameters that consider the well-being and suitability of the workers. The essence of this prototype lies in its algorithm's explainability and its focus on adaptability and scalability. The algorithm evaluates a pool of workers, assigning scores based on resilience and personal

[13] JAVA Agent DEvelopment Framework. (2023). https://jade.tilab.com/

preferences for specific production lines. This scoring system, which currently emphasises resilience slightly more heavily, allows for a ranked recommendation of workers for each line, ensuring an optimal match between the worker's capabilities and the line's requirements. The AI service is designed to handle multiple changing scenarios across various production lines within a factory. It dynamically adjusts to the number of available agents, thus exhibiting flexibility and scalability that are crucial in a modern manufacturing environment. This holistic approach not only enhances operational efficiency but also fosters a more human-centric.

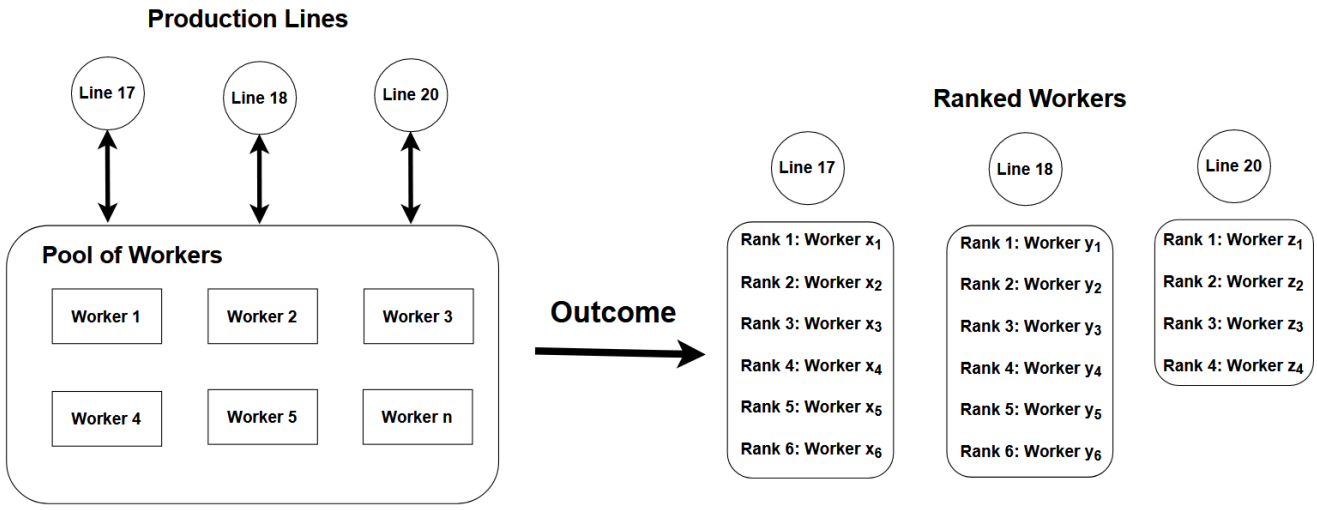

Figure 42: MAS-based Worker Allocation dynamic.

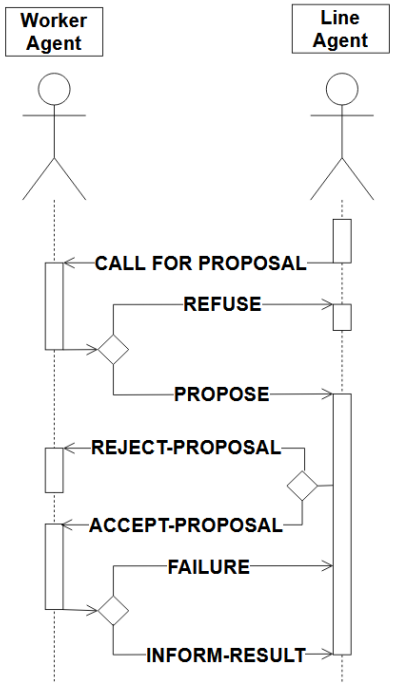

Figure 43: UML Sequence Diagram of the Multi-Agent-based Workload Balance.

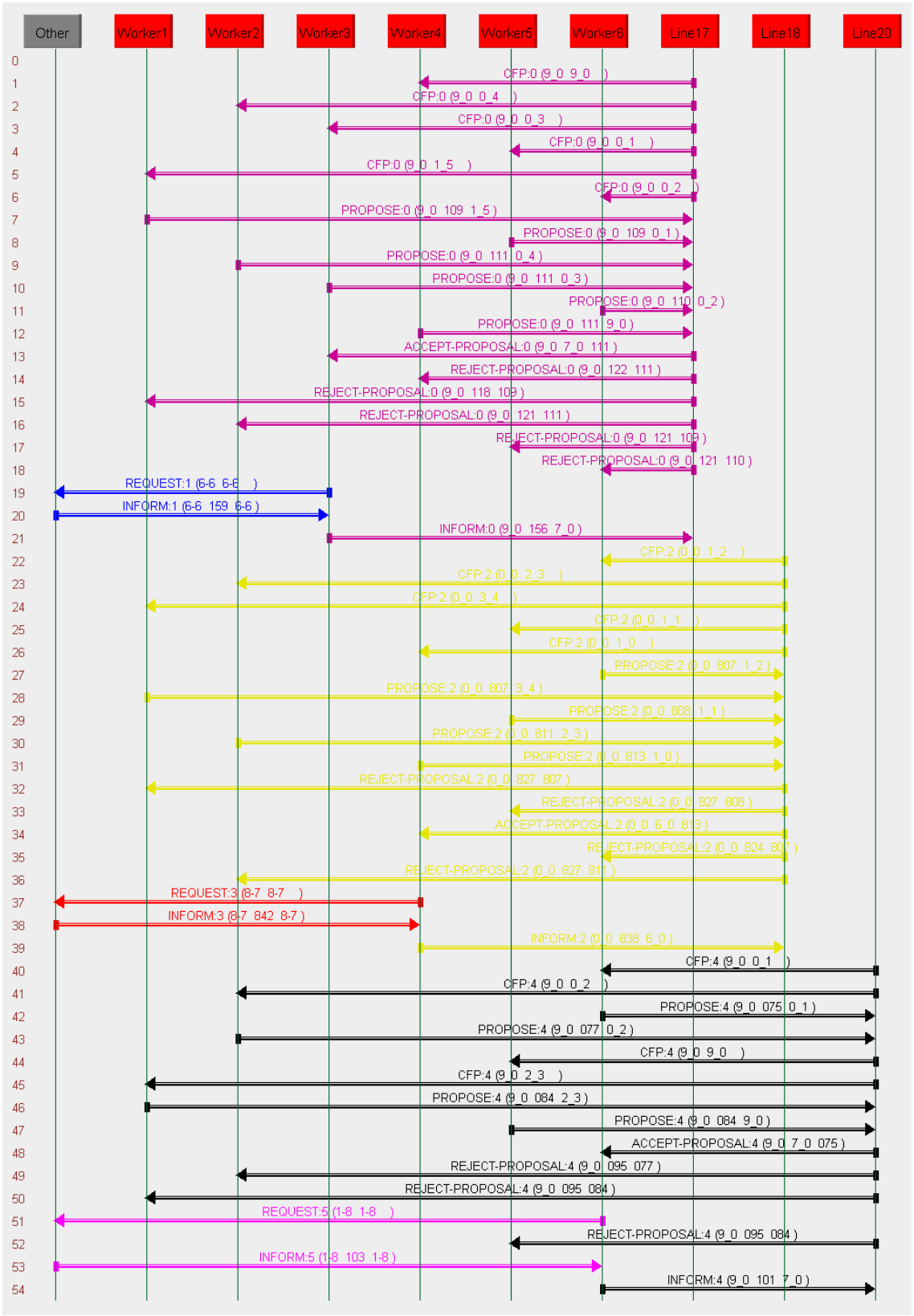

Figure 44: Worker Allocation agent message exchange example.

2.7 Real World Data Providers

2.7.1 Intelligent Sensor Boxes

The initial FAIRWork Knowledge Base, as described in D4.1, was based on an independent data source, accessible via microservices, with representative dummy data for test purposes. Now, deliverable D4.2 covers the first iteration of the demonstrator providing an integrated data lake, where legacy systems and also Intelligent Sensor Boxes (ISB) will deliver necessary information into the data lake.

The Intelligent Sensor Box (ISB) (see Figure 45) enables the measurement of a worker's physiological and psychological stress while doing tasks and provides information about the worker's estimated resilience. It consists of a framework for a set of stationary and wearable sensors and AI-based analytics for assessment and optimisation functions. It can be applied to evaluate the ergonomics and design of industrial training and work environments.

Figure 45 depicts the system architecture of the proposed Intelligent Sensor Box. The Local Wearable Sensor Network collects bio-signals from the wearables that are attached to the workers and the Local Workplace Sensor Network records. The gathered data streams are stored in the Local Database with the aid of the Local Data Receiver and Local Data Management components. The local Decision Support System (ISB-DSS) generates higher abstractions of human factors and accesses Human Factors Intelligent Services for this purpose. The internal Human Factors Intelligent Services are also used by the user interfaces, providing a worker their human factors state. Additionally, the Intelligent Sensor Box has a Developer Dashboard to display various bio signals, human factors parameters and current system state. The higher-level DAI-DSS Knowledge Base can access all necessary data from the Intelligent Sensor Box via the dedicated REST API. To ensure data protection, all provided data is anonymised or at least pseudonymised by the Data Anonymization component.

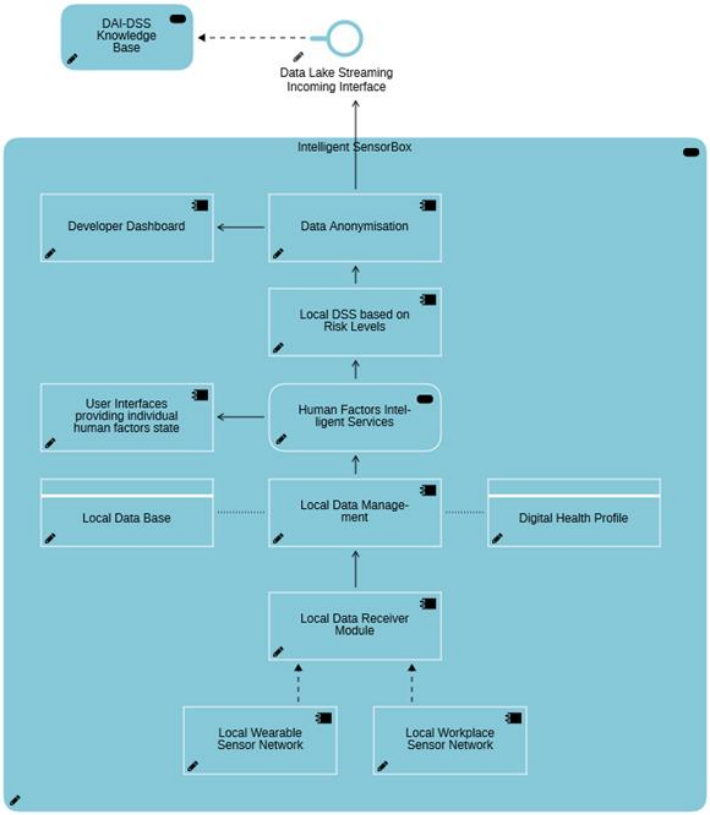

Figure 45: System Architecture of DAI-DSS Intelligent Sensor Box

Resilience Score

The ISB-DSS (see Figure 46) provides a **resilience score** as a service for the consideration of further worker allocation. This resilience score represents the current capacity of a worker to bear the load in terms of physiological or cognitive-emotional strain without negative long-term impact. In other words, a worker with a high resilience score is capable of operating for some time in more stressful environments and more demanding tasks than a worker with low capacity in this respect.

Physiological strain. In the current version of the demonstrator, the resilience score essentially incorporates a measure integrating the physiological strain of a worker based on the "estimated" Physiological Strain-Index (PSI [1]) of its recent experience at the workplace. We consider incorporating a "history of resilience" of the last previous *n=20* working days and integrating it into a heuristic estimate of how the capacity would be on the resulting, i.e., a working day following the other previous 20 working days. Each day, it will, therefore, be assigned a representative resilience score, and the resulting resilience score is computed by a linear combination of the 20 previous resilience scores, by

$$\bar{R}(\mathrm{d}) = \left(\sum_{n=1}^{20} w_{-n}\, R_{d-n}\right) \qquad \text{(Eq. 1)}$$

We assume that there are estimated PSI scores for every single one of the past *n=20* days. For the estimation of the overall PSI that will configure each day's resilience score, we will first determine the estimated PSI score about every *10* seconds (6 times a minute). The PSI is estimated based on the measured skin temperature and heat flux (sensed by a biosensor that is laterally attached to the chest) and the heart rate of the worker (being measured by either a biosensor shirt or an intelligent armband). To increase the robustness against outliers, the individual PSI values are averaged over a 1-minute time window.

From strain to resilience. For the concrete resilience calculation, we implement and apply a functionality (within component ISB-DSS, see Figure 45) that integrates quantities of PSI over time and, in this sense, would eventually provide a measure of resulting intensive fatigue experienced by the worker on that specific day that would have a mid or long-term impact on the resilience of the worker. For the resilience calculation, the ISB-DSS measures the time a worker spends in a specific PSI range and weights the sum by a range-specific intensity factor. The PSI, ranging from 0 (no strain) to 10 (maximum individual strain), is therefore divided into 5 zones [0-2], (2-4], (4-6], (6-8] and (8-10]. The weighted sum is normalised in relation to an eight-hour shift, and then the inversely proportional resilience score ranging in the interval of [0,1] would be determined for that specific day R_d. Now, the final overall resilience score $\bar{R}$(d) (see Eq. 1) based on the weighted average of the last n days in the past is determined. The weighting currently carried out uses an exponential decay function, which assigns the most recent days more impact than those from longer ago. Figure 46 exemplarily shows some captured biosignals and the derived PSI scores during physically challenging laboratory exercises. The top diagram depicts the course of the heart rate. The middle diagram shows the course of skin and core body temperature, and the bottom diagram shows the resulting PSI scores (based on skin temperature and core temperature). The different colours (green, light green, yellow, orange and red) in the lower diagram correspond to the 5 defined PSI zones for rating the strain.

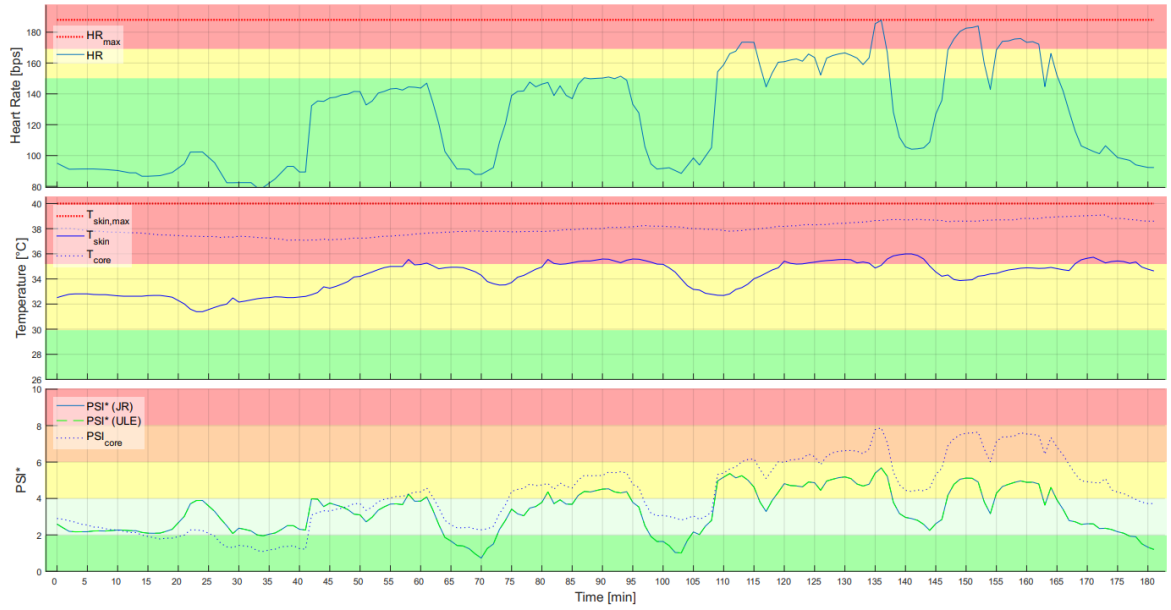

Figure 46: Exemplary course of heart rate (top diagram), skin and core body temperature (middle diagram) and PSI/PSI* scores (bottom diagram) during physically challenging laboratory exercises. The different colours (green, light green, yellow, orange and red) of the bottom diagram correspond to the 5 defined PSI zones for rating the strain. [2]

Resilience score table. The ISB stores and provides the resilience score information in a table that can be accessed via a REST interface. The table consists of m rows, each representing a worker with a unique ID. The ID is stored in the first column, the next 20 columns represent the resilience scores of the past $n=20$ days and the current weighted resilience score $\bar{R}(d)$ is stored in the last column. The table is updated after each day. Table 4 gives an overview of the resilience score table design.

Table 4: Description of resilience score table.

Variable	Field name	Type / Normal Range	NaN Value	Description
UID	user_id	string / -	""	User Identifier, unique identifier of a worker e.g. FW05
res_20	resilience_20	float [0,1]	-1.0	resilience score R_d of the 20th day in the past
…	…	…	…	…
res_01	resilience_01	float [0,1]	-1.0	yesterday's resilience score R_d
res	resilience_overall	float [0,1]		current resilience score as a weighted sum of the resilience scores of the last 20 days

Interface

The resilience table within the database can be accessed via a REST API call. The following methods will be supported:

getCurrentResilienceScore (user_id=string OR persona=string, session_id=None)

Returns the latest resilience score of a specific user or a persona type.

Input	user_id: string OR persona: string, optionally session_id for further filtering
Output	single integer value

getCurrentGroupedResilienceScore (group=string OR affiliation=string, session_id=None)

Returns the latest resilience scores of users assigned to a specific group or affiliation.

Input	group: string OR affiliation: string , optionally session_id for further filtering
Output	dictionary: keys: string, such as “user_id1”,…,”user_idn” values: single double values

getLastResilienceScores (user_id=string OR persona=string, session_id=None)

Returns metadata and the average of the hourly resilience score of the last 20 days of a specific user.

Input	user_id: string OR persona: string, optionally session_id for further filtering
Output	dictionary: keys: string, such as “resilience_1”, …, ”resilience_overall”, ”last_updated” values: single double values

getLastGroupedResilienceScores(group=string OR affiliation=string, session_id=None)

Returns metadata and the average of the hourly resilience score of the last 20 days of all users assigned to a specific group or affiliation.

Input	group: string OR affiliation: string, optionally session_id for further filtering
Output	dictionary: key: string, such as “user_id”/”persona” value: dictionary keys: string, such as “resilience_1”, …, ”resilience_overall”, ”last_updated” values: single double values

In the near future, we will also provide methods that directly push the latest available resilience information forward to the dedicated endpoint into the Knowledge Base of the joint DAI-DSS of FAIRWork. This would provide direct access via the Knowledge Base from any service in the DAI-DSS system.

Discussion and outlook

The current implementation of data about Human Factors in terms of a worker's resilience score provides a simplified but proof-of-concept model for considering human-centred decision support. At this time of the demonstrator development, we are focusing exclusively on physiological strain aspects. In a next development step, we will also add a consideration of the cognitive strain that is experienced at the workplace, for example, by decision-makers, and that would have an impact on the worker's resilience by means of mental fatigue. For this purpose, the Cognitive-Emotional Stress (CES) score will be integrated into an overall consideration of the workers' resilience. This will include additional sensor-based data, such as data from eye-tracking glasses and the derived innovative metrics on eye movement features.

In order to enable a fast kickoff for further processing by other demonstrator components, we will first generate artificial data to fill them into a "first-version" resilience score table. In a next step, we intend to provide data directly derived from measured biosensor data in specifically prepared sessions in the Human Factors Lab. We will simulate activities that are characteristic of the ones that are actually experienced by the workers in the workplace of the manufacturing partners. These tasks will be either physiologically demanding or cognitively challenging. We will measure the biosensor data for a range of test subjects and apply the computations to result in resilience scores. Even if we only recorded a few typical load profiles on a random basis, we would still be able to provide estimates for real activities and at least cover resilience scores for several days within the 20-column score table. This would be sufficient to provide a proof of concept in this actual demonstrator version.

For the application within the real-world conditions of the manufacturing companies' workplace, we certainly would have to provide several additional components. For example, we simplified any calendar-based implementation by assuming that there are 20 days of continuous working days, neglecting the existence of weekends, sick leaves, or vacations, etc. In this sense, our demonstrator version currently projects into a simplified use case that nevertheless provides results about the most important functionalities of strain and resilience. This calendar functionality as well as other aspects for a real-world implementation will be considered and anticipated by simplified workarounds as mentioned above. However, they will neither be fully considered nor implemented in this project but refer to necessary conditions for concrete industrial system implementations that are beyond the scope of the FAIRWork project.

3 PROTOTYPE DEPLOYMENT

This chapter details methodologies employed in deploying the services while providing an insightful overview of how the prototype is deployed within the broader architecture. The chapter aims to present a comprehensive understanding of the deployment lifecycle, from installation to execution. An overview in the form of a deployment diagram is shown in Figure 47: Deployment DiagramFigure 47.

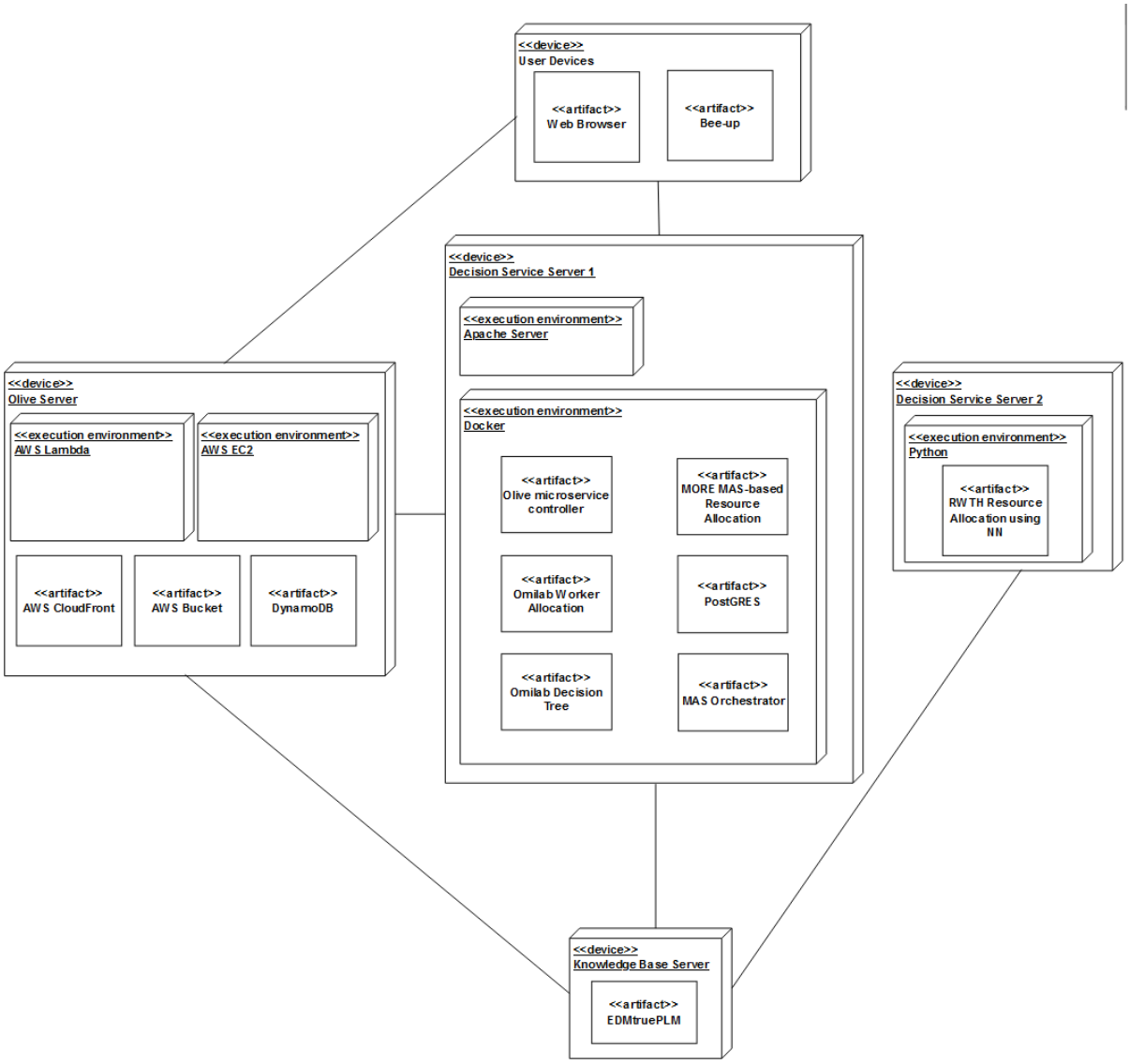

Figure 47: Deployment Diagram.

3.1 Deployment of OLIVE Framework, Workflow Engine and User Interfaces

The OLIVE framework and the workflow engine are deployed in AWS as a combination of serverless AWS Lambda functions and AWS EC2 instances. They are relying on AWS DynamoDB for file storage. While the instance of the OLIVE Framework that we are using for the FAIRWork project is deployed in one sandbox environment on AWS, the instance of the workflow engine is currently shared with other OLIVE Framework instances. But each sandbox, as that from FAIRWork, has its own APIs for storing, listing and triggering workflows. So, when a workflow is saved

with the corresponding API, it is stored in a service that is sandbox-specific. The UI components are deployed in an AWS Bucket and distributed through AWS CloudFront CDN.

3.2 Rule-based resource Allocation and Decision-Tree Service Prototype Deployment

The Bee-Up modelling tool is deployed as a standalone modelling tool that can be downloaded and installed on Windows, Ubuntu, and MacOS. To use this tool in this context, the extensions must be added. See the service description for more information. The installation introduction can be found on the download page of the modelling tool, which is provided in the description of the service.

The rule-based service consists of two applications, which are used together. An OLIVE instance and a server for the worker allocation. OLIVE is used to start configured decision services (like the *Line Assignment Assessment* service) and to wrap the resource allocation service and allow configuring them and making them usable. For example, to pre-configure the endpoint for the line assignment assessment.

Both applications are provided as a docker image, which is stored in the GitLab repositories of the corresponding projects. Then, on the server, these images were pulled, and the containers were started. Further information on how the two applications can be downloaded and started is provided in the documentation on their GitLab pages.

The decision tree service was deployed as one service, which can be started locally as Python or over its docker image. The docker image is available over the corresponding GitLab project.

To ease the installation and configuration of the firewalls on the prototype server, provided by our partner JOANNEUM RESEARCH, an Apache2 server was installed. Only the two standard ports (80 for HTTP and 443 for HTTPS) are open. The calls to these are then forwarded to the correct server with its port internally.

3.3 AI-services deployment

The deployment of AI services within the project involves hosting and accessibility through dedicated servers provided by JOANNEUM RESEARCH. These servers feature four exposed ports, with port 20 allocated for remote SSH connections and ports 80, 443, and 8080 reserved for exposing functionalities such as REST-API endpoints.

The Anaconda environment manager is utilised on the hosting servers to manage the Python environments effectively and prevent conflicts related to third-party libraries. Using Anaconda enables each AI service to operate within a dedicated Python environment.

In the context of AI services, deployment entails encapsulating the core functionality of an AI service, including the pertinent model or algorithm, into a REST API. This REST API endpoint is executed as a continuous process on the server.

The deployment leverages the Flask library to establish the REST API endpoint. Flask eases the creation of RESTful endpoints and offers testing and development capabilities. However, an additional Web Server Gateway Interface (WSGI) is essential for production deployments.

In our AI services deployment, we implement the WSGI interface using the Python library "Werkzeug".

This ensures a robust and reliable mechanism for handling communication between the Flask application and the production server. Utilising Anaconda, Flask, and Werkzeug collectively forms a streamlined and efficient prototype deployment infrastructure, laying the foundation for the successful integration and scaling of our AI services.

3.4 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 48. 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.

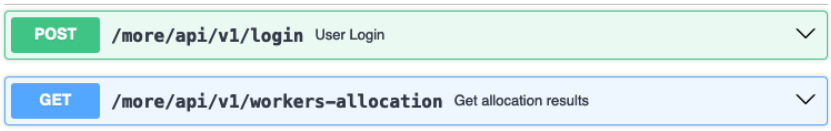

Figure 48: Available endpoints to registered users.

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 is able to 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 were 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 containerisation 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.

3.4.1 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 49.

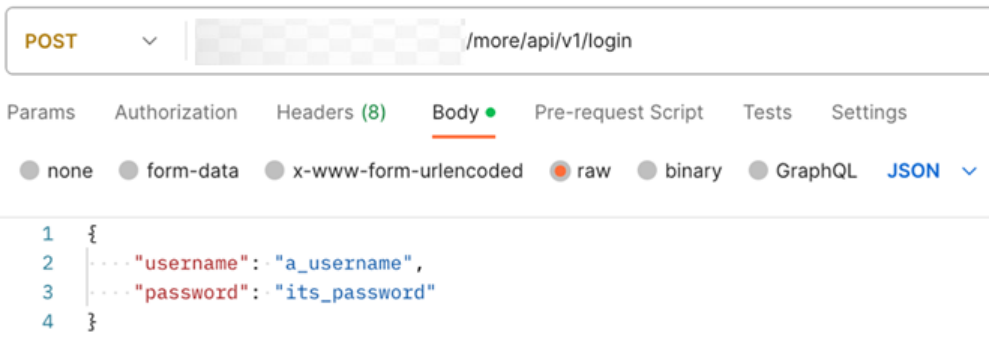

Figure 49: Login endpoint usage.

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 are presented in the response's body in Figure 50. Token validity is set for 30 minutes.

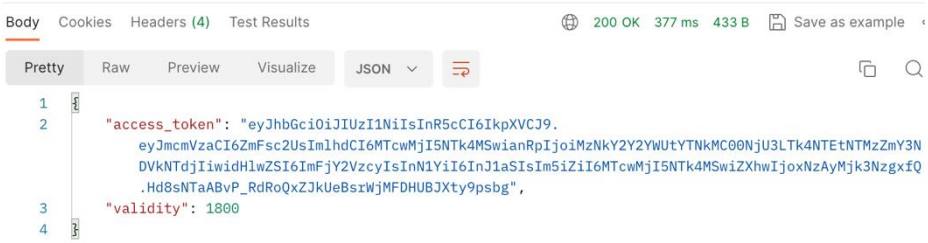

Figure 50: Successful login response.

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 51.

Figure 51: Login endpoint data models.

3.4.2 Workers' allocation endpoint

This endpoint requires a valid JWT access token, retrieved from the login endpoint, to be passed in the Authorization header as a **Bearer** token. Figure 52Figure 52 shows the configuration of the REST request to comply with this API property.

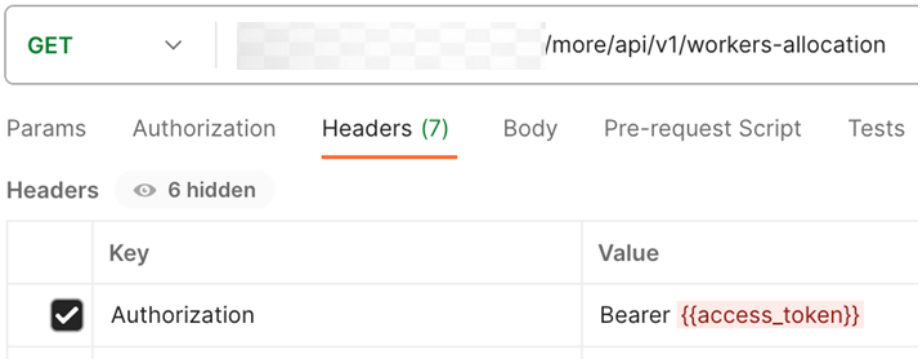

Figure 52: Workers' allocation endpoint usage.

If a malformed/invalid/expired token is used, the API's 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 API returns the allocation results in a JSON according to the agreed-upon format to convey this information. Just as in the login endpoint, data models were defined to guarantee data integrity. 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 OutputModel includes the other two. Thus, it is presented in Figure 53 to depict all models and their integration as described above.

Figure 53: Output model.

4 EXTENDING DAI-DSS FOR NEW USE CASE SCENARIOS

4.1 Extending workflows and user interfaces

The modular architecture of the FAIRWork DAI-DSS enables it to support other use cases, in addition to the one covered in the preceding chapters. 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 section 2.3.1, new workflows can be added to the workflow engine, or tasks can be added or removed to modify workflows that have already been created. In order 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. With regard to the UI components, our goal for the second iteration is to offer a collection of more general, reusable components that can be applied to particular use cases with only minor modifications. However, in certain rare circumstances, a UI component may require a whole new design because of its highly specific function.

4.2 Extending Decision Services through Conceptual Modeling

This section focuses mainly on the approach that is described in 2.4.1, and the two experiment services introduced in sections 2.6.1 and 2.6.2. It will discuss what the two services can be used in other use cases, what this may imply, and how the presented conceptual modelling approach can be utilised for other use cases or services. In this context, different aspects of where extensions can be made are possible.

But all aspects discussed here are in the context of understanding and defining decision cases and the needed information, as described in section 2.4.1, and are related to conceptual modelling as a way to encode and represent knowledge as artefacts used in the procedure.

A new decision service endpoint must be established to use the rule-based decision-making experiment for other use cases. Therefore, a new model must be created containing the decision knowledge. How such a model can be created and transformed into a decision endpoint can be read in section 2.6.1. But before the model can be created, the decision problem itself must be understood, and the rules must be tested to evaluate if they really solve what they are designed to do.

A rule-based approach can be well used for decisions where experts already have the needed knowledge, and the important task is to gather and encode it. Thereby, the decision must not be made as a whole but can be separated into sub-decisions, depending on the problem. But these sub-decisions can also be defined as rules, helping the service derive the needed intermediate steps for themselves. Depending on the complexity of the problem, rules can be adapted rather quickly if needed and then tested to be evaluated.

Another benefit of rules is that their results and how the result was derived can be comprehended by humans, which is not always that easy with machine learning approaches, like artificial neural networks. On the other hand, for very complex decisions, the encoding in rules can be complex and difficult compared to machine learning approaches, especially if how the best decisions are made is not completely clear. Therefore, rule-based approaches can be helpful in situations where the knowledge is already available but not easily processable by a machine or if it is very important that involved humans must understand how the machine has come to the specific solution.

For using the models to create a rule-based decision service out of it, first, the parameters on which the decision is based are identified and defined. Then, the structure of the overall decision should be designed based on possible sub-decisions. If all rules are defined, then the decision service must be instantiated and can then be used.

If the experiment of the decision tree-based service should be used in other use cases, a new decision tree must be trained. Therefore, the data of already made decisions must be prepared and sent to the decision service, where a tree can be generated and then tested. The data can be either real historical data or generated data in a first understanding of the decision itself. If a tree is trained, it can then be integrated into the overall decision process and tested and improved for the future.

As decision trees are learned from a provided data set, this approach can be applied in cases where data about the real environment already exist or where example data can be created. Decision trees themselves can be used for classification, therefore assigning a label to a data point or for the regression assigning a value to a data point.

The created decision trees themselves could be improved by exploring further training algorithms and for which cases or types of data they can be used. Such differences in training could also be integrated into the decision service to allow the comparison of different approaches to find the best for a concrete decision problem.

The conceptual modelling part can also be extended to other cases. At the moment, it was used in two experiments: the rule-based and the decision tree-based decision services to enable a configuration and/or a visualisation. However, these are not the only approaches that can be used together with conceptual modelling to allow a configuration and adaptation to concrete decision problems. Here, one possibility of extensions would be to analyse how other approaches can be configured or supported through conceptual modelling, especially out of the modelling method that is already used in the procedure introduced in section 2.4.1.

By allowing the use of other modelling methods or improving the existing one, other use cases could be covered. If such models are then kept in a combined tool, the different models themselves could also be linked to allow an improved understanding of the decision situation and what is used or tried to support this decision case.

Another extension in this context could be to make experiments on how conceptual models can help to visualise the decision to support their understanding. This can either be on a general way to explain how the overall process is to make a decision for this problem or to visualise how a concrete decision was made. Such visualisation of a concrete decision can support its understanding of the individuals who are influenced by the decision.

If a model-based approach is used for supporting the configuration of decision services, the models can also be used to support users. For example, syntactic checks can be made to ensure that the encoded knowledge is feasible for a specific service. In the context of the rule-based system, this could be that for each rule, a value in a feasible data type is defined. For example, if a parameter is defined as a Boolean, only values can be used in the rules, which can be mapped to the Boolean value.

To improve the usage of the models in the overall procedure, the models can be better linked, and supportive technology can help. This can be that the models in the different steps can be linked together, and the overall process can be supported by reusing information that was already modelled.

Finally, the usage of the models can be improved by integrating the modelled knowledge into the orchestration. The two introduced experiments focused on how modelling can be used in the context of individual decision services. An extension could be on how modelling can be used to support the configuration of the orchestration.

To summarise, the extensions made in the context of the two introduced decision service experiments in combination with conceptual modelling. Possible extensions could be made to improve or further utilise the underlying decision algorithms, adapting them to other cases with the introduced approaches or improving the

modelling to better support the users and support other decision algorithms. Improving the modelling capabilities can be aligned with the procedure to understand and define the decision cases.

4.3 Extending DAI-DSS capabilities through AI-enrichment services

Expanding the capabilities of DAI-DSS to meet the evolving needs identified in partner CRF's data involves addressing intricate resource allocation challenges in the production process. One possible extension of the use case scenario revolves around restructuring it into a two-layered problem: scheduling orders and allocating workers to machines. A block diagram of the two-layered structure is provided in Figure 54.

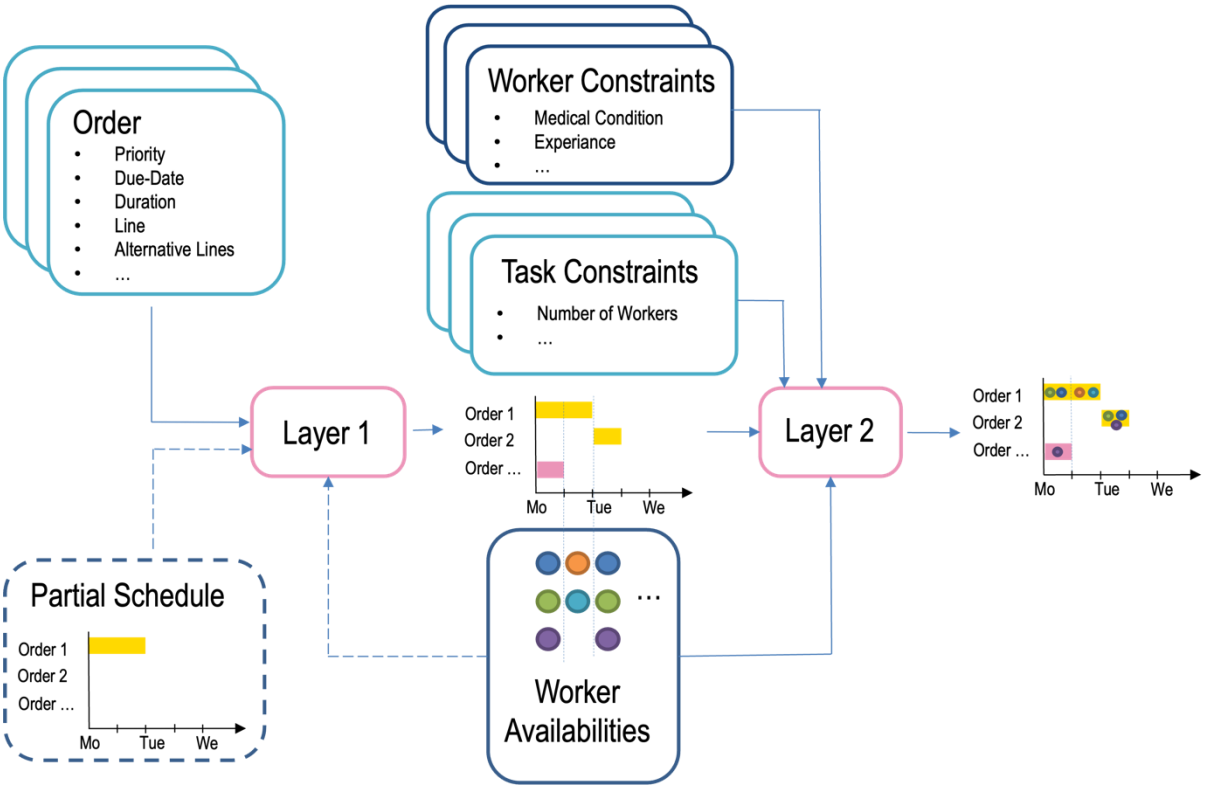

Figure 54: Block diagram of a two-layer optimisation problem structure.

Upon examination of the data provided by CRF, it is noticeable that the resource allocation problem of the use case can capture more details and peculiarities in production when modelled as a two-layered problem. The first layer involves scheduling mapping orders to machines based on geometry, due dates, and production priority. The daily decisions on worker allocation to production lines further manifest the second layer.

The worker allocation use case, already addressed in the prototype, can be refined by incorporating a time dimension. Extending worker allocation decisions over time allows for a fair distribution of taxing tasks among workers. For example, workers handling physically demanding tasks on one day can be allocated to less taxing lines the next day, promoting fairness and enhancing worker well-being.

The proposed two-layered approach involves sequence planning for machines in layer 1, or in other words, creating a production schedule and worker allocation based on the schedule in Layer 2. The core scheduling problem in layer 1 resembles a Flexible Job Shop with use case-specific constraints, where each order can be considered a job and each geometry a task. The second layer can be viewed as a use case-specific assignment problem.

Both of these layers formally result in mathematical optimisation problems. The Flexible Job Shop Problem is a complex NP-hard problem that cannot be solved efficiently by exact solving approaches due to its NP-hardness.

The problem-specific constraints and objective function result in an inability to apply problem-specific heuristics. AI, with its learning and adaptation capabilities, is especially well suited to tackle optimisation problems at hand.

Implementing the first layer of the suggested two-layered structure requires a sophisticated objective function in the CRF use case, considering priorities, due dates, alternative lines, and worker availabilities for optimal scheduling concerning the CRF's business and worker well-being goals. The inclusion of fairness considerations introduces additional complexities to the optimisation process. Each layer becomes a workflow component, implemented as services in the system architecture. The services from the initial prototype can fit into the second layer of this structured approach through further development. Planning worker allocation beyond shifts anticipates adjustments, ensuring fairness even in unforeseen circumstances, and strategic planning minimises disruptions in case of unexpected worker absence at the beginning of a shift.

A rigorous problem-solving approach in the suggested two-layered structure requires formalising each layer as a concrete, discrete optimisation problem with well-defined constraints and objective functions that arise from the use case. That formalisation could be done as an exact constraint program or mixed-integer program using state-of-the-art general-purpose optimisation libraries like Google OR Tools. Due to the NP-Hardness of the problems of the layers at hand, these formulations are expected to scale poorly in terms of runtime with increasingly large problem instances. Nonetheless, they are required to act as a baseline for evaluating the quality of AI Methodologies that can adhere to the runtime of a real-world production environment.

Two concrete methodologies that could be utilised to solve such optimisation problem instances are reinforcement learning and Monte Carlo Tree Search. Reinforcement learning is, in essence, a generic optimisation framework capable of finding sophisticated solution strategies by learning from experience collected by interacting with a so-called environment. Utilising reinforcement learning requires mainly defining the problem or system dynamics in an environment. After this step, state-of-the-art algorithms like the Proximal Policy Optimization algorithms can be leveraged to learn solution strategies for the dynamics of the defined environment.

Such reinforcement learning algorithms require large initial computational resources to learn these solution strategies. Once these solution strategies are learned, they can infer solutions for given problem instances in a short amount of time, which addresses the runtime requirements of a real-world application.

Monte Carlo Tree Search, on the other hand, also requires capturing the system's dynamics in the form of an environment. It looks into the future by simulating different scenarios systematically through promising action sequences. In essence, Monte Carlo Tree Search performs what a human does when playing chess: It goes through different possibilities of actions; in the chess example, this corresponds to the player's possible moves, evaluates them and chooses the action that is the best action it encounters. It does not go through all the moves, only rather promising ones. In the first layer, Monte Carlo Tree Search would sequentially construct a schedule by adding one task at a time. The second layer would allocate one worker at a time and yield a full allocation suggestion for a given schedule at the end. One could summarise that reinforcement learning finds solutions by looking into the past, while Monte Carlo's Tree Search looks into the future. Both approaches can also be combined into a class of algorithms called Neuro-guided Monte Carlo Tree Search. One member of this class of algorithms is Deepmind's AlphaGo, which famously surpassed human performance in the Boardgame Go. Neural guidance in this context means employing neural networks in certain parts of the Monte Carlo Tree Search algorithm, for example, the action selection for the simulations or the evaluation of action sequences.

The proposed two-layered approach has the possibility to integrate into the existing system architecture, with specific services designed to handle resource allocation for layer 1 (sequence planning) and layer 2 (worker allocation). By extending DAI-DSS to accommodate these sophisticated use case scenarios, the overall system could enhance its capabilities to meet the dynamic challenges present in CRF's production environment. The two-

layered structure enables the application of a wide range of algorithms, like reinforcement learning and Monte Carlo Tree Search.

4.4 Extending Real-World Data Provisioning

In the upcoming prototype, JR intends to incorporate additional components to facilitate decision-making in an industrial context.

Firstly, a data catalogue will be added, which will prove useful for data professionals to extract maximum value from their organisation's data and enable developers to manage their ever-changing data ecosystem more efficiently. This component is a data asset catalogue, enabling external resources to provide items that can be used by different service-based infrastructures. Its functions include discovering and managing data assets, such as user registration, searching for data assets, and ingesting and describing data assets. Secondly, the search function offers numerous standard functions. For instance, search terms will match against various aspects of a data asset. These include asset names, descriptions, tags, terms, and owners, as well as specific attributes, such as the names of columns in a table. Thirdly, it provides a robust method for annotating entities within the FAIRWork data asset marketplace.

Second, JR will work on more decision pattern services which are highly relevant in the industrial context, such as the electronic and automotive industries.

The group of services is around process optimisation of resources by using multiple objectives for different decision patterns.

- Optimization/Heuristic-Based Resource Allocation Service: This service generates a specific mapping of workers to individual tasks on the production line. The decision-making process takes into account time constraints, various constraints, and human factors, such as worker stress levels. The resulting approach leads to efficient and timely decisions.
- Service: Optimisation/Heuristic-Based Resource Allocation Service: AI-based service assesses the effectiveness of job rotation in improving productivity and decreasing physical and mental fatigue. The evaluation is grounded on the available data on the current workload, the worker's stress levels, and their preferences.
- Service: Predictive Maintenance for complex production units: In this service, we develop by using ML techniques to predict possible machine failures.

Further services are planned to be developed to support the Human Factors analyses and human-centred optimisation, such as:

- Service: Stress Detection using Machine Learning-based models: from the exploratory study in the Human Factors Lab, we will collect data using wearables (smartwatch, biosensor shirt, eye tracking glasses, virtual events) about physiological strain with respect to physical work, as well as data about cognitive-emotional strain with respect to decision-making at the workplace. In the context of the different workplaces, we will develop a refined estimation of physiological and cognitive-emotional strain, respectively.
- Service: Persona-based clustering of human-centred data: We will apply clustering methodologies, such as the Expectation-Maximisation algorithm, on these data and receive persona-based clusters of characteristic strain patterns for further use by optimisation. These persona-based representations will enable the use of characteristic and, at the same time, anonymised information about the workers' psychophysiological requirements.

5 SUMMARY, CONCLUSIONS AND OUTLOOK

The documentation outlines the foundational elements and architecture of the system, as established in deliverable "D4.1 – DAI-DSS Architecture and Initial Documentation and Test Report". The initial segment provides a comprehensive overview of the building blocks, detailing their general structure and integral role within the system. The focus then shifts to the DAI-DSS User Interface (UI), using the Industrial Partner CRF's Workload Balance Use Case Scenario as a reference point. This part of the documentation emphasises the UI's crucial role in decision-making processes. It offers an insightful perspective on how the UI facilitates the visualisation of key components related to decision-making, enhancing the overall functionality and value of the system. The UI's significance is further highlighted by its ability to bridge the decision-maker with the system, ensuring a transparent and effective visualisation of the decision-making process, as demonstrated through introductory images of the UI in the prototype stage.

The next key component discussed is the DAI-DSS Orchestrator, which is portrayed as the central nerve of the system's architecture. Its primary function is to orchestrate the flow of information among various segments of the system, akin to a conductor ensuring harmony in an orchestra. This orchestration currently happens through two parallel mechanisms, with the advanced Workflow-based Orchestrator being a notable feature. This orchestrator utilises a workflow engine to execute tasks along a predefined path, ultimately facilitating the activation of AI services that provide recommendations for decision-making while the second uses agents for a decentralized approach in a Multi-Agent perspective. The documentation conveys the orchestrator's functionality through a practical example - the Workload Balance Use Case Scenario - illustrating how worker data is processed and how parts orders are managed. This detailed explanation of the step-by-step orchestration process in a real-world manufacturing scenario underscores the Orchestrator's role in integrating various building blocks for efficient system functioning.

The DAI-DSS Configurator is a comprehensive tool designed to streamline the configuration of decision support systems, which is comprised of two main components: the Configuration Framework and the Configuration Integration Framework. The former facilitates the creation of decision models and strategies in various modelling environments, while the latter generates configurations for other system components based on these models. In this deliverable, the steps for configuring the current prototype are explained with images for a deeper understanding of how it graphically occurs in the frameworks approached. This prototype includes an experiment for configuring rule-based decision services using conceptual modelling, employing tools like the ADOXX metamodeling platform and the Scene2Model tool for workshop facilitation and digital conceptual modelling. Additionally, it integrates Business Process Model and Notation (BPMN) and Decision Model and Notation (DMN) for standardising decision processes. The Configurator also focuses on the configuration of microservices and workflows using various tools, enabling the DAI-DSS User Interface configuration through a wizard that allows for the combination of different UI components. The DAI-DSS Configurator represents an approach to configuring and optimising decision support systems, encompassing a range of tools and methods for efficient and effective decision-making support.

Following this, the DAI-DSS Knowledge Base acts as a repository for data and decision models essential for the system's decision support. It stores a wide range of information, including user properties, sensor data, and machine learning processed data, being accessed by various DAI-DSS components for configuration and decision-making purposes. It integrates with the DAI-DSS Configurator and uses REST API for data retrieval, playing a crucial role in the data flow within the DAI-DSS architecture. This setup facilitates the creation of model-aware web applications and the orchestration of AI services and workflows, which are essential for effective decision support.

In sequence, the integration of AI services in the project is approached. It is achieved through REST-API endpoints, aligning with industry standards for ease of access and interoperability. The use of Swagger documentation plays a role in defining these API endpoints, providing human-readable descriptions to facilitate understanding. The standardised format of HTTP requests and JSON-formatted data ensures a streamlined exchange of information, enhancing the system's compatibility across various software environments. Algorithms involving Neural Networks, Decision Trees, Multi-Agent Systems, Conceptual Modeling and Linear Sum Assignment Solver were experimented with in this prototype to provide support to decisions in an industrial use case. Furthermore, a method for integrating human factors data into a data lake system, measuring workers' physiological and psychological stress to assess their resilience through the use of wearable sensors providing a resilience score based on historical data and current strain levels, accessible via a REST API. These approaches aim to optimise human-centred decision-making in the manufacturing industry.

To provide a complete set for this prototype, its deployment is covered in detail regarding the deployment of the AI services, including their access points and containerisation. It deals with the hosting and management of AI services on servers, as well as frameworks for REST API creation. The text also hints at future enhancements, including the integration of a data catalogue for improved data management and decision pattern services for resource allocation and predictive maintenance. It provides a technical overview of the deployment and integration of various software tools and AI services in an industrial environment, with a focus on security, efficiency, and future enhancements.

The last section discusses the extension of the DAI-DSS with new services for various industrial use cases. It delves into the adaptation and application of rule-based approaches, explaining how services can be customised through conceptual modelling, which involves understanding decision problems, encoding expert knowledge into rules, and creating models for decisions. It highlights the comprehensibility of rule-based approaches over machine learning methods like neural networks, especially in situations where decision-making knowledge is already available. It also highlights optimisation and heuristic-based resource allocation, predictive maintenance using machine learning, and persona-based clustering of human-centred data. That chapter also outlines plans to enhance real-world data provisioning with a data catalogue, facilitating efficient data management and decision-making in industrial contexts. These services aim to optimize resource allocation, improve productivity, and address human factors like stress and fatigue in the workplace. The ultimate goal is to enhance the DAI-DSS's capabilities for complex decision-making scenarios.

Looking ahead, the future of the DAI-DSS presents an exciting and transformative journey towards more efficient, intelligent decision-making in industrial contexts. The system's architecture lays a solid foundation for this evolution. The integration of sophisticated AI services through REST-API endpoints and the utilisation of various AI approaches point towards a future where decision-making is not only supported by AI technologies but also highly adaptable to industrial needs. The focus on human-centred design, evident in the integration of human factors data, underscores a commitment to enhancing worker well-being and productivity.

6 REFERENCES

References are included as footnotes within the text.