Technologies for Manufacturing as a Service Ecosystems

Deliverable D3.5

D3.5. Context-driven optimised (re-)configuration services v1

WP3: DT Modelling, Operation and Governance for resilient value networks

Editor:	Ioannis Avgerinos
Lead beneficiary:	AUEB
Version:	1.0
Status:	Final
Delivery date:	30/06/2025
Dissemination level:	PU (Public)

This project has received funding from the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101138517.

Deliverable Factsheet

Grant Agreement No.	101138517
Project Acronym	Tec4MaaSEs
Project Title	Technologies for Manufacturing as a Service Ecosystems
Start date	01/01/2024
Duration	36 months

Deliverable Name	D3.5. Context-driven optimised (re-)configuration services v1
Related WP	WP3: DT Modelling, Operation and Governance for resilient value networks
Due Date	30/06/2025

Author(s)	Ioannis Avgerinos (AUEB)
Contributor(s)	Aggelos Ioannis Lagos, Yiannis Mourtos, Georgios Zois (AUEB), Sotiria Antaranian (ATC), Konstantinos Kaparis (UoM), Metin Turkay (SMO), Konstantinos Kalaboukas (MAG)
Reviewer(s)	Patricia Kasla (TEK), Kym Watson (IOSB)
Approved by	All partners

Document History

Version	Date	Author(s)	Organisation	Description
0.1	30/04/2025	Ioannis Avgerinos	AUEB	ToC
0.2	16/06/2025	Ioannis Avgerinos, Aggelos Ioannis Lagos, Yiannis Mourtos, Georgios Zois	AUEB	First draft
0.3	25/06/2025	Patricia Casla	TEK	Peer review
0.4	26/06/2025	Kym Watson	IOSB	Peer review
0.5	27/06/2025	Ioannis Avgerinos, Aggelos Ioannis Lagos, Yiannis Mourtos, Georgios Zois	AUEB	Addressing the comments from the internal review
0.6	30/06/2025	Armend Duzha	MAG	Quality check
1.0	30/06/2025	Ioannis Avgerinos	AUEB	Final version for submission

Executive Summary

This deliverable reports on the progress of the optimisation service, which supports the computation of alternative production schedules within the contexts of VN1 and VN2. The optimisation service in the Tec4MaaSEs platform plays a central role in combining incoming production demands over time with the available resources provided by industrial partners. To compute alternative compositions that ensure both rapid production completion and a fair distribution of tasks among participating industries. Following a description of the scheduling challenges faced by the industrial partners, the deliverable presents the mathematical models employed during the first phase of implementation.

For VN1, a primary mathematical model addresses the Hybrid Flexible Flowshop Scheduling Problem, which is relevant to both industrial partners. For VN2, a more advanced level of implementation has been achieved, enabling the inclusion of an extended section on experimental results using randomly generated, yet co-monitored, small-scale datasets in collaboration with the involved partners. This section also details the optimisation methods developed so far. In addition, the deliverable outlines the next steps, which include the exploration of alternative methods to enhance the scalability of the service.

The outcomes of this deliverable are expected to be evaluated by other platform services, either as providers of input data to the optimisation module or as consumers of its output - for instance, in subsequent negotiation phases or in the visualisation of results. Further details on the integration of the optimisation service with the overall platform are provided in Deliverable D4.1.

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

Acronym	Description
MaaS	Manufacturing-as-a-Service
CP	Constraint Programming
MILP	Mixed-Integer Linear Program
VN	Value Network
HFFS	Hybrid Flexible Flowshop Scheduling problem
IoT	Internet-of-Thing
AI	Artificial Intelligence
LB	Lower Bound
UB	Upper Bound
WIP	Work-In-Progress
EB	Electronic Board
JIT	Just-In-Time scheduling
rpm	Rate-per-minute
LNS	Large Neighbourhood Search
PSO	Particle Swarm Optimisation

Glossary

Term	Description
Constraint Programming	A problem-solving approach that involves defining variables, their possible values (domains), and constraints that restrict combinations of values. Solutions are found by systematically exploring feasible combinations that satisfy all constraints.
Flowshop scheduling	A production scheduling problem where a set of jobs must be processed on multiple machines in the same order.
Just-In-Time scheduling	A scheduling strategy aimed at minimising inventory and work-in-progress by producing goods only as they are needed, aligning production closely with demand. The objective of JIT scheduling problems is the minimisation of earliness and tardiness over a due-date.
Large Neighbourhood Search	A heuristic optimisation technique that iteratively destroys and repairs parts of a solution to explore large neighbourhoods of the solution space. It is effective for solving complex combinatorial problems.
Makespan	The total time required to complete a set of jobs from start to finish. In scheduling, it typically refers to the time when the last job is finished.
Mixed-Integer Linear Program	A mathematical optimisation model in which some variables are constrained to be integers and the relationships among variables are expressed as linear equations or inequalities.
Optimality gap	The difference between the best-known feasible solution and the best known bound (typically lower or upper) on the optimal solution. It is often expressed as a percentage and used to measure how close a solution is to optimality.
Pareto frontier	In multi-objective optimisation, the set of all non-dominated solutions, where no objective can be improved without worsening at least one other. Also known as the Pareto front.
Particle Swarm Optimisation algorithm	A computational method inspired by the social behaviour of birds or fish. It optimises a problem by having a population (swarm) of candidate solutions (particles) move through the solution space, adjusting their positions based on individual and collective experiences.
Reinforcement Learning	A machine learning paradigm where an agent learns to make decisions by interacting with an environment, receiving rewards or penalties based on its actions, and aiming to maximise cumulative reward over time.

1 Introduction

Manufacturing-as-a-Service (MaaS) is a dynamic paradigm that transforms how manufacturing resources are provisioned and consumed. Unlike traditional centralized models, MaaS operates on a decentralized and service-oriented architecture in which companies alternately act as service providers or service consumers, depending on real-time demand and capacity. This dual-role flexibility fosters collaboration and enhances the adaptability of the manufacturing network to changes in production requirements and external disruptions.

A MaaS ecosystem is typically composed of distributed manufacturing services connected through a digital platform that coordinates scheduling, logistics, and execution. These services may include additive manufacturing, machining, plastic injection moulding, EB's production etc - each offered by different providers who register their capabilities and capacities on the platform. In parallel, consumers submit their production requests, specifying constraints and priorities.

The platform orchestrates this interaction by matching requests with available services and generating optimised production schedules that account for multiple objectives, such as minimizing makespan, locality of operations, earliness, tardiness. The optimisation engine supports multi-objective decision-making using models like Constraint Programming (CP) or Mixed-Integer Linear Programming (MILP).

This architecture is dynamic for several reasons:

1. Role Flexibility: Each company can switch roles between consumer and provider depending on the context.
2. Real-time Scheduling: Job assignments and machine allocations are determined on-demand, considering up-to-date machine availability and transport times.
3. Decentralized Resource Use: The system avoids centralized bottlenecks by distributing jobs across multiple nodes-providers-machines, balancing load while respecting precedence and eligibility constraints.
4. Digital Interoperability: Interaction with analytics and production planning is enabled through digital twins and service composition mechanisms.

5. Resource availability: The available production units, and even the participating companies, may change over time, resulting in dynamic production capacities within the platform.

The consumers in this model are entities that request the manufacturing of specific parts and services, each combination is defined as a job, with specific resource and time constraints. The providers, on the other hand, make their manufacturing infrastructure available to fulfill these requests, subject to capacity, required capability, and time window limitations.

MaaS benefits include:

- Improved resource utilisation through sharing of underused capacity.
- Cost reduction via ad-hoc reconfiguration and avoidance of idle time.
- Enhanced resilience to demand variability and supply chain disruptions, due to its modular and distributed nature.
- Increased flexibility in scheduling complex jobs involving multiple operations and inter-provider transport.

In conclusion, the general MaaS architecture underpinning Tec4MaaSEs reflects a service-based, adaptive, and collaborative model. It leverages advanced digital technologies and optimisation methods to support complex, multi-stakeholder production processes in real time, unlocking new opportunities for efficiency, scalability, and value co-creation in manufacturing.

1.1 Document Scope

This deliverable outlines the design and implementation of the optimisation services developed within the Tec4MaaSEs platform. Its scope includes the description of the optimisation services implemented within Tec4MaaSEs, the way they are placed into the overall platform workflow, and their configuration and use across the three main value networks addressed in the project.

It further presents a MaaS representative technical approach developed in detail for VN2, which also serves as the basis for the methodology adopted in VN1. While VN3 is discussed in terms of its relevance, it is not addressed in depth due to limited integration with the platform's optimisation capabilities. VN3 is characterized by long-term procurement and strategic negotiation for infrastructure projects. Service requests within this context are infrequent, highly customized, and tailored to specific project requirements. Consequently, decision-making is guided not by volume or repetition, but by complex, high-level trade-off analyses across potential suppliers. Given these dynamics, the composition service holds limited relevance and is not meaningfully applicable within the VN3 environment.

The optimisation service in Tec4MaaSEs plays a central role in transforming structured manufacturing requests into executable production schedules. It operates as part of a broader (re-)configuration framework within the Tec4MaaSEs platform, which orchestrates service composition through the interaction of analytics, optimisation, and data registry modules. These components are designed to support agile scheduling decisions in dynamic manufacturing environments, where both demand and provider availability can change rapidly.

The end-to-end flow begins with the submission of a service request by a consumer. This request typically includes qualitative and quantitative descriptions of the desired manufacturing output, such as part specifications, deadlines, and any known constraints. At this stage, some values, such as tardiness penalties or part eligibilities, may be undefined.

These inputs are first passed to the analytics service, which enriches the data by filling in missing values or estimating these parameters. The combination of inputs acquired from the "Search & Match", "Dataspace connector", and "Bill of Process Generator" (see D4.1 for more details) is a well-structured dataset that includes a list of operations (with precedence relations), eligible machines, time windows, and optional business preferences such as locality or fairness weights.

This data is then encoded in a JSON format and forwarded to the optimisation service, which applies formal mathematical modelling to generate feasible and efficient production schedules. Specifically, Tec4MaaSEs leverages CP models to solve a distributed version of the flowshop scheduling problem. These models incorporate precedence constraints, machine-specific eligibility, inter-provider transportation delays, and multi-objective trade-offs.

The output of the optimisation process is a set of alternative schedules, returned to the platform for further use. These schedules are exposed to the stakeholders - both consumers and providers - in a negotiation phase, where trade-offs can be evaluated and a final plan agreed upon. This is a post-optimisation coordination step, where stakeholders select the most suitable schedule from the alternatives, balancing cost, fairness, and production constraints. If a consensus on the schedule cannot be easily reached, the platform may request additional alternative compositions from the optimisation service.

Once a schedule is selected, it can be communicated to the providers - potentially through their digital twins - allowing them to update their machine execution plans accordingly. The providers also return feedback to the platform regarding machine usage, real-time capacity changes, or maintenance status. In the case of disruptions (e.g., a machine becomes unavailable), the system can re-trigger the analytics and optimisation phases with the updated context.

This complete pipeline - from initial request to final schedule agreement - is depicted in the figure below, where the platform acts as the central mediator between consumers and providers, exchanging information related to job requests, available resources, and scheduling outcomes.

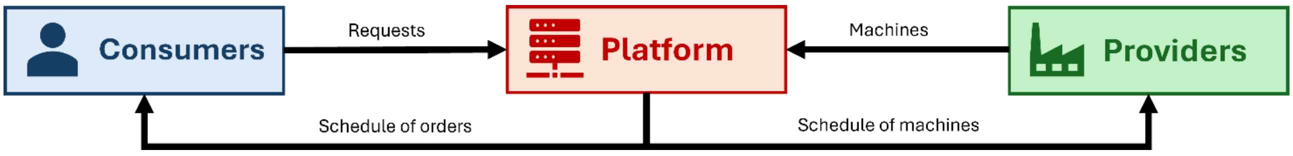


Figure 1. Exchange of information between the stakeholders and the platform

In summary, the optimisation service in Tec4MaaSEs enables the platform to deliver high-quality, adaptive production schedules within complex, multi-actor manufacturing ecosystems. It operates on real-time data processed by analytics, applies formal scheduling models tailored to the flowshop structure of the problem, and integrates with execution processes through clearly defined input/output channels. This design supports the agile, context-aware manufacturing vision central to the Tec4MaaSEs initiative, and ensures that dynamic reconfiguration of production plans remains both optimal and responsive.

1.2 Document Structure

This deliverable opens with an Executive Summary, then moves into an Introduction that clarifies the document’s scope and briefly explains how the remaining chapters are organised. The core technical content follows in the “Optimisation models and methods” chapter 3, where each value network is treated in its own subsection, together with a short discussion on how the optimisation service exchanges data with the rest of the platform’s components. The 4th chapter gathers the functional requirements for each optimisation service, detailing user-facing use-cases, an architectural view, and the underlying technology stack.

2 Optimisation models and methods

Within the Tec4MaaSEs framework, all value networks (VNs) share a common structural characteristic: they involve the scheduling of manufacturing operations across multiple services and machines, typically under a flowshop-like model. Each job consists of a series of operations that must follow a specific sequence, executed on machines with eligibility constraints, transport dependencies, and varying availability windows. This shared structure provides a unified foundation for optimisation modelling, especially in the context of MaaS, where services are offered and consumed dynamically by different actors.

Despite this shared structure, the relevance and applicability of optimisation differ across the three VNs. VN3, which concerns facilities construction in the hydrogen sector, involves highly specialised equipment and bilateral exchanges between a contractor and suppliers. The nature of interactions in VN3 is largely strategic and document-driven, rather than focused on production-level scheduling or short-term execution. As a result, VN3 does not align with the operational logic of the MaaS optimisation approach developed within Tec4MaaSEs and is therefore not included in the core optimisation workflow.

In contrast, VN2 and VN1 represent complementary use cases well-suited to dynamic optimisation. VN2, which focuses on additive manufacturing, machining and plastic injection moulding manufacturing services, presents the most general and complex case. It involves three organisations - Moldes URA, ERREKA, and Tekniker - operating with overlapping roles as both providers and consumers. The production workflows in VN2 require coordination across distributed resources, integration of inter-provider transportation delays, job precedence constraints, and multi-objective decision-making. The scheduling problem in this setting is modelled as a distributed flowshop with additional constraints, including machine eligibility, renewable resource constraints, and discrete time window availability. The optimisation models developed for VN2 were implemented using CP models, with objectives focused on minimising makespan and locality weight, and experimental evaluation through Pareto frontier analysis.

VN1, on the other hand, represents a more specific but significantly larger-scale version of the problem. It concerns the production of electronic boards for white goods and includes a simpler service structure. While the constraints are less diverse, the number of jobs and processing entities is substantially higher, resulting in scalability challenges for exact optimisation methods. The problem can still be modelled within the same distributed flowshop framework, but due to the size of the instances, the use of MILP or CP in their standard form becomes computationally challenging. For this reason, ongoing work within Tec4MaaSEs explores enhancements through task batching, grouping, and clustering techniques to enable decomposition or approximation of large-scale instances. These methods allow us to apply the same general optimisation approach used in VN2 to VN1, by pre-processing the input data to reduce dimensionality or enforce structure.

Across both VN1 and VN2, the problem is inherently multi-objective. Different stakeholders have distinct and sometimes conflicting priorities: consumers seek rapid service completion, platform operators aim for fair workload distribution, and providers prefer assignments that reduce cost or increase utilisation. Objectives addressed in our models include minimising makespan, earliness, tardiness, transportation-based locality, and fairness in job allocation. The scheduling decisions must therefore balance these competing criteria under complex constraints, while remaining robust to changes in resource availability or demand.

In summary, the optimisation approach developed in Tec4MaaSEs is rooted in a general flowshop scheduling model tailored to the needs of VN2, the most demanding and generic use case. This approach can be directly extended to VN1, which poses scalability rather than modelling challenges. VN3, although relevant to manufacturing coordination at a strategic level, does not fit within the scope of the scheduling-focused MaaS optimisation services targeted by this work.

2.1 VN1

2.1.1 Background/State-of-art

VN1 comprises the production lines of Acron and Karel. Each production line presents a Hybrid Flexible Flowshop Scheduling (HFFS) problem. A flowshop scheduling problem involves a set of jobs, each consisting of multiple operations that must be completed in a specific sequence. Each operation is processed on a dedicated machine. A detailed description of the general flowshop scheduling problem is provided by **Error! Reference source not found.**. The hybrid flexible variant extends this framework by introducing multiple identical parallel machines at each stage. Consequently, each job's operation must be assigned to exactly one of the available machines. Moreover, not all jobs require the same sequence of operations - it is possible for specific stages of the production flow to be skipped.

Due to its wide applicability in manufacturing, HFFS has been extensively studied in the literature. **Error! Reference source not found.** proposed MILP models and a particle swarm optimisation (PSO) algorithm for the problem. More recently, **Error! Reference source not found.** introduced alternative constraint programming (CP) models, while **Error! Reference source not found.** focused on metaheuristic methods for a variant that includes transportation times between machines in consecutive stages. The MILP model by **Error! Reference source not found.** addresses a bi-objective version of HFFS that incorporates uncertainties in processing times, setup times, and machine-allocation costs, aiming to minimise both makespan and total machine-allocation costs. Finally, **Error! Reference source not found.** developed mathematical models for an HFFS variant that includes Work-In-Progress (WIP) inventory units between stages. All of these works focus on the minimisation of makespan; the providers of VN1 have highlighted the importance of Just-In-Time (JIT) scheduling for their production environment. A description of JIT in a flowshop scheduling environment is provided by **Error! Reference source not found.**.

2.1.2 Methodology

Problem description. Although the development of optimisation methods incorporating all aspects of the involved production lines is still in progress, an abstract CP model for the HFFS is introduced. For this problem, let J be a set of jobs – each job is a unit of an Electronic Board (EB) type. The production lines are defined by a set of stages S. Each job follows a designated sequence of stages, probably skipping some of the stages of S – a subset S_j for each job *j* contains the stages in which *j* is processed. The components of S_j are placed in the appropriate sequence – e.g., $s_j^i \in S_j$ is the i^{th} stage of job *j*. Set M contains all machines – each machine is dedicated to a particular stage *s*, thus subsets M_s indicate the machines of each stage. Also, each machine belongs to a provider *p* of a set P – subsets M_p indicate the machines of each provider.

Each job *j* requires a sequence of operations, with each operation assigned to a specific stage *s*. Since all parallel machines within a stage are identical, the processing time of an operation, which is deterministically defined by preceding analytics services, is denoted by p_{js}. Although each operation may require specific components and resources, a simplified version is assumed in the initial modelling phase: each job *j* requires a total number of components r_j, and each provider *p* has a maximum production rate R_p. The optimised schedule must ensure that this production rate is not exceeded at any point during the production process.

The JIT scheduling environment introduces a due-date d_j for each job *j*. The objective is to minimise the deviation between a job's completion time and its due date - effectively minimising both earliness and tardiness. It has been noted that tardiness is more critical than earliness; therefore, weights *e* and *t* represent the penalty per unit time for earliness and tardiness, respectively. Table 1 consolidates the mathematical annotations which are used for modelling the scheduling problem of VN1.

Table 1. Mathematical annotations of VN1

Annotation	Description
J	Set of jobs
S	Set of stages; S_j contains the stages of job *j*
P	Set of providers
M	Set of machines; M_s contains the machines of stage *s* and M_p contains the machines of provider *p*
p_{js}	Processing time of job j on stage *s*
R_p	Production rate of components (units/hour) in the industry of provider p
r_j	Number of required components for job *j*
d_j	Due-date of job *j*
e, t	Penalty weights of earliness and tardiness respectively

Mathematical modeling. The problem described is modeled by a CP formulation, which exploits state-of-art CP predicates to enforce precedence and resource-allocation constraints. CP model 1 is presented in Figure 2:

$$
\begin{array}{lll}
\min \sum_{j\in J}(e\cdot E_j + t\cdot T_j) & & (1)\\
\quad \texttt{alternative}(\texttt{jobIntervalA}_{js}, [\texttt{jobIntervalB}_{jm} | m \in M_s]) & \forall j \in J, s \in S_j & (2)\\
\quad \texttt{noOverlap}([\texttt{jobIntervalA}_{js} | s \in S_j]) & \forall j \in J & (3)\\
\quad \texttt{startAtEnd}(\texttt{jobIntervalA}_{js_j^i}, \texttt{jobIntervalA}_{js_j^{i-1}}) & \forall j \in J, i \in [1, |S_j|] & (4)\\
\quad \texttt{noOverlap}([\texttt{jobIntervalB}_{jm} | j \in J]) & \forall m \in M & (5)\\
\quad E_j = \max\{0, d_j - \texttt{endOf}(\texttt{jobIntervalA}_{js})\} & \forall j \in J, s \in S_j & (6)\\
\quad T_j = \max\{0, \texttt{endOf}(\texttt{jobIntervalA}_{js}) - d_j\} & \forall j \in J, s \in S_j & (7)\\
\quad \sum_{j\in J}\sum_{m\in M_p} \texttt{pulse}(\texttt{jobIntervalB}_{jm}, r_j) \le R_p & \forall p \in P & (8)\\
\quad \text{if } x_j \neq p \rightarrow \texttt{presenceOf}(\texttt{jobIntervalB}_{jm}) = 0 & \forall j \in J, p \in P & (9)\\
 & & \\
\quad \texttt{jobIntervalA}_{js} & \forall j \in J, s \in S_j & \\
\quad \texttt{jobIntervalB}_{jm} : \texttt{optional} = True, \texttt{sizeOf} \in [p_{js}, p_{js}] & j \in J, s \in S_j, m \in M_s & \\
\quad x_j \in P & \forall j \in J &
\end{array}
$$

Figure 2. CP model 1 - VN1

The CP model introduces two sets of interval variables – variables which are defined by assigning a start value, an end value and a size value representing the start time, the end time and the duration of a task. Group A (*jobIntervalA*) defines the time intervals of a job on each stage, and group B (*jobIntervalB*) indicates the time intervals of the operations on the assigned machines. Notably, the variables of group B are *optional*: each operation must be assigned to exactly one machine of the stage involved.

The *alternative* constraint (2) ensures that exactly one variable $jobIntervalB_{jm}$ over all machines of a particular stage is selected; variable $jobIntervalA_{js}$ is synchronised with the variable of the selected machine. Constraints (3) ensure that the operations of the same job do not overlap. The appropriate sequence of operations is indicated by constraints (4): operation s_j^i starts at the end of the previous operation s_j^{i-1}. Each machine can process one operation at the same time, as indicated by constraints (5).

Variables E_j and T_j denote the earliness and tardiness of job *j* respectively. Earliness is the non-negative difference between the due-date and the completion time (6), and tardiness is the non-negative difference of the exact opposite – the completion time minus the due-date (7). Variable x_j indicate the provider which is assigned to produce *j*. Each job is processed at the production line of the assigned provider, also consuming its components. *Pulse* constraint (8) ensures that no more than R_p components are consumed by the jobs processed on machine of *p*. Finally, conditional constraints (9) indicate that if *p* is not the assigned provider of a job *j*, then *j* cannot be processed on any machine of *p*. The objective function (1) minimises the total weighted violation of earliness and tardiness.

As mentioned earlier, the large scale of production demand can pose challenges for exact modeling approaches. If the weekly planning horizon leads to an excessive volume of demand, a batching preprocessing stage may be applied. In this case, jobs with similar due dates and operation sequences can be grouped together and treated as single production demands. Consequently, the set J is redefined to represent batches of jobs rather than individual ones. A dedicated "Task Batching Service" - also mentioned in D3.3 - is expected to be responsible for this pre-optimisation stage.

2.1.3 Data requirements

To support the scheduling and (re-)configuration of manufacturing services within the Tec4MaaSEs platform, the optimisation service requires structured input from both consumers and providers. This data is pre-processed by the analytics service and passed to the optimisation engine in a structured format, such as JSON – expected to be defined in future deliverables. The required information includes the following:

Consumer-side input. Each job is associated with the following parameters:

- A unique RequestID
- The RequestDate when the manufacturing need is submitted
- A DeliveryDate window specifying acceptable completion dates – connected with parameter d_j
- The EBTypeID, connected with a designated sequence of operations

Provider-side Input. Each machine registered by a provider (e.g., Acron or Karel) must include the following details:

- Identification:
  - The Company that owns or operates the machine
  - A unique Machine identifier
- Service Capabilities:
  - A list of all the service types the machine can perform – connected with designated operations of EB types
  - A processing time for each operation (i.e., pair of EB type and stage p_{js})
  - The consumption of components for each operation per EB type – connected with parameters r_{js}
- Production rate:
  - The maximum number of available components per hour – connected with parameters R_{jp}
- JIT parameters:
  - Weights per time unit of earliness and tardiness – connected with parameters *e* and *t*.

2.2 VN2

2.2.1 Background/State-of-art

The development of collaborative production networks in the context of MaaS has been examined in production research due to its potential to optimise resource utilisation and minimise operational costs **Error! Reference source not found.**, **Error! Reference source not found.**. In particular, the design of ecosystems in which companies alternately take on the roles of service providers and consumers - as applied in VN2 of Tec4MaaSEs - has been previously presented in Leng, J., Zhong, Y., Lin, Z., Xu, K., Mourtis, D., Zhou, X., Zheng, P., Liu, Q., Zhao, J.L., Shen, W. (2023). Towards resilience *in Industry 5.0: A decentralized autonomous manufacturing paradigm. Journal* of Manufacturing Systems, 71, pp. 95-114.

and **Error! Reference source not found.**. Promoting a decentralised structure enables resilient responses to demand variability and supply chain disturbances **Error! Reference source not found.**. However, it also introduces the need to address additional objectives, such as ensuring that the conflicting interests of stakeholders are fairly balanced.

A comprehensive review by **Error! Reference source not found.** explores the evolution of digital servitisation and its impact on manufacturing enterprises. Their analysis demonstrates how technologies such as Internet-of-Thing (IoT), Artificial Intelligence (AI), and cloud computing facilitate the shift from product-centric to service-oriented business models. While MaaS is not the primary focus of their study, their discussion of digital platforms, business model innovation, and value co-creation provides a conceptual foundation that closely aligns with MaaS principles. The review also underscores the importance of frameworks that enable modular, service-driven, and digitally integrated manufacturing systems. In a more technical context, **Error! Reference source not found.**examine modelling strategies for additive manufacturing within MaaS frameworks, emphasising how digital twins and cloud-based optimisation can enhance production scheduling and flexibility. Their findings highlight the need for real-time, adaptive planning tools, particularly when coordinating tasks among multiple service providers operating under diverse constraints.

In the designed platform, the scheduling problem involves distributing jobs among the service providers within the VN2 ecosystem: Moldes URA, ERREKA, and Tekniker. Recent studies have investigated multi-objective approaches to dynamic task allocation and service matching in MaaS contexts, including game-theoretic models for platform-service coordination **Error! Reference source not found.**, machine brokering in failure-tolerant production networks **Error! Reference source not found.** and metaheuristic approaches for distributed flowshop scheduling problems **Error! Reference source not found.**. However, the specific constraints present in VN2, such as inter-provider transportation, trade-offs between multiple objectives, job precedence relations, and limited machine availability, necessitate the development of mathematical models. CP is evidently an efficient modelling approach for such scheduling problems. These challenges underscore the suitability of exact methods as the most promising solution approach.

2.2.2 Methodology

Problem description. The dynamic manufacturing ecosystem receives a demand of a predefined planning period to a set of machines, made available by the participating companies. Let R be a set of manufacturing requests. Each request consists of a number of identical products, namely *parts*, denoted by I. Subsets I_r correspond to the part of a particular request *r*. Each part must go through a sequence of operations to be manufactured. The operation of each part of a request forms a job; set J contains all operations of all parts, and subsets J_i are limited to the operations of a part *i*. The jobs of the same part are indexed by the order of completion: e.g., j_i^1 is the first operation of part *i*, j_i^2 is the second operation of the same part etc.

The companies make available a set of machines K. Each machine has the capabilities for designated operations. Machine-dependent processing times p_{jk} denote the duration of processing of job *j* on machine *k*. Transferring uncompleted parts to different machines incurs additional delays, indicated by a matrix c_{nk}, for all pairs of machines *n*, *k*. Also, transferring materials from the consumers to the providers or completed products from the providers to the consumers adds delays, defined by matrix f_{rk} for all pairs of consumers of request *r* and machines *k*. Several factors, such as operational compatibility, material constraints and machine capacity restrict the assignment of jobs to machines: a matrix of binary values β_{jk} indicates whether job *j* can be assigned to machine *k* or not. Finally, the availability of each machine is restricted, since the companies may not be willing to provide their resources for designated periods of time. Set T contains pairs of values $[l_t, u_t]$, l_t being the lower limit and u_t being the upper limit of a time window: if a job is assigned to this time window, then processing must start after l_t and be completed before u_t. Each time window relates to exactly one machine on which the respective time window is applied; subsets T_k indicate the time windows of any machine *k*.

Table 2 summarises the mathematical annotations that are used for the Constraint Programming model:

Table 2. Mathematical annotations of VN2

Annotation	Description
R	Set of requests
I	Set of parts; I_r is the subset of parts for request r
J	Set of jobs; J_i is the subset of jobs for part i
K	Set of machines
T	Set of time windows; T_k is the subset of time windows of machine k
p_{jk}	Processing time of job j on machine k
$[l_t, u_t]$	Limits of time window t
c_{nk}	Transportation time between machines n, k
f_{rk}	Transportation time between the consumer of request r and machine k
β_{jk}	Eligibility; 1 if job j can be processed on machine k, 0 otherwise

Constraint Programming formulation. As mentioned above, the involvement of stakeholders with differing interests necessitates the use of multi-objective optimisation approaches. In the first version of the mathematical model, two objectives are considered. The first is the minimisation of makespan, defined as the maximum completion time across all jobs. This classical scheduling objective supports the planning of production within the shortest possible time horizon. The second is a locality weight, a customised metric reflecting the preference of consumers to be served by geographically closer providers.

The overall objective function is formulated as a weighted sum of the two criteria. The weights are determined following a common approach used in Pareto frontier construction **Error! Reference source not found.**, where the sum of the weights equals 1. Specifically, for the two-objective case, a single coefficient α, ranging between 0 and 1, is introduced: the weight of the first objective (makespan) is set to α, while the weight of the second objective (locality weight) is set to $1-\alpha$.

The locality weight is denoted by q_{jk}, representing the preference for assigning job *j* to machine *k*. In the experiments presented in this deliverable, the locality weight q_{jk} is directly derived from the transportation time between the consumer associated with request *r* and the assigned machine *k*. Figure 3 presents CP model 2:

$$\min \alpha \cdot C_{max} + (1-\alpha) \cdot \sum_{j \in J} \sum_{k \in K} q_{j,k} \cdot \texttt{presenceOf}(\texttt{jobIntB}_{j,k}) \tag{10}$$

$$\begin{array}{lll}
\sum_{k \in K} \sum_{t \in T_k} \texttt{presenceOf}(\texttt{jobIntB}_{j,k}) = 1 & \forall j \in J & (11) \\
\texttt{alternative}(\texttt{jobIntB}_{j,k}, [\texttt{jobIntA}_{j,k,t} | t \in T_k]) & \forall j \in J, k \in K & (12) \\
\texttt{presenceOf}(\texttt{jobIntB}_{j,k}) \leq \beta_{j,k} & \forall j \in J, k \in K & (13) \\
\texttt{noOverlap}(\texttt{machineSeq}_k) & \forall k \in K & (14) \\
\texttt{endBeforeStart}(\texttt{jobIntB}_{j_i^{n-1},k}, \texttt{jobIntB}_{j_i^{n},m}, c_{k,m}) & & \\
\forall i \in I, n = 2, \ldots, |J_i|, k \in K, m \in K & & (15) \\
\texttt{startOf}(\texttt{jobIntB}_{j_i^1,k}) \geq f_{r,k} \cdot \texttt{presenceOf}(\texttt{jobIntB}_{j_i^1,k}) & \forall r \in R, i \in I_r, k \in K & (16) \\
C_{max} \geq \texttt{endOf}(\texttt{jobIntB}_{j,k}) & \forall j \in J, k \in K & (17) \\
\texttt{jobIntA}_{j,k,t} : \texttt{interval}, \texttt{optional}, [l_t, u_t] & \forall j \in J, k \in K, t \in T_k & \\
\texttt{jobIntB}_{j,k} : \texttt{interval}, \texttt{optional} & \forall j \in J, k \in K & \\
\texttt{machineSeq}_k : [\texttt{jobIntB}_{j,k} | j \in J] & \forall k \in K & \\
C_{max} \geq 0 & &
\end{array}$$

Figure 3. CP model 2 - VN2

The CP model uses optional interval variables, denoted $jobIntA_{jkt}$ and $jobIntB_{jk}$, to represent the processing of job *j* on machine *k* during time window *t*. Each interval variable captures the start time, end time, and duration of the processing task. These variables are **optional**, meaning they are only active (i.e., present in the solution) if job *j* is actually assigned to machine *k* (within time window *t* for variables *jobIntA*). Otherwise, the variable remains absent and does not participate in the model's constraints. The presence or absence of each interval variable is governed by the built-in expression presenceOf, which evaluates to true if the corresponding variable is active in the solution. This mechanism allows the model to flexibly handle job assignments while enforcing constraints only on relevant variables.

The objective function (10) is the minimisation of the weighted sum of objectives. Constraints (11) assign each job to exactly one machine. Constraints (12) synchronise variables $jobIntB_{jk}$ with exactly one variable $jobIntA_{jkt}$, indicating that the time interval of processing is within an available time window of the assigned machine. Constraints (13) prohibit ineligible assignments of jobs to machines. Constraints (14) use the CP predicate *"noOverlap"* on a sequence variable of machine k: all interval variables $jobIntB_{jk}$ on the same machine cannot overlap, ensuring that no more than one job can be processed on the same machine at the same time. Constraints (15) add delays for transferring products between consecutive stages. Constraints (16) add delays for transferring materials from the consumer to the machine which is assigned to process the first operation in chronological order of a part. Constraints (17) define the value of makespan.

Scheduling alternative compositions. A key feature of the optimisation service is its ability to generate alternative compositions - that is, multiple schedules that satisfy the same production demand. These alternative schedules are made available on the platform to support post-optimisation processes, such as stakeholder negotiation and agreement on a preferred composition.

To produce multiple solutions, the CP model is solved repeatedly - five times in the initial experimentation phase - each time using a randomly selected set of objective weights. This variation encourages diversity in the solutions by shifting the optimisation focus between makespan and locality.

To further ensure diversity, additional constraints are introduced in each iteration to prevent replication of previously found solutions. Specifically, assuming P_j is the company assigned to job *j*, and K_{P_j} is the set of machines owned by company P_j, the following constraints:

$$\sum_{j \in J_i} \sum_{k \in K_{P_j}} \texttt{presenceOf}(\texttt{jobIntB}_{j,k}) \leq |J_i| - 1 \qquad \forall i \in I$$

force at least one job from each part to be assigned to a different provider in subsequent solutions. This cutting-plane-like approach guarantees that each new solution differs meaningfully from the previous ones, especially when combined with changes in objective weighting.

2.2.3 Data requirements

As preceded in 2.1.3, this Section presents the required structured input from both consumers and providers:

Consumer-side Input. Each manufacturing request, submitted by a consumer, contains the following data:

- Request Metadata:
  - A unique RequestID
  - The RequestDate when the manufacturing need is submitted
  - A DeliveryDate window specifying acceptable completion dates
- Requested Services:
  - The ProductRequirements:
    - Material type (e.g., Plastic, Aluminum)
    - Parts: the number of identical items to be manufactured
    - Processing Times: estimated per-part processing durations for each service
- Optimisation Criteria:
  - A list of high-level business objectives (e.g. locality, makespan)
  - These criteria influence the objective function and trade-off preferences used by the optimiser.
- Transportation Parameters:
  - A mapping of TransportationTime for each known provider or company. This represents the expected time delay between the consumer and each manufacturing site, affecting locality, delivery feasibility, and fairness considerations.

Platform-side Input.

- A list of Manufacturing Services required (e.g., Milling, Drilling), in the order that is required to be followed in manufacturing.

These inputs define the scheduling constraints and preferences for each job and guide how the optimising service prioritises scheduling decisions across multiple requests.

Provider-side Input. Each machine registered by a provider (e.g., Tekniker, Moldes URA, Erreka) must include the following details:

- Identification:

    - The Company that owns or operates the machine
    - A unique Machine identifier
- Service Capabilities:
    - A list of all the service types the machine can perform
    - A processing time for each service of the machine
- Availability Constraints:
    - A list of Available Time Windows specifying the operational periods for each machine. These windows may be continuous (e.g., open-ended availability starting at a certain time) or segmented (e.g., multiple specific time intervals over different days).

This information allows the optimising service to assess machine eligibility for each operation, determine time feasibility under the request's delivery window, and respect scheduling constraints such as downtime or machine overload.

Together, this consumer-side and provider-side data supports the construction of a flowshop-style scheduling model, where jobs are routed through a series of operations on eligible machines within defined time windows, subject to business and logistical objectives.

2.2.4 Results

For the first phase of experimentation, the optimisation service was tested on synthetically generated datasets. Although randomly created, these datasets were reviewed by the collaborating partners to ensure the embedded assumptions remain realistic and industry-relevant. To evaluate the performance of the CP model, two experimental procedures were conducted:
(a) The model was first solved using a single-objective formulation focused on minimising the makespan. This allowed for the calculation of the **optimality gap**, offering a mathematically sound estimate of the solution quality with respect to the theoretical optimum.
(b) In the second round, a **Pareto frontier** was constructed based on the two-objective formulation (makespan and locality weight). This analysis aimed to identify values of the scalarisation coefficient α that yield a balanced trade-off between the objectives, i.e., configurations where neither objective dominates the other disproportionately.

Generation of instances. The use-case ecosystem comprises 24 parallel machines distributed across three providers: Moldes URA, Erreka, and Tekniker. Each machine supports a predefined set of eligible operations, such as Additive Manufacturing, Machining, and Plastic Injection Moulding. For each operation, a rate-per-minute (rpm) weight is assigned, which determines the processing duration of scheduled tasks. The machine configuration remains consistent across all datasets.

Two request volumes are considered: 5 and 10. Each request contains a randomly selected number of parts, ranging from 5 to 20. Additionally, for every request, a random subset of services is drawn from those available across the machines. Three service levels are defined for the maximum number of required services per request: 2, 3, or 4. This selection dictates how many different services must be applied to each part, effectively defining the cardinality of the service set J_i for each part *i.*

Nominal processing times for parts are sampled from a uniform integer distribution between 20 and 100 minutes. These values are then adjusted per job *j* based on the rpm weight of each eligible machine *k*, yielding p_{jk}. Transportation times depend on provider relationships. Transfers between machines from different providers range between 1 to 5 hours (converted to 60–300 minutes), while intra-provider transfers are shorter, between 5 and 15 minutes. Furthermore, initial transportation from the request origin to machines is also modeled as a locality weight, ranging from 1 to 5 hours (60–300 minutes).

Table 3. Datasets generator rules

Parameter	Rule
$\|R\|$	{5, 10}
$\|I_r\|$	integer(5, 20)
$\|J_i\|$	{2, 3, 4}
$Nominal\ processing\ time$	integer(20, 100)
f_{rk}	integer(1, 5) x 60

Table 3 summarises the rules used to generate the requests. A total of 24 instances are created by systematically combining the number of requests (5 or 10) with the maximum number of operations per request (2, 3, or 4), with each combination replicated four times. Instances 1–12 contain 5 requests each, while instances 13–24 contain 10 requests. Within these groups, instances 1–4 and 13–16 are limited to a maximum of 2 operations per request, instances 5–8 and 17–20 allow up to 3 operations, and instances 9–12 and 21–24 permit up to 4 operations per request.

Assessment of the CP model. In the first round of experiments, the CP model was solved for each of the 24 generated problem instances. A time limit of 1800 seconds was applied to each run. If an optimal solution was not found within this time frame, the best solution identified so far was returned. The primary metric used to evaluate solution quality is the optimality gap. As an exact optimisation method, CP calculates a lower bound (LB) - a valid estimate that is guaranteed to be less than or equal to the true optimal value - and an upper bound (UB), representing the objective value of the best solution found. The optimality gap is defined as $\frac{UB-LB}{LB}\%$. A smaller optimality gap indicates that the returned solution is close to optimal, and thus of high quality.

Table 4 summarises the results of the CP model for the 24 datasets:

Table 4. Results of experiments on CP model 2

Instance #	Lower Bound	Upper Bound	Gap (%)	Time (s)
1	**1616**	**1616**	**0.00**	**988**
2	**1607**	**1607**	**0.00**	**580**
3	**919**	**919**	**0.00**	**657**
4	**1600**	**1600**	**0.00**	**236**

Instance #	Lower Bound	Upper Bound	Gap (%)	Time (s)
5	**1653**	**1653**	**0.00**	**178**
6	**1810**	**1810**	**0.00**	**800**
7	**1938**	**1938**	**0.00**	**839**
8	1688	1729	2.37	1800
9	1503	1894	20.64	1800
10	1639	1699	3.53	1800
11	1870	1882	0.64	1800
12	1680	1782	5.72	1800
13	**1875**	**1875**	**0.00**	**1017**
14	1211	1421	14.78	1800
15	1811	2190	17.31	1800
16	1577	1617	2.47	1800
17	**1607**	**1607**	**0.00**	**921**
18	1811	2224	18.57	1800
19	1808	1848	2.16	1800
20	**1721**	**1721**	**0.00**	**1148**
21	1802	1941	7.16	1800
22	1822	2079	12.36	1800
23	1988	2280	12.81	1800

Instance #	Lower Bound	Upper Bound	Gap (%)	Time (s)
24	1843	2383	22.66	1800

The CP model successfully computed the optimal solution for 10 out of the 24 datasets, including all six of the smallest instances. For an additional 5 datasets, the optimality gap remained below 5%, indicating near-optimal solutions. As the problem scale increases - measured by the number of jobs and requests - the model's performance tends to decline, as expected. Nonetheless, even the largest observed optimality gaps remain within a reasonable range (i.e., 22.66% for dataset 24 and below 20% for the rest of the datasets), demonstrating that the CP model consistently produces high-quality solutions across varying instance sizes.

A Pareto frontier for the two objectives. To simultaneously minimise makespan and locality weight, a Pareto frontier was constructed to provide quantitative insight into the trade-off between these two objectives. This was achieved using a scalarisation approach, where the CP model was solved iteratively for varying values of the weighting coefficient α. Starting from 1 and decreasing to 0 in fixed steps of 0.05, each iteration applied a different weighting: α for makespan and 1-α for locality weight. This process generated a set of solutions that collectively form an approximate Pareto frontier, illustrating how improvements in one objective typically come at the expense of the other. Table 4 presents the minimum and maximum values of each objective in the total of 21 iterations (all values of α from 0 to 1 stepped by 0.05). The distance between the extreme values is denoted by the value of *Err* = $\frac{max-min}{min}$%.

Table 5. Pareto frontier of Makespan/Locality weight

Instance #	Makespan			Locality value		
	min value	max value	Err (%)	min value	max value	Err (%)
1	1616	1814	12.25	20040	21180	5.69
2	1607	5905	267.45	21480	24180	12.57
3	919	2102	128.73	32760	32760	0.00
4	1600	5742	258.88	17280	18840	9.03
5	1653	5903	257.11	20400	24660	20.88
6	1810	5898	225.86	21420	26700	24.65
7	1938	5708	194.53	32820	35940	9.51
8	1730	5897	240.87	38280	42660	11.44
9	1894	5780	205.17	52020	52860	1.61
10	1699	5215	206.95	36480	37860	3.78
11	1882	5809	208.66	33000	33480	1.45
12	1782	5861	228.9	38400	42600	10.94
13	1875	5781	208.32	27600	28560	3.48
14	1421	2093	47.29	45780	48660	6.29
15	2190	2980	36.07	49620	50100	0.97
16	1617	5369	232.03	42360	43560	2.83
17	1607	5788	260.17	46560	51120	9.79
18	2224	5719	157.15	71100	76080	7.00
19	1848	8758	373.92	45660	50160	9.86
20	1721	6005	248.93	53220	59040	10.94

Instance #	Makespan			Locality value		
	min value	max value	Err (%)	min value	max value	Err (%)
21	1941	6441	231.84	93000	100740	8.32
22	2079	5740	176.09	87060	97560	12.06
23	2280	6441	182.5	110940	117840	6.22
24	1843	8065	337.6	76560	81360	6.27

Notably, the makespan objective exhibits greater sensitivity to variations in the weighting coefficient α, as its values show a wider range across the Pareto frontier compared to those of the locality weight. This indicates that small changes in α have a more pronounced effect on makespan than on locality. As a result, assigning higher values to α tends to stabilise the trade-off, often yielding solutions where neither objective dominates excessively - particularly useful when aiming for a more balanced compromise between production efficiency and provider locality.

Graph of Figure 4 supports this observation. For each value of α, the figure displays the average error percentage (Err %), calculated as the sum of the average relative errors for makespan and locality weight across all instances. Notably, the average Err % stabilizes for values of α greater than 0.45, indicating reduced variability in the trade-off performance. The lowest average errors are observed in the range of α from 0.85 to 0.95, suggesting that solutions generated with a strong emphasis on makespan tend to offer the most efficient balance in terms of overall objective quality.

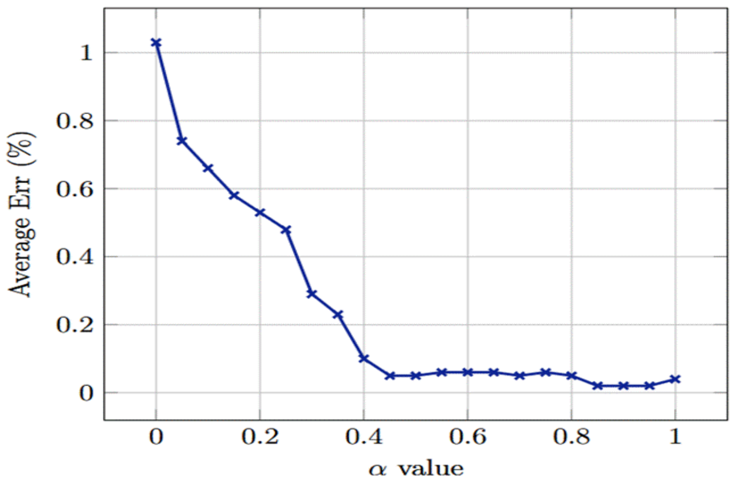


Figure 4. Average Error per α value

An illustrative example. To provide a clearer view of the operation of the optimisation service, an illustrative example is presented in detail. The applied operations are Milling, Turning, Drilling, Threading, Grinding, Additive Manufacturing, Plastic Injection Moulding, Variothermic Heating, Compression Technology, Assembly of components and SEMD - enumerated from 1 to 11 respectively. Table 6 lists the available machines, along with the eligible operations and their nominal processing time in minutes:

Table 6. List of machines - operations eligibilities

Machine\Operation	1	2	3	4	5	6	7	8	9	10	11
Tekniker_Machine_1	9	9	12	12	-	-	-	-	-	-	-
Tekniker_Machine_2	5	-	10	10	-	-	-	-	-	-	-

Machine\Operation	1	2	3	4	5	6	7	8	9	10	11
Tekniker_Machine_3	12	-	15	15	-	-	-	-	-	-	-
Tekniker_Machine_4	-	-	-	-	6	-	-	-	-	-	-
Tekniker_Machine_5	-	-	-	-	-	3	-	-	-	-	-
Tekniker_Machine_6	-	-	-	-	-	10	-	-	-	-	-
Erreka_Machine_1	-	-	-	-	-	-	8	-	-	-	-
Erreka_Machine_2	-	-	-	-	-	-	4	-	-	-	-
Erreka_Machine_3	-	-	-	-	-	-	6	15	15	-	-
Erreka_Machine_4	-	-	-	-	-	-	3	-	-	-	-
Erreka_Machine_5	-	-	-	-	-	-	12	-	-	-	-
Erreka_Machine_6	-	-	-	-	-	-	3	-	-	-	-
Erreka_Machine_7	-	-	-	-	-	-	-	-	-	11	-
URA_Machine_1	9	-	15	15	-	-	-	-	-	-	-
URA_Machine_2	7	-	12	12	-	-	-	-	-	-	-
URA_Machine_3	8	-	20	20	-	-	-	-	-	-	-
URA_Machine_4	15	-	19	19	-	-	-	-	-	-	-
URA_Machine_5	8	-	12	12	-	-	-	-	-	-	-
URA_Machine_6	10	-	15	15	-	-	-	-	-	-	-
URA_Machine_7	-	19	-	-	-	-	-	-	-	-	-
URA_Machine_8	-	-	-	-	-	-	-	-	-	-	12
URA_Machine_9	-	-	-	-	-	-	-	-	-	-	11
URA_Machine_10	-	-	-	-	-	-	-	-	-	-	20
URA_Machine_11	-	-	-	-	-	-	-	-	-	-	5

Each machine has a set of time windows of availability. Given that the time instance of scheduling is set to 0, the time windows of the example are as presented in Table 7 (in minutes after time instance 0):

Table 7. Time windows of machine availability

Machine	Time windows		
Tekniker_Machine_1	(1500, infinity)		
Tekniker_Machine_2	(1500, infinity)		
Tekniker_Machine_3	(60, 720)	(720, 1320)	(1500, infinity)
Tekniker_Machine_4	(600, infinity)		
Tekniker_Machine_5	(60, 900)	(1020, 1200)	(1800, infinity)
Tekniker_Machine_6	(60, 600)	(1320, infinity)	
Erreka_Machine_1	(60, 900)	(1020, 1200)	(1800, infinity)
Erreka_Machine_2	(60, 600)	(1200, infinity)	
Erreka_Machine_3	(600, 960)	(1200, 1500)	(1800, infinity)
Erreka_Machine_4	(600, 1020)	(1200, 1500)	(1800, infinity)

Machine	Time windows		
Erreka_Machine_5	(600, infinity)		
Erreka_Machine_6	(60, 480)	(600, 1140)	(1800, infinity)
Erreka_Machine_7	(600, infinity)		
URA_Machine_1	(60, 360)	(600, infinity)	
URA_Machine_2	(300, 1200)	(1440, infinity)	
URA_Machine_3	(540, 1200)	(1320, infinity)	
URA_Machine_4	(60, 120)	(900, 1140)	(1260, infinity)
URA_Machine_5	(60, 120)	(1260, infinity)	
URA_Machine_6	(900, 1140)	(1260, infinity)	
URA_Machine_7	(60, 120)	(1800, infinity)	
URA_Machine_8	(600, 960)	(1800, infinity)	
URA_Machine_9	(600, 960)	(1200, 1500)	(1800, infinity)
URA_Machine_10	(600, 960)	(1800, infinity)	
URA_Machine_11	(1200, 1500)	(1800, infinity)	

For the five generated requests, the following parameters are considered:

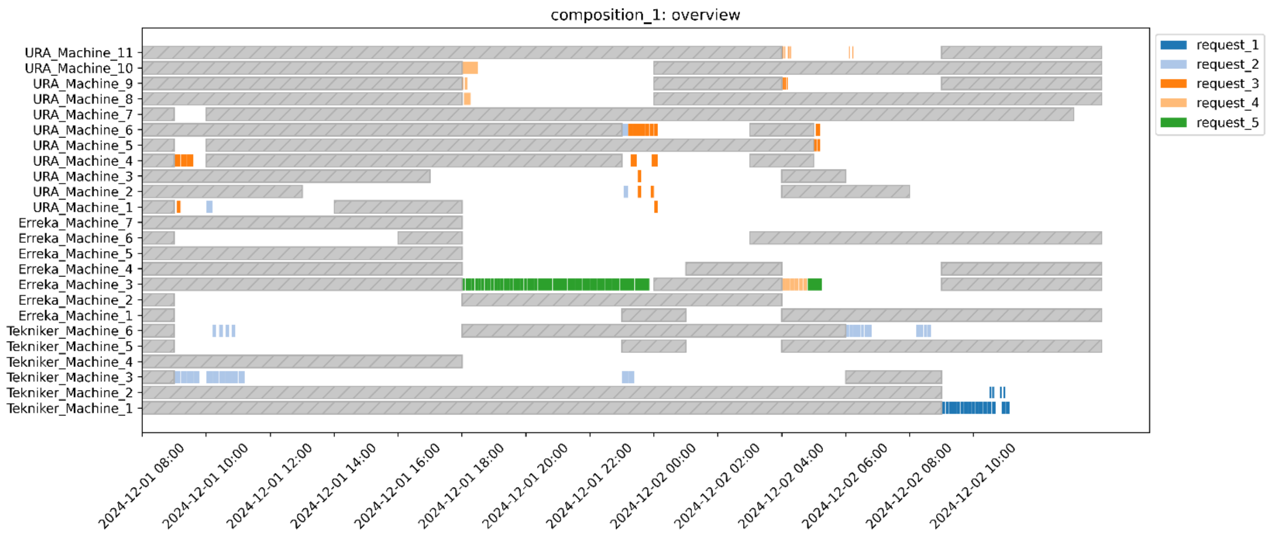
Table 8. Generated requests

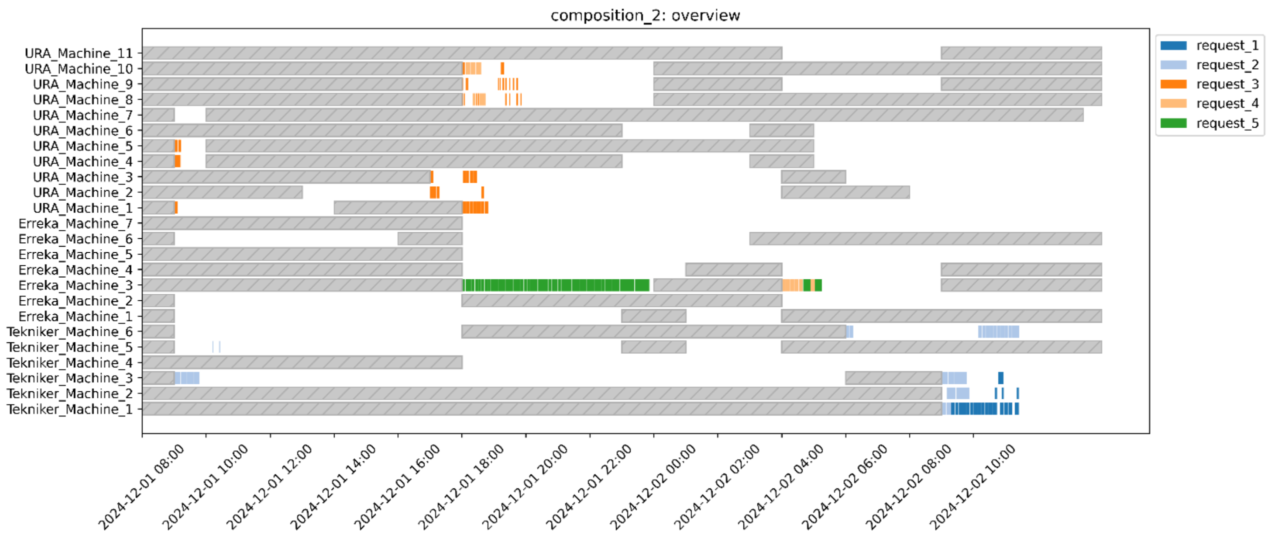
Request	Parts	Operations	Parameters
1	10	[2, 1]	[0.85, 0.89]
2	15	[3, 6]	[0.83, 0.79]
3	20	[1, 11]	[0.86, 0.32]
4	6	[11, 9]	[0.50, 0.54]
5	19	[8, 9]	[0.44, 0.95]

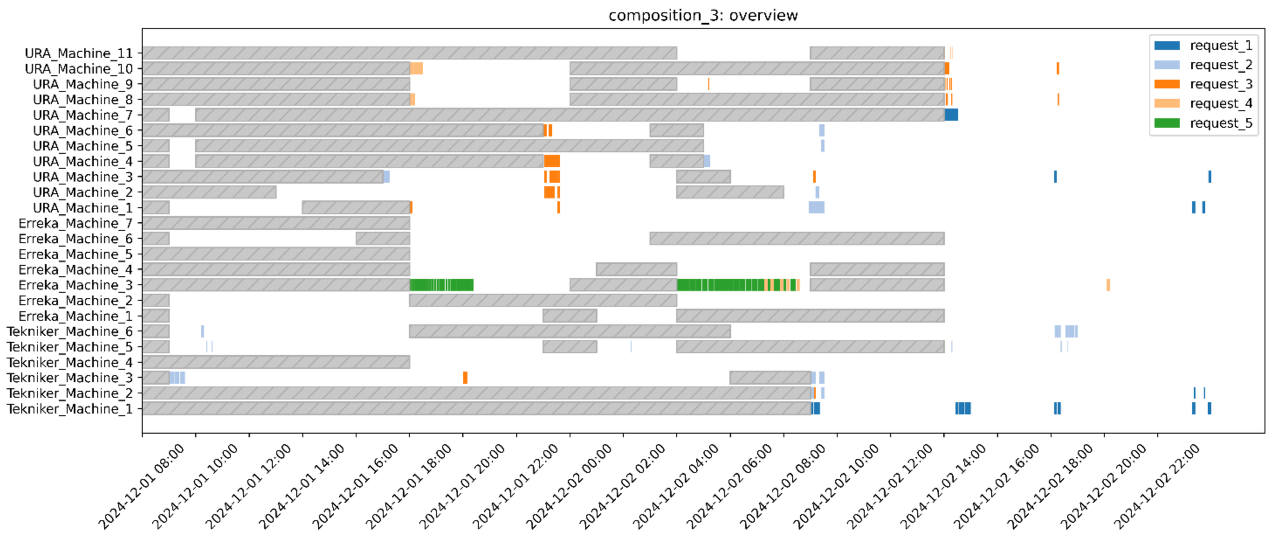
The sequence of operations listed in the "Operations" column indicates the scheduling order for each part within a request. To compute the processing times p_{jk}, the nominal values provided in Table 6 are multiplied by a machine-specific adjustment factor from Table 8. This adjustment accounts for variations in machine performance or operational efficiency across different providers.

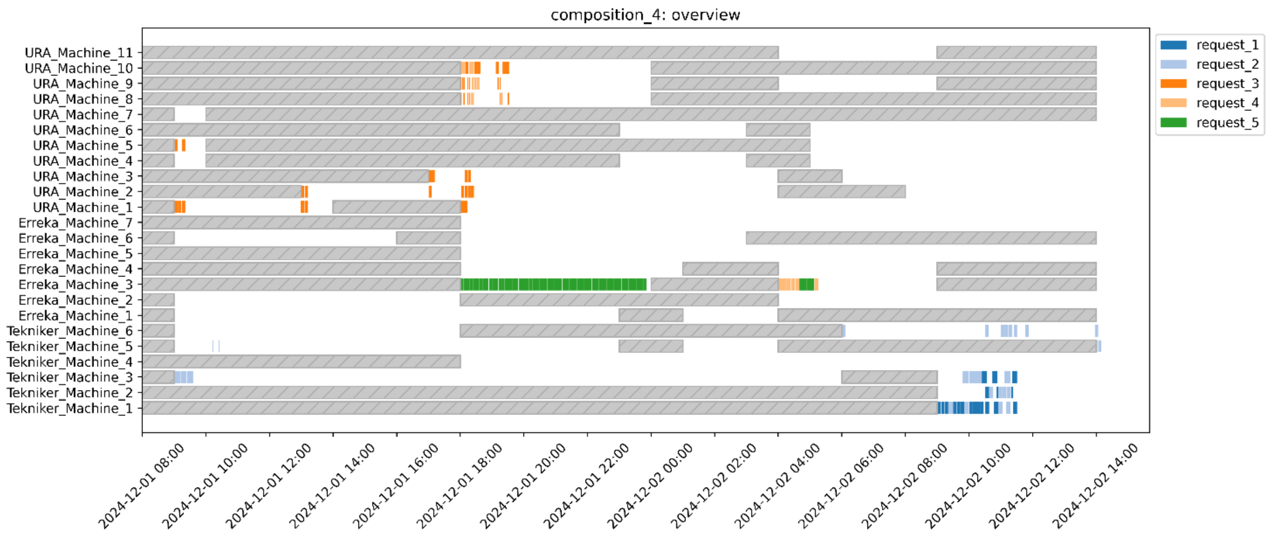
As described in Section 2.2.2, the optimisation service generates five alternative compositions on the same sets of requests and machines. The computed schedules can be monitored from three perspectives: the consumer, the provider and the platform.

Figure 5 presents the alternative compositions from the perspective of the platform. Each colour represents a different request:


composition_1: overview
URA_Machine_11
URA_Machine_10
URA_Machine_9
URA_Machine_8
URA_Machine_7
URA_Machine_6
URA_Machine_5
URA_Machine_4
URA_Machine_3
URA_Machine_2
URA_Machine_1
Erreka_Machine_7
Erreka_Machine_6
Erreka_Machine_5
Erreka_Machine_4
Erreka_Machine_3
Erreka_Machine_2
Erreka_Machine_1
Tekniker_Machine_6
Tekniker_Machine_5
Tekniker_Machine_4
Tekniker_Machine_3
Tekniker_Machine_2
Tekniker_Machine_1
request_1
request_2
request_3
request_4
request_5
2024-12-01 08:00
2024-12-01 10:00
2024-12-01 12:00
2024-12-01 14:00
2024-12-01 16:00
2024-12-01 18:00
2024-12-01 20:00
2024-12-01 22:00
2024-12-02 00:00
2024-12-02 02:00
2024-12-02 04:00
2024-12-02 06:00
2024-12-02 08:00
2024-12-02 10:00


composition_2: overview
URA_Machine_11
URA_Machine_10
URA_Machine_9
URA_Machine_8
URA_Machine_7
URA_Machine_6
URA_Machine_5
URA_Machine_4
URA_Machine_3
URA_Machine_2
URA_Machine_1
Erreka_Machine_7
Erreka_Machine_6
Erreka_Machine_5
Erreka_Machine_4
Erreka_Machine_3
Erreka_Machine_2
Erreka_Machine_1
Tekniker_Machine_6
Tekniker_Machine_5
Tekniker_Machine_4
Tekniker_Machine_3
Tekniker_Machine_2
Tekniker_Machine_1
request_1
request_2
request_3
request_4
request_5
2024-12-01 08:00
2024-12-01 10:00
2024-12-01 12:00
2024-12-01 14:00
2024-12-01 16:00
2024-12-01 18:00
2024-12-01 20:00
2024-12-01 22:00
2024-12-02 00:00
2024-12-02 02:00
2024-12-02 04:00
2024-12-02 06:00
2024-12-02 08:00
2024-12-02 10:00


composition_3: overview
URA_Machine_11
URA_Machine_10
URA_Machine_9
URA_Machine_8
URA_Machine_7
URA_Machine_6
URA_Machine_5
URA_Machine_4
URA_Machine_3
URA_Machine_2
URA_Machine_1
Erreka_Machine_7
Erreka_Machine_6
Erreka_Machine_5
Erreka_Machine_4
Erreka_Machine_3
Erreka_Machine_2
Erreka_Machine_1
Tekniker_Machine_6
Tekniker_Machine_5
Tekniker_Machine_4
Tekniker_Machine_3
Tekniker_Machine_2
Tekniker_Machine_1
request_1
request_2
request_3
request_4
request_5
2024-12-01 08:00
2024-12-01 10:00
2024-12-01 12:00
2024-12-01 14:00
2024-12-01 16:00
2024-12-01 18:00
2024-12-01 20:00
2024-12-01 22:00
2024-12-02 00:00
2024-12-02 02:00
2024-12-02 04:00
2024-12-02 06:00
2024-12-02 08:00
2024-12-02 10:00
2024-12-02 12:00
2024-12-02 14:00
2024-12-02 16:00
2024-12-02 18:00
2024-12-02 20:00
2024-12-02 22:00



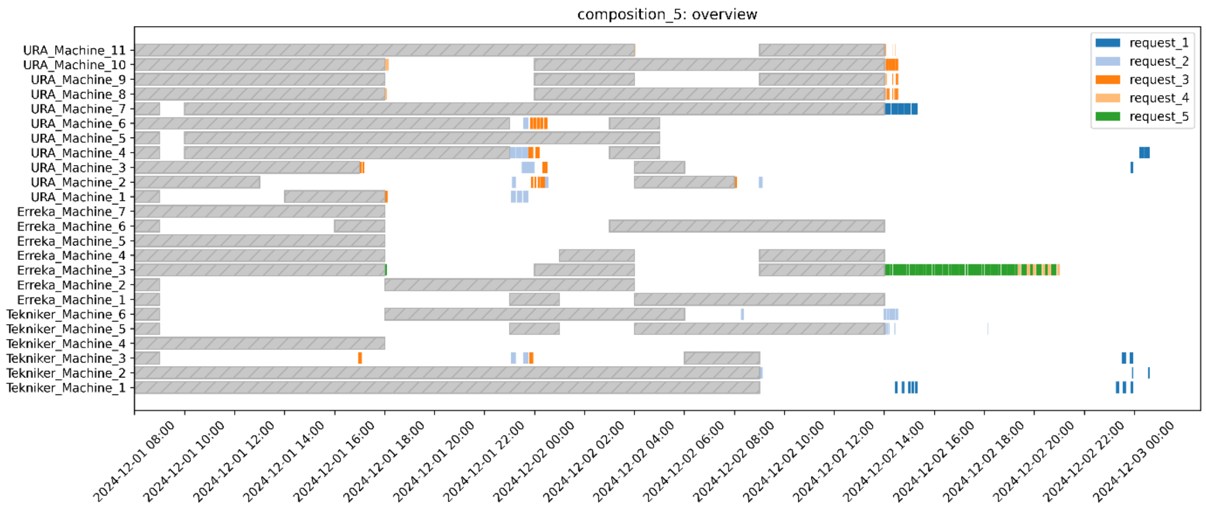


Figure 5. View of the compositions from the perspective of the platform

2.3 Next steps

The next steps in integrating the proposed methodology focus on scaling the applied datasets - particularly for VN2, where small-scale random benchmarks have been examined so far. Upcoming experimentation may reveal limitations in the current framework, thereby motivating the development of alternative optimisation methods and deeper integration with the Task Batching Service. These methods could combine CP models with Large Neighbourhood Search (LNS) or PSO algorithms. Additionally, incorporating Reinforcement Learning (RL) techniques may enhance the efficiency of the optimisation service, especially in improving its ability to adapt and refine existing schedules in response to urgent disruptions.

3 Optimisation service: Functional requirements

3.1 Use-cases

The use-cases of the shell are listed below:

● Use-case 1 (UC1):	Authenticate user.
● Use-case 2 (UC2):	List the available optimisation services.
● Use-case 3 (UC3):	Submit a new optimisation job.
● Use-case 4 (UC4):	Get the solution of a completed optimisation job.
● Use-case 5 (UC5):	Store the solution of a completed optimisation job.

The listed functional requirements are also demonstrated in the image below:

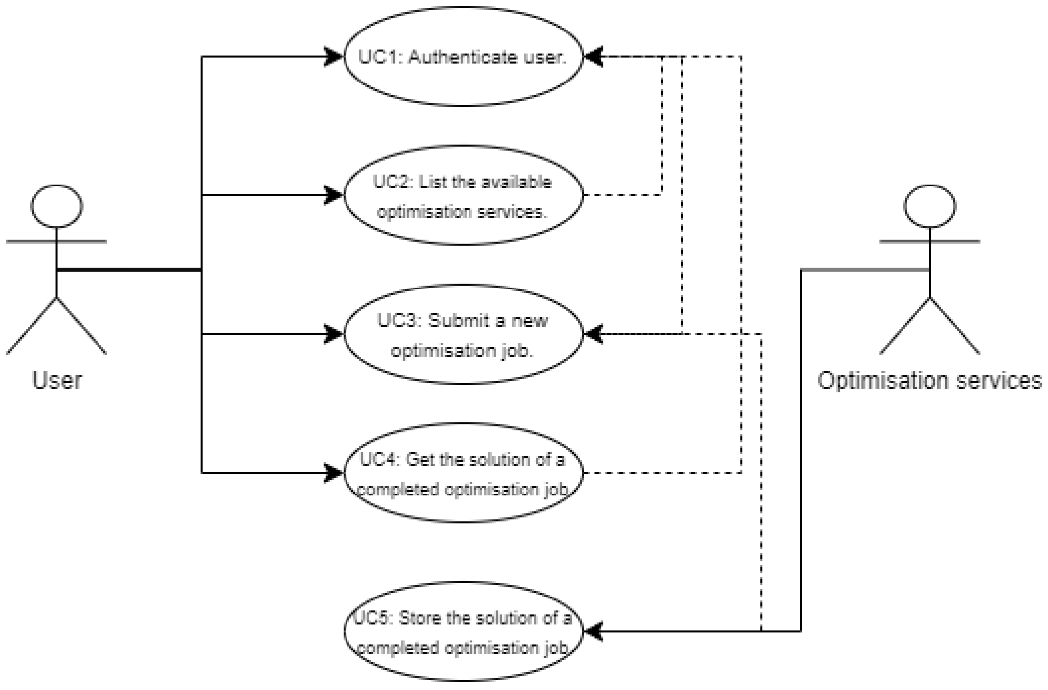

UC1: Authenticate user.
UC2: List the available optimisation services.
UC3: Submit a new optimisation job.
UC4: Get the solution of a completed optimisation job.
UC5: Store the solution of a completed optimisation job.
User
Optimisation services

Figure 6. optEngine functional requirements

3.2 Architectural view

OptEngine works as a shell around the optimisation. Its architecture focuses on both **asynchronous** and **synchronous (real-time) optimisation requests,** broadening the usability of the tool in time-sensitive scenarios. *OptEngine* is agnostic to optimisation-specific data requirements, receiving and forwarding the optimisation data to the optimisation service requested by the end-user, then it stores the computed solution to the Dataspace.

Optimisation requests along with the respective data are received via a web API – the process of posting requests and retrieving their solutions is described in D4.1. The communication with the API requires authentication, is encrypted (https) and asynchronous, meaning that the user does not have to wait for the completion of a submitted optimisation job.

Data stores within *optEngine* work in a twofold manner:

- **Permanent storage via a dataspace:** the requests and the related data are permanently stored or retrieved and updated, when necessary, from a Dataspace. For example, the Day-to-Day optimisation module stores the acquired solution to the Dataspace. If a disruption occurs during the implementation of the daily schedule, the Event-triggered optimisation module retrieves the stored solution to update it.
- **Temporal storage via the use of queues:** optimisation data are stored to the Dataspace until they are retrieved from the optimisation services that read these queues.

Regarding the optimisation data, both the dataspace and the queues are data-agnostic following a general json schema. This allows the storage, permanent and temporal, of different data structures required from different optimisation services.

The employed queues allow the asynchronous processing of an optimisation job. Additionally, by being durable they ensure that when *optEngine* or an optimisation service fail, the job along with data are available in the respective queue. This means that *optEngine*, upon reception of a new optimisation job, forwards it to the requested optimisation service via a queue. Each optimisation service listens for a new optimisation job to a specific queue and writes status/progress updates to another queue. Last, *optEngine* listens to (a) a queue for status/progress updates and (b) multiple queues for optimisation results.

The flow of data and the architectural approach of *optEngine* are depicted below.

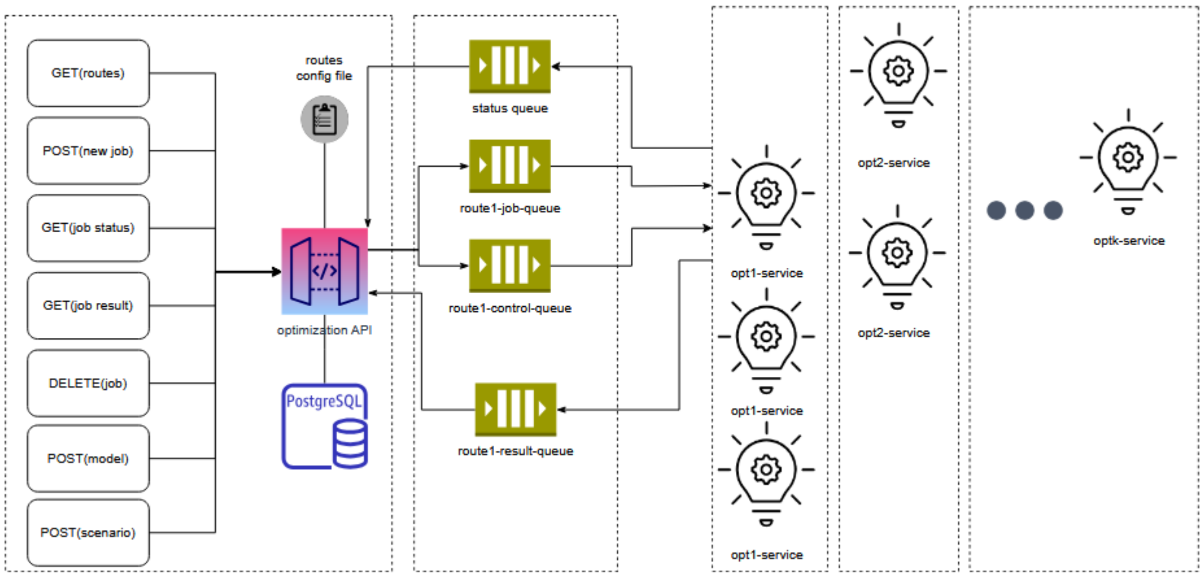


Figure 7. optEngine data flow

OptEngine requires a minimum of 2 virtual CPUs, 4 GB of RAM, and 20 GB of disk space to operate. Any optimisation service that uses OptEngine has its own hardware requirements, which are independent of those for OptEngine. The optimisation engine has been originally designed for the 2020 Horizon project FACTLOG (Grant agreement: 869951). An elaborate description of optEngine is presented in D5.1[1].

[1] https://www.factlog.eu/about/deliverables/

3.3 Technology stack

The following technologies (Table 9) are used for the development of *optEngine*, including a programming language platform (Java 8), framework for rapid Java-based application development (Spring Boot), Object-Relational Mapping framework for Java (Hibernate), standards for defining RESTful APIs and an ecosystem of tools for working with OpenAPI specs (OpenAPI, Swagger), message broker for asynchronous communication (RabbitMQ), advanced open-source relational database (PostgreSQL), and a containerization platform (Docker) to ensure consistent, portable, and isolated execution across development, testing, and production environments, facilitating seamless deployment and integration within diverse system landscapes.

Table 9. optEngine data stack

Java 8	Java 8
Spring-boot 2.4.5	spring boot
Springdoc openAPI 1.5.2	Open API Specification Swagger
Hibernate 1.0.0	HIBERNATE
PostgreSQL 11	PostgreSQL
RabbitMQ 3.8.16	RabbitMQ
Docker 18.09.7	docker

References

[1] Amirteimoori, A., Mahdavi, I., Solimanpur, M., Ali, S. S., & Tirkolaee, E. B. (2022). *A parallel hybrid PSO-GA algorithm for the flexible flow-shop scheduling with transportation*. Computers & Industrial Engineering, 173.

[2] Armstrong, E., Garraffa, M., O'Sullivan, B., & Simonis, H. (2021). *The Hybrid Flexible Flowshop with Transportation Times*. 27th International Conference on Principles and Practice of Constraint Programming, pp. 1-18.

[3] Chen, W., Feng, P., Luo, X., & Nie, L. (2024). *Task-service matching problem for platform-driven manufacturing-as-a-service: A one-leader and multi-follower Stackelberg game with multiple objectives*. Omega (129).

[4] Duan, J., Wang, M., Zhang, Q., & Qin, J. (2023). *Distributed shop scheduling: A comprehensive review on classifications, models and algorithms*. Mathematical Biosciences and Engineering, 20(8), pp. 15265-15308.

[5] Hamdan, S., Cheaitou, A., Shikhli, A., & Alsyouf, I. (2023). *Comprehensive quantity discount model for dynamic green supplier selection and order allocation*. Computers & Operations Research (160).

[6] Karamanli, A., Xanthopoulos, A., Gasteratos, A., Koulouriotis, D. (2024). *Manufacturing-as-a-Service: A Systematic Review of the Literature*. 5th Olympus International Conference, ICSC 2024.

[7] Leng, J., Zhong, Y., Lin, Z., Xu, K., Mourtis, D., Zhou, X., Zheng, P., Liu, Q., Zhao, J.L., Shen, W. (2023). *Towards resilience in Industry 5.0: A decentralized autonomous manufacturing paradigm*. Journal of Manufacturing Systems, 71, pp. 95-114.

[8] Lee, J., Kao, H.-A., & Yang, S. (2014). *Service Innovation and Smart Analytics for Industry 4.0 and Big Data Environment*. Procedia CIRP, 16, pp. 3-8.

[9] Medeiros, G. H., Cao, Q., Zanni-Merk, C., & Samet, A. (2020). *Manufacturing as a Service in Industry 4.0: A Multi-Objective Optimisation Approach*. Intelligent Decision Technologies, pp. 37-47.

[10] Mollaei, A., Mohammadi, M., & Naderi, B. (2019). *A bi-objective MILP model for blocking hybrid flexible flow shop scheduling problem: robust possibilistic programming approach*. International Journal of Management Science and Engineering Management, 14(2).

[11] Naderi, B., Gohari, S., & Yazdani, M. (2014). *Hybrid flexible flowshop problems: Models and solution methods*. Applied Mathematical Modeling, 38(24), pp. 5767-5780.

[12] Pulkkinen, A., Nagarajan, H. P., & Heilala, J. (2024). *A Review of Modelling Methods for Manufacturing as a Service - Case Additive Manufacturing*. Advances in Transdisciplanry Engineering (52), pp. 313-321.

[13] Schöppenthau, F., Patzer, F., Schnebel, B., Watson, K., Baryschnikov, N., Obst, B., Chauhan, Y., Kaever, D., Usländer, T., Kulkarni, P. (2023). *Building a Digital Manufacturing as a Service Ecosystem for Catena-X*. Sensors, 23(17).

[14] Shabtay, D. (2012). *The just-in-time scheduling problem in a flow-shop scheduling system*. European Journal of Operational Research, 216(3), pp. 521-532.

[15] Shen, L., Sun, W., & Parida, V. (2023). *Consolidating digital servitization research: A systematic review, integrative framework, and future research directions*. Technological Forecasting and Social Change, 191.

[16] Vatikiotis, S., Mpourdakos, I., Papathanasiou, D., & Mourtos, I. (2024). *Makespan Minimisation in Hybrid Flexible Flowshops with Buffers and Machine-Dependent Transportation Times*. Advances in Production Management Systems, pp. 258-273.

[17] Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A., Lenstra, J. K., Sevast'janov, S. V., & Shmoys, D. B. (1997). *Short Shop Schedules*. Operations Research, 45(2), pp. 288-294.

[18] Zhang, L., Luo, Y., Tao, F., Hu Li, B., Ren, L., Zhang, X., . . . Liu, Y. (2014). *Cloud manufacturing: a new manufacturing paradigm*. Enterprise Information Systems, 8(2).