



## **Production harmonizEd Reconfiguration of Flexible Robots and Machinery**

Horizon 2020 – Factories of the Future, Project ID: 680435

### **Deliverable 4.2**

## **Planning procedures for energy and agent-based planning and (re)-scheduling**

Lead Author: Loughborough University

Dissemination level: PUBLIC  
Date: 09/05/2018  
Revision: 1.0

## Version history

Version	Date	notes and comments
0.1	26/02/2018	Report structure & table of content by LBORO
0.2	30/02/2008	Completion of Chapter One by LBORO
0.3	15/03/2018	Completion of Chapter Two and Three by LBORO
0.4	30/03/2018	Completion of Chapter Four & Five by LBORO
0.5	04/04/2018	First draft ready for proofreading and internal reviews (TUBS)
0.6	04/05/2018	Incorporation of feedback and comments from TUBS
1.0	09/05/2018	Final proofreading

### Author List:

Phil Ogun (LBORO)  
Lennart Bueth (TUBS)  
Niels Lohse (LBORO)  
Sebastian Thiede (TUBS)

## Abstract

Flexible job-shop scheduling problem is one of the well-known combinatorial optimisation problems and it has been studied extensively due its practical importance. Traditional centralised production planning and scheduling systems can provide optimal solutions with respect to certain performance measures such as makespan, tardiness and energy consumption. However, they are inflexible and are unable to cope with real-world environments that are characterised by complexities, dynamic alterations and unanticipated changes in the production conditions. Multi-agent-based scheduling technology provides a promising way to address dynamic changes and unpredictable disturbances in the manufacturing shop floor without disruption in production. Although agent-based technology has been proven to be an appropriate solution for handling real-time and on-demand changes, optimal production schedules cannot be guaranteed.

The work reported in this report is focused on the harmonization and standardization of techniques for achieving both flexible and optimal or near-optimal planning and scheduling systems. The performance measures to be minimised are energy consumption and tardiness costs. The aim is to generate an optimal schedule based on ideal production circumstances. Instead of using a centralised control system to execute the schedule, multi-agent systems are deployed. Decision-making policies that follow the selected schedule are built into the agents and they attempt to execute the preliminary policies as much as possible. In the event of machine breakdown or any other unforeseen events, the agents react quickly by allocating parts to other available resources without disruptions in production. This hybrid solution attempts to produce optimal schedules with respect to the chosen performance measures and also respond to emerging system behaviors that cannot be precisely predicted in advance.

There are two aspects to the proposed solution. The first is the design of an effective genetic algorithm for the flexible job-shop scheduling problem. Although there are no guarantee that stochastic and heuristic search methods will produce optimal schedules, they are known to give satisfactory results since they do not need to evaluate all the feasible search space to extract good solutions. Secondly, an agent-based simulator that uses the initial schedule data as input is implemented. Although the developed tool and methods can be generalised, simulation experiments and results based on the GKN use case are presented.

## Table of Contents

<b>1. INTRODUCTION .....</b>	<b>7</b>
1.1. STRUCTURE OF THE REPORT.....	7
1.2. PROBLEM DEFINITION .....	7
1.3. AIMS AND OBJECTIVES OF TASK 4.2 .....	8
<b>2. REQUIREMENTS SPECIFICATION AND SYSTEM ANALYSIS .....</b>	<b>9</b>
2.1. SHOP FLOOR LAYOUT .....	9
2.2. FLEXIBILITY REQUIREMENTS .....	11
2.3. PERFORMANCE EVALUATION CRITERION .....	12
2.4. HYBRID CENTRALISED AND AGENT-BASED SCHEDULING .....	13
2.5. DATA MODEL .....	15
2.6. AGENTS MODELLING, SIMULATION AND DEVELOPMENT TOOLS.....	17
<b>3. FLEXIBLE JOB-SHOP OPTIMISATION .....</b>	<b>22</b>
3.1. PROBLEM FORMULATION .....	22
3.2. GENETIC ALGORITHM .....	23
<b>4. AGENT-BASED SCHEDULING MECHANISM DESIGN .....</b>	<b>30</b>
4.1. AGENTS AND CHARACTERISTICS .....	31
4.2. COMMUNICATION ACTS.....	33
4.3. INTERACTION PROTOCOLS .....	35
4.4. AGENTS STATES, TRANSITIONS, ACTIONS AND EVENTS.....	38
4.5. ROUTING AND SEQUENCING RULES .....	41
<b>5. SIMULATOR IMPLEMENTATION AND RESULTS .....</b>	<b>43</b>
5.1. MODEL CLASSES.....	43
5.2. SIMULATION RESULTS.....	49
<b>6. CONCLUSIONS.....</b>	<b>60</b>
<b>7. REFERENCES.....</b>	<b>61</b>

## 1. List of Figures

Figure 1 Micro-Flow Cell [14] .....	9
Figure 2 Hybrid cellular and process machine layout .....	10
Figure 3 Parts flow diagram through workstations and Micro-Flow cells .....	12
Figure 4 Hybrid centralised and agent-based scheduling system .....	13
Figure 5 MES architecture for the hybrid scheduling system .....	14
Figure 6 Data model for Task 4.2 .....	15
Figure 7 FIPA Agent management reference model .....	18
Figure 8 JADE architectural elements .....	19
Figure 9 Components of a WADE-based application .....	20
Figure 10 Multi-agent architecture for scheduling .....	33
Figure 11 Request interaction protocol .....	36
Figure 12 Brokering interaction protocol .....	37
Figure 13 Contract net interaction protocol .....	38
Figure 14 Resource agent state chart.....	39
Figure 15 Part agent state chart.....	40
Figure 16 Simulator main page .....	45
Figure 17 Plant layout in simulator .....	46
Figure 18 Resource agent state transition during simulation .....	47
Figure 19 Part agent state transition during simulation .....	48
Figure 20 Gantt chart for optimised schedule .....	54
Figure 21 Resources states chart for optimised schedule .....	55
Figure 22 Gantt chart for non-optimised schedule.....	56
Figure 23 Resources states chart for non-optimised schedule .....	57
Figure 24 Gantt chart for schedule under dynamic resource failure conditions .....	58
Figure 25 Resources states chart for schedule under dynamic failure conditions .....	59

## 2. List of Tables

Table 1 Operations and machines assignment data .....	23
Table 2 Chromosome .....	25
Table 3 Agents tasks and required information .....	32
Table 4 Resources, operations and energy costs data .....	49
Table 5 Parts due time and tardiness costs data .....	49
Table 6 Processing time of Part 1 .....	49
Table 7 Processing time of Part 2 .....	50
Table 8 Processing time of Part 3 .....	50
Table 9 Processing time of Part 4 .....	50
Table 10 Optimal allocation of resources to parts operations generated from genetic algorithm .....	51
Table 11 Optimal schedule generated from genetic algorithm .....	51
Table 12 Schedule Gantt chart color keys .....	52
Table 13 Optimised schedule costs .....	52
Table 14 Non-optimised schedule costs .....	53

## 1. Introduction

### 1.1. Structure of the Report

This report contains the outcome of Task 4.2, titled “Planning procedures for energy and agent-based planning and (re)-scheduling”. There are five main sections in the report. Section 1 is the introductory section, which contains the problem definition and the objectives of this report. Section 2 contains requirements specification and system analysis. The details of the tools and methods developed in this task using genetic algorithm and multi-agent systems are presented in Section 3 and 4. Simulation results obtained from an agent-based simulator developed using AnyLogic software are presented in Section 5.

### 1.2. Problem Definition

Various methods for planning production systems have been reported in the literature and are also found in shop floor manufacturing execution systems. The methods differ in the number of components involved, levels of centralisation, performance measures optimisation capability, flexibility, reliability and scalability. In this task, two classes of planning and scheduling problems, namely centralised (static) and distributed (dynamic) scheduling are considered.

Classic scheduling problems are centralised and are solved using algorithms that optimise global performance objectives such as makespan, flowtime and tardiness and energy consumption [1, 2, 3, 4, 5]. Although centralised approaches can produce optimised schedules, they require accurate knowledge of the overall system states and the availability of resources throughout the execution period of a schedule is implicitly assumed. That is, the generated schedules are tied to specific resources at a particular time of the production process. This is suitable to static environments, whereas most real-world environments are characterised by unexpected disruptions and contingencies such as machine breakdowns, stochastic incoming jobs and changes in due dates. The approaches developed for solving static scheduling problems are often impractical in dynamic environments. The optimal schedule generated in advance may become obsolete or even infeasible when released to the shop floor as it does not make use of real-time information [6]. When a resource becomes unavailable or a new product is inserted into the production process, the processing plan may be subject to a major regeneration.

The evolution of manufacturing control systems from static to dynamic scheduling techniques that can provide real-time reactions to environmental changes has motivated the development of more adaptive schedulers. In recent years, multi-agent technology has been adopted for creating test beds for the examination of various planning and scheduling methods in dynamic environments [7, 8, 9]. It has been proven that agent-based schedulers provide better overall performance in environments that require reconfigurability, flexibility and reliability. In agent-based scheduling, dispatch rules or other heuristics are used to guide the agents in assigning and sequencing operations on resources in real-time instead of executing a pre-generated schedule. Decision making occurs in a distributed fashion and the final schedule emerges from the negotiation and cooperation among the autonomous agents. Agent-based planning and scheduling architecture has the advantages of flexibility, adaptability, fault-tolerance and robustness, but it also suffers a major disadvantage. There is no guarantee that the emergent schedule will be optimal due to the localised decision-making policies of the agents.

One of the strategies used to deal with uncertainties in dynamic environments is known as the predictive-reactive scheduling [10]. This strategy exploits the combined benefits of both static and distributed agent-based scheduling systems. In the strategy, a scheduler generates a preliminary schedule prior to execution based on the information available on the shop floor and some desired performance objectives. The schedule is then executed and refined by the agents in response to disturbances and other changes in the environment.

Some hybrid approaches addressing optimality and feasibility have been proposed in the literature. These solutions combine multi-agent systems with various optimisation techniques [11, 12, 13]. The approach presented in Task 4.2 consists of two steps. The first step is to generate a static optimal schedule according to the available information at the time of generating the schedule. The agents attempt to follow the schedule and then adjust promptly in the event of any disturbances. Consequently, the agents will execute the optimal schedule if there are no disturbances. The agents react accordingly in the event of any disturbance to produce a feasible schedule that is as close to optimal as possible.

### 1.3. Aims and Objectives of Task 4.2

The aim of Task 4.2 is to create a planning and scheduling architecture for a flexible manufacturing system (FMS), which will guarantee tactical allocation of resources with respect to chosen performance objectives while also ensuring real-time reactions to disturbances. The objectives towards the achievement of this aim are:

- To define a structure for linking planning and scheduling layers of production systems to higher and lower level layers
- To create planning and scheduling framework that will address:
  - Dispatch rules for optimising production performance measures
  - On-demand responses to aperiodic incoming orders and customer demands
  - Stability and adaptability in the face of disturbances such as machine breakdowns
  - Efficient use of resources and energy
- To build a simulator that will facilitate systematic evaluation of the performances of various dispatch rules and algorithms under diverse types of disruptions.



## 2. Requirements Specification and System Analysis

Industrial partners involved in the PERFoRM project have provided some relevant use cases that present both research and innovation challenges. The goal of this task is to create tools and methods that are applicable to any flexible production planning and scheduling problem. However, the GKN use case [14, 15] related to production planning and scheduling is used for problem formulation and requirements specifications. The use case can be categorised as a flexible job shop scheduling problem for a combination layout manufacturing environment.

### 2.1. Shop Floor Layout

The GKN use case presents a hybrid layout problem in which machines are arranged in a combination of cellular and process layouts. In a cellular layout, machines are grouped according to the process requirements for a set of part families that require similar processing, whereas, in a process layout, machines are arranged according to functions rather than in a cellular configuration where sequential process steps are located in close proximity.

GKN has come up with the concept of Micro-Flow Cells for making parts (Figure 1). The cells have reconfigurable modules for performing a variety of operations.

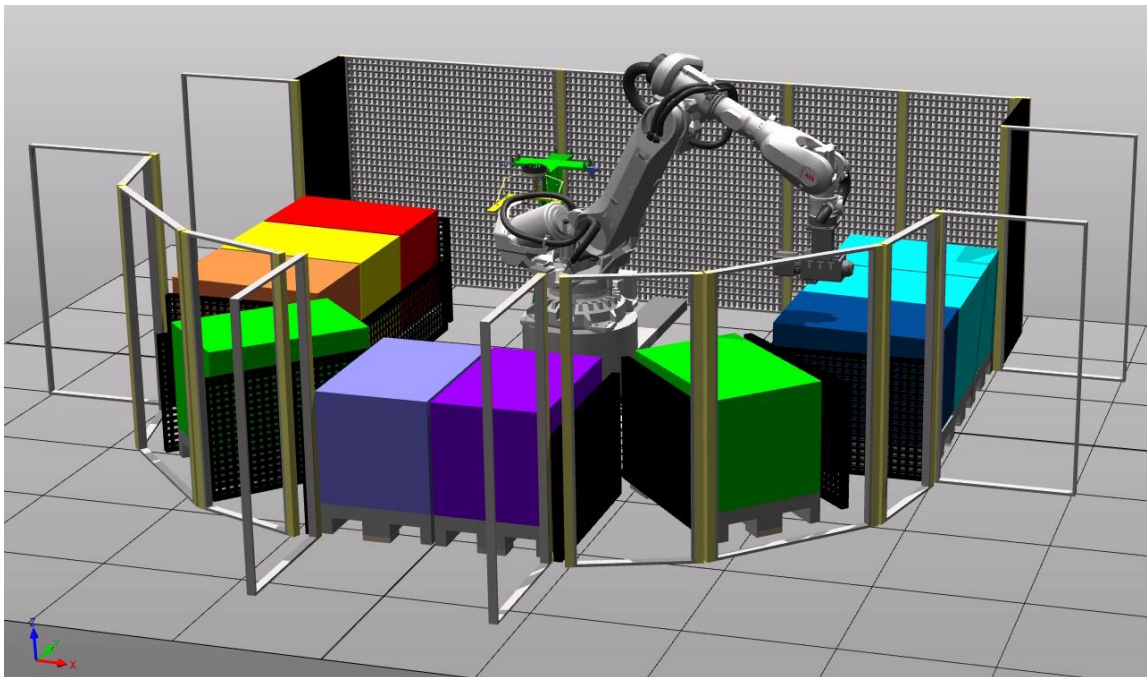


Figure 1 Micro-Flow Cell [14]

Parallel Machines – Configuration A

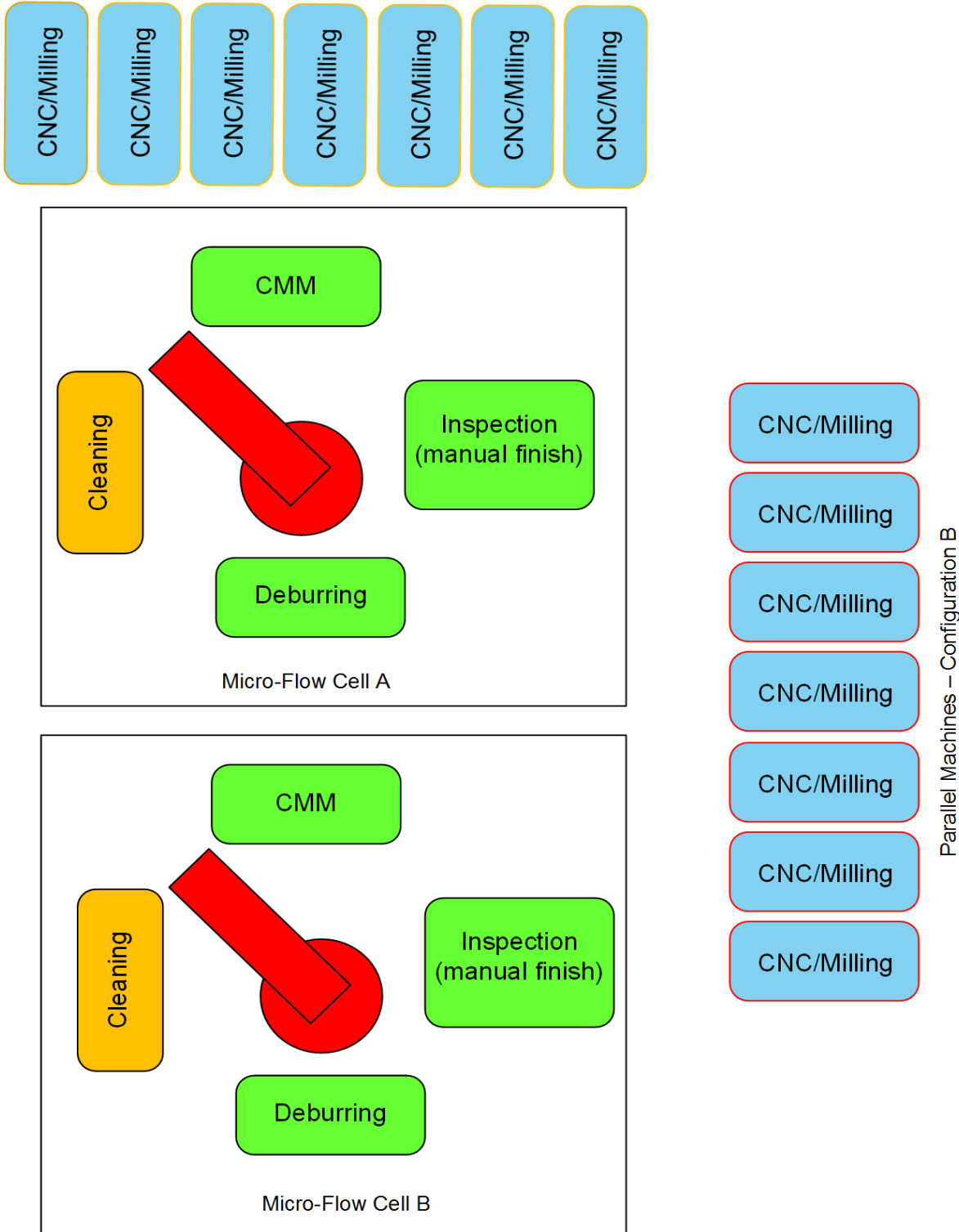


Figure 2 Hybrid cellular and process machine layout

The overall layout of the manufacturing plant is shown in Figure 2. As shown in the figure, there are workstations, each of which consists of identical parallel machine tools. The machine tools in each workstation are configured to perform similar operations. Although multiple operations can be performed in the Micro-Flow cells, each cell is effectively a single multi-purpose machine because the modules are supported by a robot. Although the literature does not reflect much research into combination layout scheduling problems, the layout is commonly used in factories where flexibility is very important. Therefore, the developed concepts are more generally applicable.

## 2.2. Flexibility Requirements

Literature on manufacturing flexibility is vast and there is no general agreement on its definition as organisations define it according to their specific needs. However, researchers have attempted to divide the concept into several classes [16]. The flexibility requirements and the key performance indicators considered in the PERFoRM project fall generally under *machine, process and routing* flexibilities [16].

Machine flexibility refers to the ability of a machine to perform a variety of operations either in fixed or variable configurations. Reconfiguration is outside the scope of Task 4.2, so this class of flexibility is not addressed. It is assumed that once machines have been configured for a set of part families, they cannot be changed during the schedule execution. The requirements capture exercise undertaken with GKN personnel revealed that the *process, routing and operation flexibilities* are applicable to their production planning and scheduling problem.

Process flexibility refers to the ability of a manufacturing system to produce various parts concurrently [16]. The planning and scheduling problem presented by the GKN use case involves manufacturing of various parts, which are put together to produce final components. The parts are made at the same time, so process flexibility is a requirement in this task. The process flexibility is a fundamental requirement in most production planning and scheduling problems.

Routing flexibility is the ability to choose any machine from a set of available ones to perform an operation rather than having a single machine. This can be achieved only if there is redundancy in terms of spare capacity in the system. The redundancy can be created either by having multiple multi-purpose machines capable of performing a variety of operations, or by configuring a group of machines to perform the same single operation. The layout in Figure 2 enables parts to be manufactured via several routes and each operation could be performed by any suitable machine tool or in any Micro-Flow cell. This layout enables the manufacturing system to maintain production under failure conditions by rerouting and rescheduling parts effectively. This is a typical flexible job-shop or flow-shop scheduling problem that is modelled to allow an operation of each part to be processed by more than one machine.

Considering the afore-mentioned requirements, the scheduling problem in Task 4.2 is multi-component. The problem is modelled as a flexible job-shop for cellular and functional layout system. Unlike the classical flexible job-shop problem where machines are arranged in purely functional layout, some operations are performed in the Micro-Flow cells. This means that once a part enters a cell, the cellular operations of the part must be constrained to that cell rather than allowing the part to move in between cells (Figure 3).

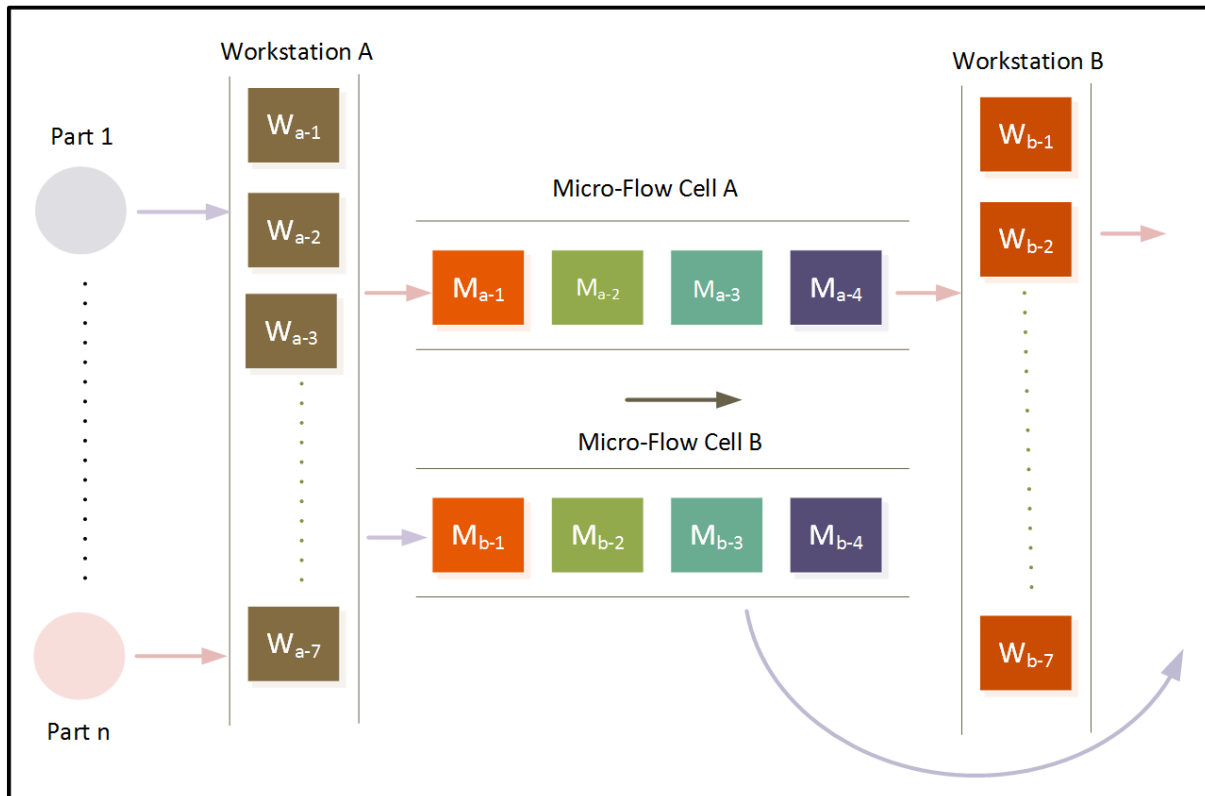


Figure 3 Parts flow diagram through workstations and Micro-Flow cells

### 2.3. Performance Evaluation Criterion

The goal of scheduling optimisation is to find the best schedule based on the objective functions of relevance. Commonly used performance measures are makespan, tardiness and flow time. However, some researchers have also considered machine utilisation and energy consumption optimisation [17, 18, 19]. The concept of makespan is not directly applicable to production facilities that have both static and dynamic parts entry patterns. The two optimality objectives considered in Task 4.2 are *energy consumption* and *tardiness*. The energy consumption is the amount of energy consumed by a machine during processing. Tardiness is a measure of delay in manufacturing a part. Since these two objectives could be conflicting, the goal is to create schedules that provide the best trade-offs between them.

Evolutionary multi-objective optimisation approaches have been applied to various scheduling problems and their ability to explore multiple trade-offs in the objective space has been shown [17, 19, 20]. The advantage of simultaneous optimisation of multi-objective functions over weighted sum of single objectives is that pareto-optimal solutions from which a solution that satisfies the subjective preference of a decision maker can be generated. However, weighted sum of energy cost and tardiness cost is used as the objective function in this task for simplicity.

## 2.4. Hybrid Centralised and Agent-Based Scheduling

As mentioned in Chapter 1, both centralised and distributed agent-based planning and scheduling systems have unique pros and cons. In agent-based scheduling, a greedy strategy is used to make a locally optimal decision at each stage of the process with the hope that a globally optimal solution could emerge. A greedy strategy does not always produce an optimal solution because of its short sightedness and non-recoverability. Agents can make certain commitments too early in the execution process, which prevent them from finding the best overall solution later. In order to create a system that is close to optimal and also robust to disturbances, the traditional centralised method is combined with agent-based approach.

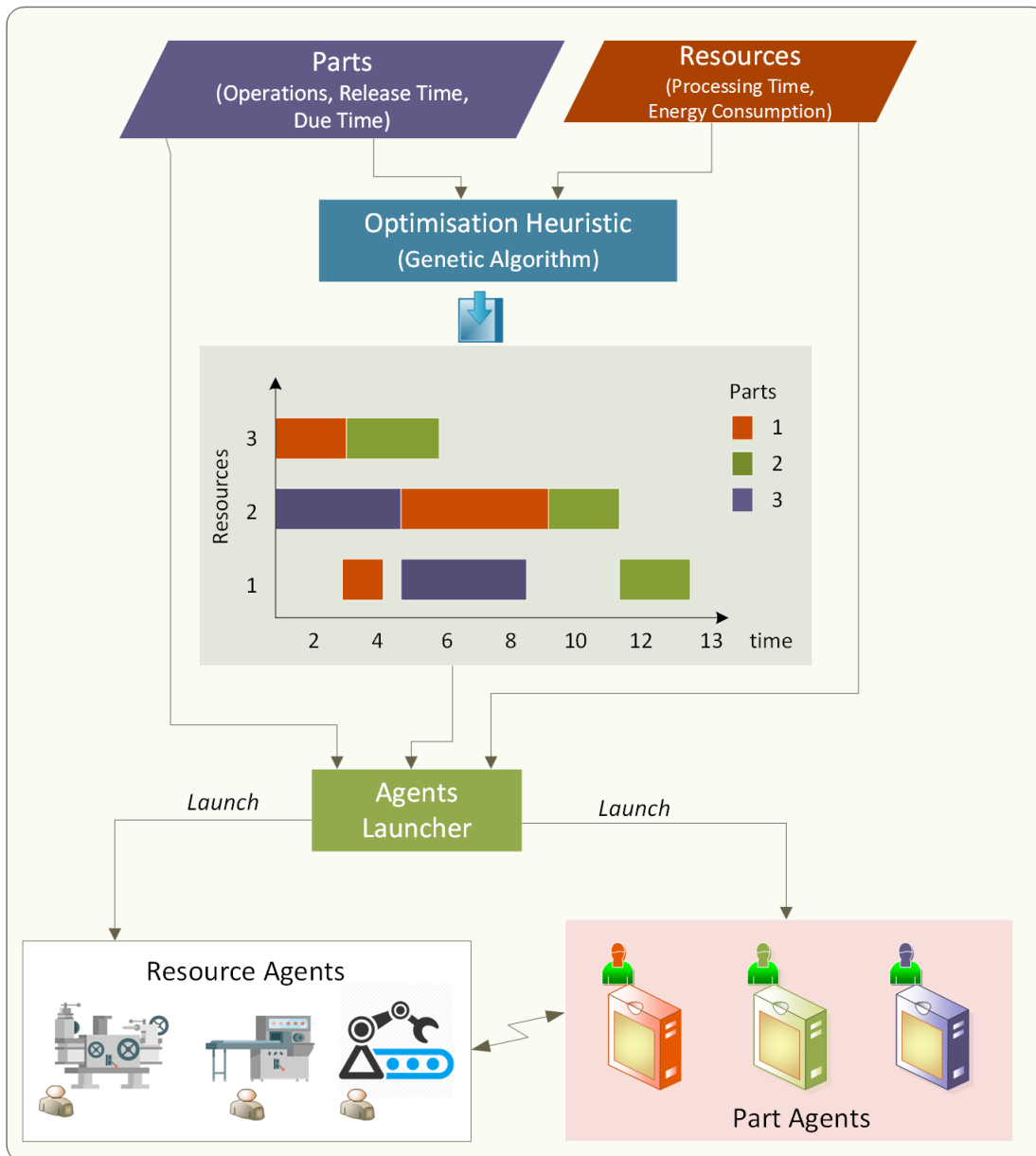


Figure 4 Hybrid centralised and agent-based scheduling system

As shown in Figure 4, genetic algorithm is used to generate an initial optimised schedule. The initial schedule is then used to generate a dispatch rule for the agents during the workflow execution phase. A dispatch rule known as the earliest operation due time (EODT) is created from the schedule. The EODT is different from the conventional earliest due time dispatch rule. It represents the start time of an operation in the optimised schedule rather than the overall due time of the part.

In terms of implementing the proposed solution on the shop floor, the manufacturing execution system (MES) layer will need to be structured differently from the existing architecture. The architecture that will support the solution is as shown in Figure 5.

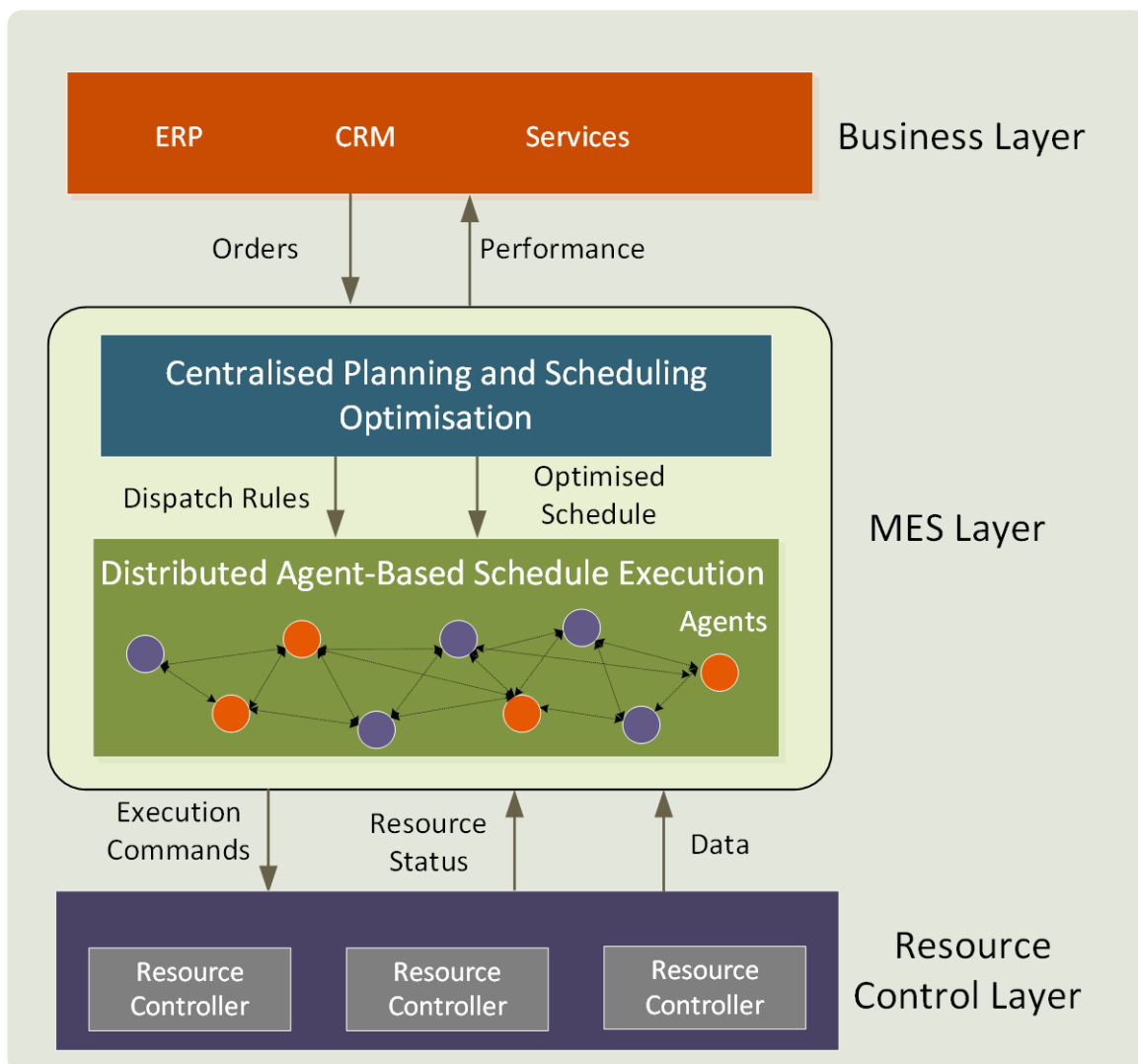


Figure 5 MES architecture for the hybrid scheduling system

## 2.5. Data Model

A baseline data model for the PERFoRM project has already been created in Work Package 2 (WP2). However, the model is not adequate for the requirements of Task 4.2. Therefore, the PERFoRM data model has been extended by introducing additional classes. The class diagram for the model is shown in Figure 6.

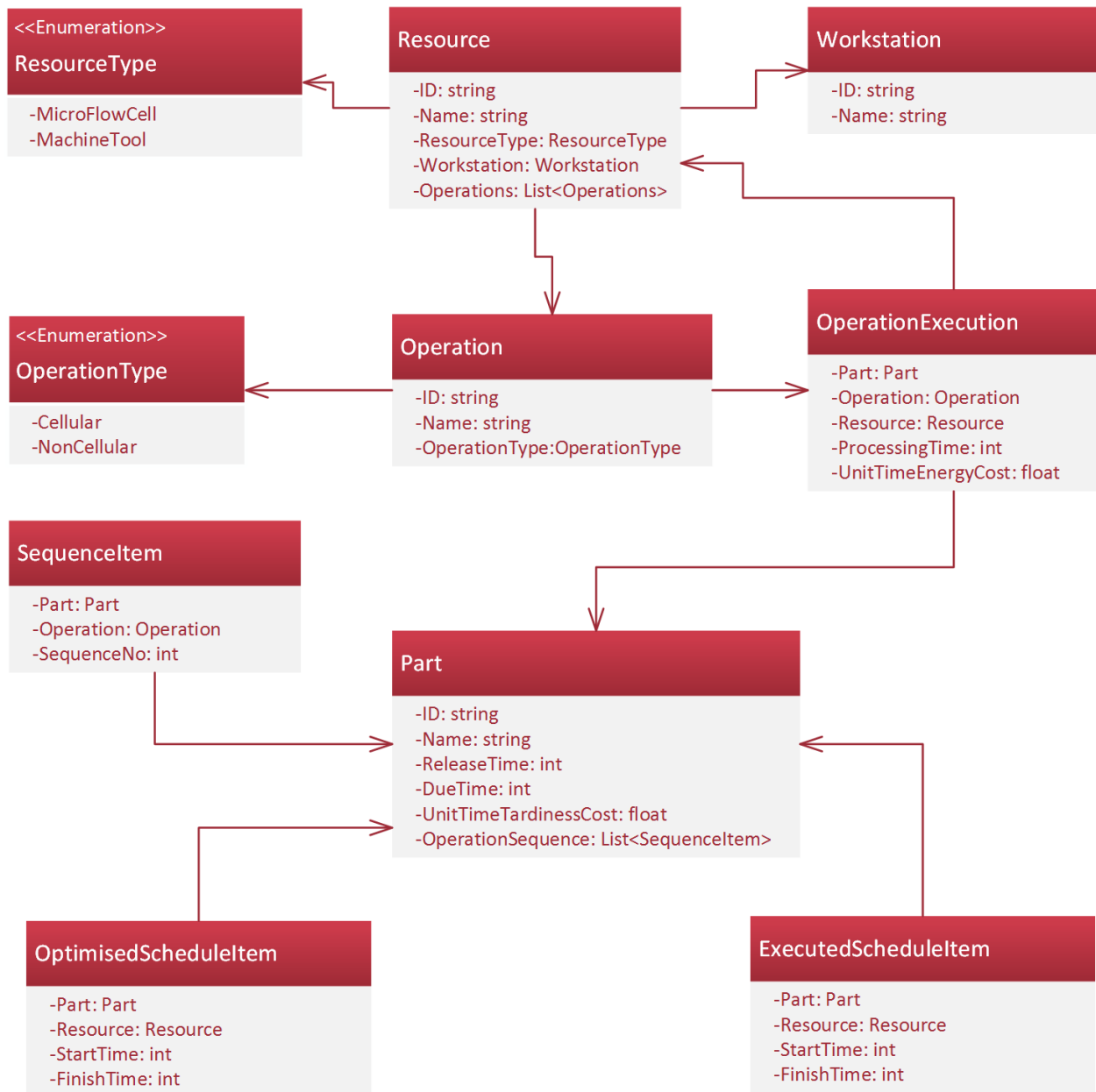


Figure 6 Data model for Task 4.2

**Operation:** The *Operation* class is a representation of a manufacturing operation

- ID: A unique identifier for an operation
- Name: Name of the operation
- OperationType: Type of the operation (Cellular, Non-Cellular)

**Workstation:** The *Workstation* class is a representation of an area in which a group of resources can be arranged

- ID: A unique identifier for a workstation
- Description: Description of the workstation

**Resource:** The *Resource* class is a representation an actual equipment

- ID: A unique identifier for a resource
- Name: Name of the resource
- ResourceType: The type of resource (Machine Tool or Micro-Flow Cell)
- Workstation: The workstation in which the resource is physically located
- Operations: The list of operations that the resource can perform

**Part:** The *Part* class is a representation of a part to be manufactured

- ID: A unique identifier for a part
- Name: Name of the part
- ReleaseTime: The time the part enters the system
- DueTime: The time the part is due
- OperationSequence: The list of operation sequence item for manufacturing the part

**SequenceItem:** The *SequenceItem* class is a representation of a part-specific operation

- Part: The associated part
- Operation: The associated operation
- SequenceNo: The execution order of the operation for the part

**OperationExecution:** The *OperationExecution* class defines the process parameters for a part and operation on a resource

- Part: The associated part
- Operation: The associated operation
- Resource: The associated resource
- ProcessingTime: The time for processing the part on the resource
- UnitTimeEnergy: The energy consumed during the processing of the part on the resource



***OptimisedScheduleItem***: The *OptimisedScheduleItem* class is a representation of a schedule item generated by global schedule optimiser

- Part: The associated part
- Resource: The associated resource
- StartTime: The time the part seizes the resource
- FinishTime: The time the part releases the resource

***ExecutedScheduleItem***: The *ExecutedScheduleItem* class is a representation of a schedule item executed by agents

- Part: The associated part
- Resource: The associated resource
- StartTime: The time the part seized the resource
- FinishTime: The time the part released the resource

## 2.6. Agents Modelling, Simulation and Development Tools

There are several definitions of Agent-Based Modelling (ABM) in the literature, but from a practical point of view, ABM can be defined essentially as decentralised, and individual-centric approach to model design. When designing an agent-based model, the active agents of the system are identified, and their behaviors are defined. The global behavior then emerges from the interactions among the individual behaviors. An agent is simply regarded as an entity or software abstraction that is similar to object-oriented programming concepts such as objects, attributes and methods. However, an agent presents a distinctive level of abstraction. Instead of being expressed in terms of attributes and logic-based methods, an agent is expressed in terms of behaviour and interaction with its environment. In a practical sense, an agent is anything that can perceive its environment through sensors and then act in response.

In recent years, the ABM community has developed several tools for developing agent-based applications. The tools differ greatly in features, but a comprehensive comparative study of the entire spectrum of tools is beyond the scope of this report.

The toolkits that have been chosen for modelling, simulation and development work in Task 4.2 are covered in this report.

### 2.6.1. Java Agent Development Framework

JADE is an opensource software framework for developing peer-to-peer agent-based applications in compliance with the Foundation for Intelligent Physical Agents (FIPA) specifications for interoperable intelligent multi-agent systems. FIPA is an organization that promotes agent-based technology and the interoperability of its standards with other technologies. JADE supports the majority of the FIPA specifications, making it an ideal choice for agent simulation and development.

JADE is completely developed in Java language and it simplifies the implementation of multi-agent systems through a middleware and a set of graphical tools for debugging and deployment. A JADE-based system can be distributed across heterogenous machines and the agents can be moved from one machine to the other as and when required. JADE also provides a powerful task execution and composition model, peer-to-peer agent communication based on the asynchronous message passing paradigm, a service supporting publish subscribe discovery mechanism and many other advanced features that facilitate the development of a distributed system [21]. Figure 7 shows the FIPA agent management reference model and Figure 8 represents the JADE FIPA-compliant agent architecture. Agents can communicate transparently regardless of whether they live in the same container (e.g. A2 and A3), in different containers (in the same or in different hosts) belonging to the same platform (e.g. A1 and A2) or in different platforms (e.g. A1 and A5).

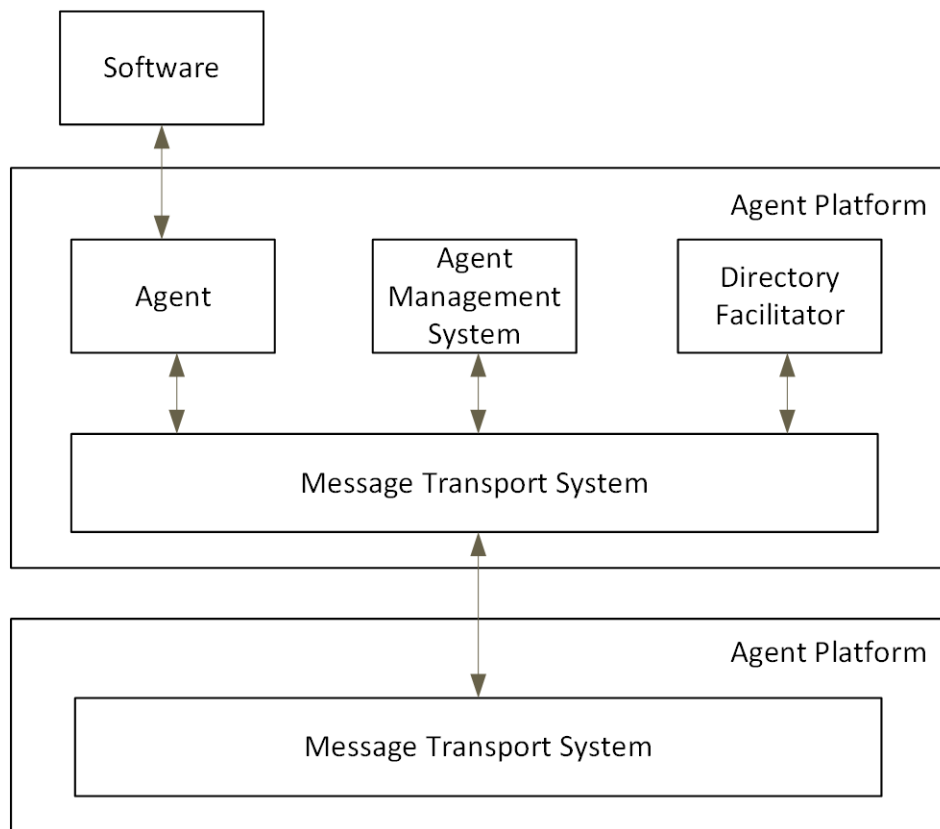


Figure 7 FIPA Agent management reference model

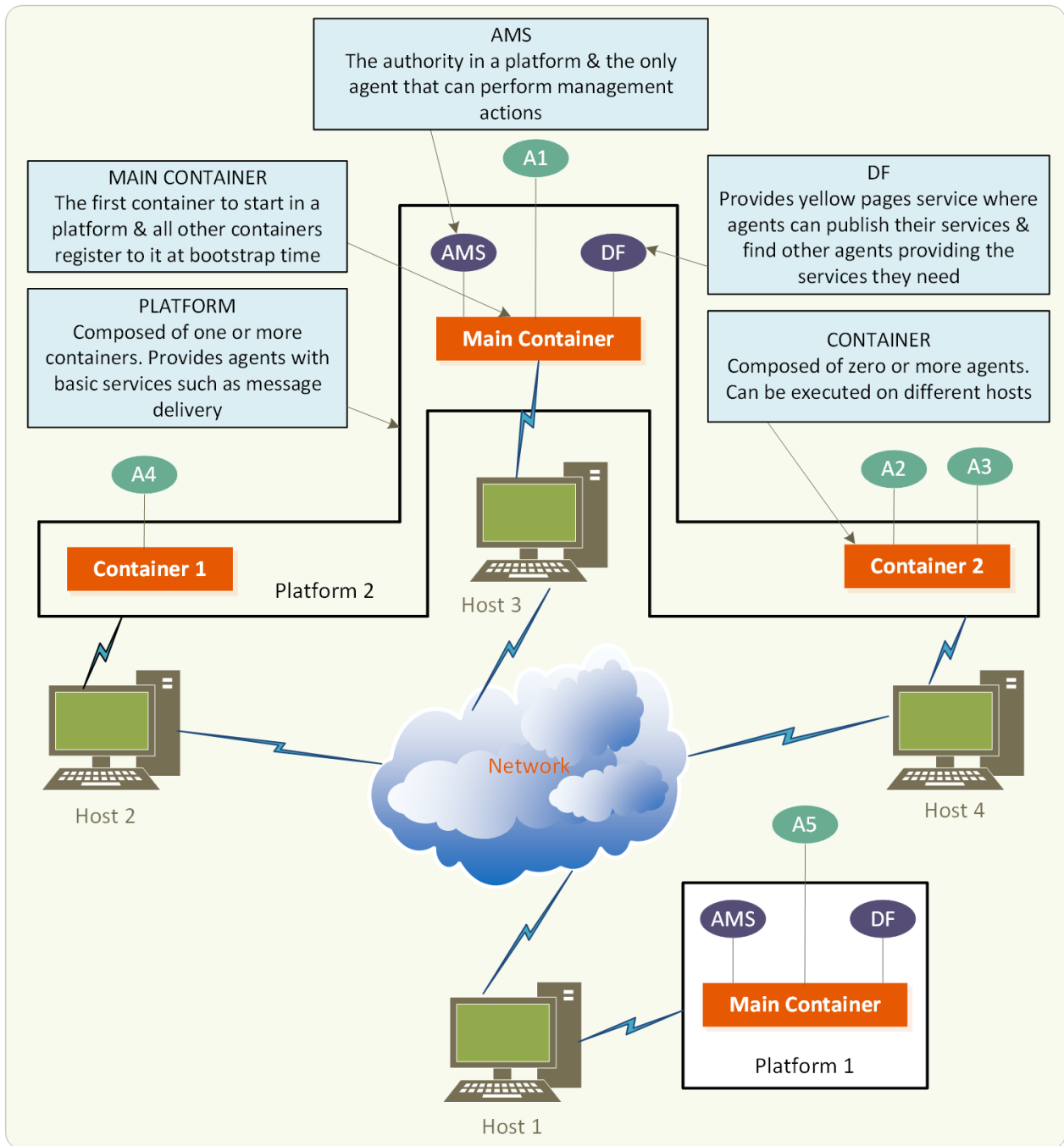


Figure 8 JADE architectural elements

### 2.6.2. Workflow and Agents Development Environment (WADE)

WADE is an opensource software platform for workflows and agents-based application. It is an extension of JADE, inheriting the basic elements of JADE such as agents, behaviours and messaging. In addition to the inherited JADE features, WADE provides support for the execution of tasks using the embedded lightweight workflow engine. The tasks to be performed by agents, their execution steps and sequences, their activation and termination criteria and other information that are required to complete the tasks are explicitly defined using the workflow metaphor. This makes it possible to create automatic mechanisms that trace the execution of a workflow thus facilitating system monitoring and problem investigation. Each agent is equipped with a set of workflows that it executes depending on the dynamic situation. One of the main advantages of the workflow approach is the possibility of representing processes in purely graphical forms. WADE comes with a development environment called WOLF that facilitates the creation of workflows using a graphical editor. The WADE suite (Figure 9) also includes a web administration console, web services and a module for persisting the state of running workflows in a database.

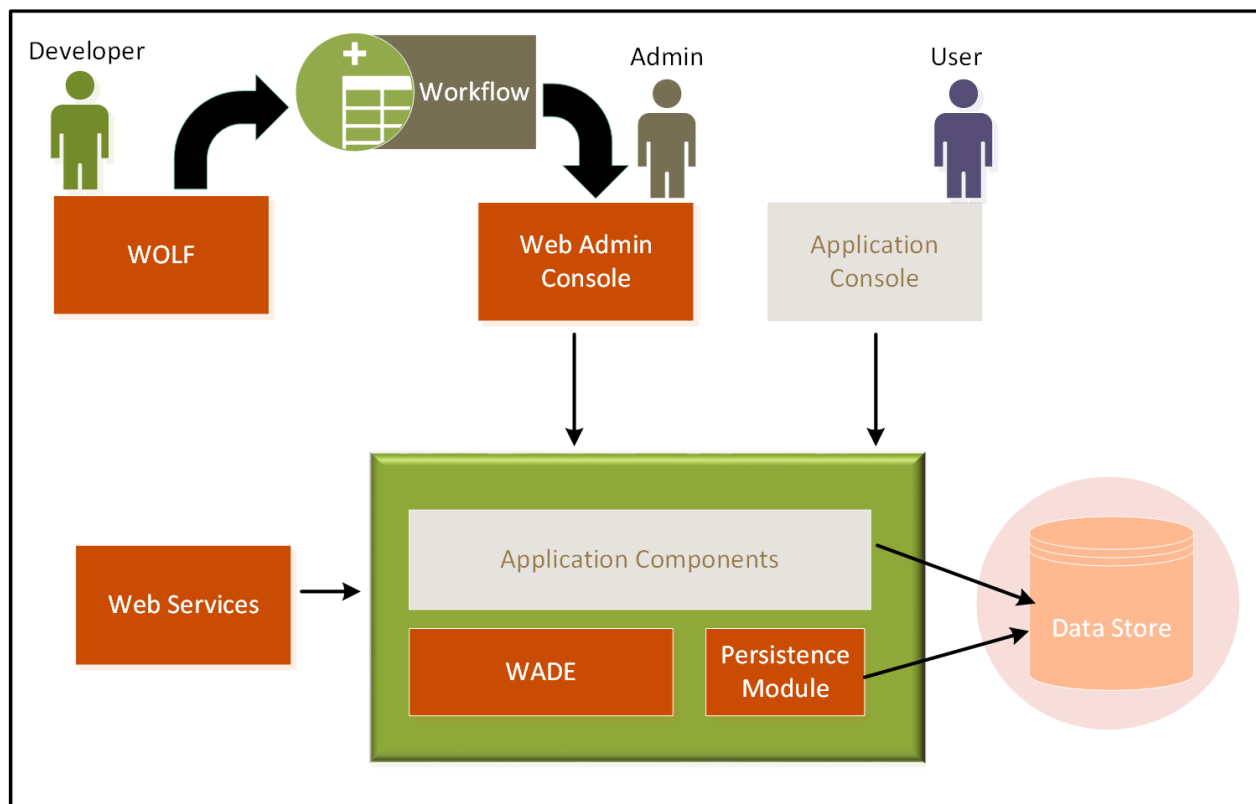


Figure 9 Components of a WADE-based application

### 2.6.3. AnyLogic Simulation Software

AnyLogic is a professional software for building multimethod simulation models (discrete event, system dynamics and agent-based) across a wide range of industries. Agent-based models can be easily combined with discrete event and system dynamics techniques in a single model to simulate business processes of any complexity. A model can be configured from an external data source when the model is run, which means that a whole model structure can be modified by simply changing the input data. This makes models reusable and many similar challenges can be solved without additional model-building effort.

Although AnyLogic as an agent-based simulation framework is not compliant with FIPA specifications for interoperable and multi-agent system, it includes a graphical modeling language and simulation models are extendable using Java language. AnyLogic provides the basic building blocks for designing and implementing agents using constructs that facilitate the manipulation of an agent's beliefs, goals, communication and decision-making structure.

### 3. Flexible Job-Shop Optimisation

#### 3.1. Problem Formulation

The flexible job-shop scheduling problem is formulated as follows. Let  $P = \{P_1, P_2, \dots, P_n\}$  be a set of  $n$  parts to be processed on  $m$  machines  $M = \{M_1, M_2, \dots, M_m\}$ , which are arranged in functional and cellular layouts. Each part  $P_i$  consists of a set of operations  $O_i = \{O_{i_1}, O_{i_2}, \dots, O_{i_s}\}$  that must be processed on a set of machines according to a predefined order. Each operation  $O_{ij}$  must be performed by one out of a set of available machines; the processing time is denoted by  $t_{ijk}$  and the unit time energy cost is denoted by  $e_{ijk}$  for machine  $M_k$ . The objective is to determine an assignment and a sequence of the operations on the machines to minimise weighted sum of energy and tardiness costs. The definition of parameters are as follows:

$n$  total number of parts

$s$  total number of operations for each part

$m$  total number of machines

$P_i$  the  $i^{\text{th}}$  part

$M_k$  the  $k^{\text{th}}$  machine

$O_{ij}$  the  $j^{\text{th}}$  operation of the  $i^{\text{th}}$  part

$e_{ijk}$  the unit time energy cost of processing the  $j^{\text{th}}$  operation of the  $i^{\text{th}}$  part on the  $k^{\text{th}}$  machine

$t_{ijk}$  the processing time of the  $j^{\text{th}}$  operation of the  $i^{\text{th}}$  part on the  $k^{\text{th}}$  machine

The assumptions and constraints on the parts and machines are listed as follows:

- All machines are available at any time
- All parts are released at time  $t = 0$
- Each machine (and Micro-Flow cell) can process only one part at a time
- There are no precedence constraints among operations of different parts
- The sequence of operation is predefined and cannot be modified i.e. operation  $O_{ij}$  must be completed before operation  $O_{ij+1}$
- No operation preemption. i.e. an operation cannot be interrupted on a machine once started until it is completed

- All cellular operations of a part must be processed in the same Micro-Flow cell
- A part can visit a machine more than once

### 3.2. Genetic Algorithm

Genetic algorithm is a heuristic search method used in artificial intelligence and computing to find optimised solutions to search problems based on the theory of natural selection and evolutionary biology. The method begins with a population of individuals, which represents a set of potential solutions in the search space. An individual in the population is assigned a fitness value according to a problem-specific objective function. The individuals attempt to combine the good features in each parent in the population using reproduction operators to construct offspring which are fitter than the previous generations. Depending on the needs of the application, the procedure continues until an acceptable solution is derived or until a certain number of generations have passed.

The key elements of genetic algorithm are the type of chromosome representation that is used, the crossover operator, mutation operator and the selection method. The details of the model developed in this task are explained using the data shown in Table 1. A value of 1 in the machine column indicates that the corresponding operation can be processed by the machine while the value of 0 indicates the opposite.

**Table 1 Operations and machines assignment data**

Part	Operation	Machines					
		$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$
<b>1</b>	O <sub>11</sub>	1	1	0	0	0	0
	O <sub>12</sub>	0	0	1	1	0	0
	O <sub>13</sub>	0	0	0	0	1	1
<b>2</b>	O <sub>21</sub>	1	1	0	0	0	0
	O <sub>22</sub>	0	0	1	1	0	0
	O <sub>23</sub>	0	0	1	0	1	1
	O <sub>24</sub>	0	0	0	0	1	1
<b>3</b>	O <sub>31</sub>	0	0	0	0	0	1
	O <sub>32</sub>	0	0	0	1	1	0
	O <sub>33</sub>	1	0	0	0	1	1
<b>4</b>	O <sub>41</sub>	1	1	0	0	0	0
	O <sub>42</sub>	0	0	1	1	0	0
	O <sub>43</sub>	1	1	0	0	0	0

O <sub>44</sub>	1	0	1	0	0	0
O <sub>45</sub>	0	0	0	0	1	1

The model adopted in this task is based on the algorithm proposed by Du & Xiong [22] for a flexible job-shop scheduling problem. The algorithm has been modified to account for the combination layout scenario described in Section 2.

According to Table 1, a matrix called the *operation-machine matrix*,  $M_{om}$ , is derived. It represents the constraint model of relation between operations and machines. The machines and operations are arranged along the rows and columns of the matrix respectively. If the element  $M_{om}(i, j) = 1$ , then the operation in the  $j^{th}$  column can be processed by the machine in the  $i^{th}$  row.

$$M_{om} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

A second matrix called the *part-operation matrix*,  $M_{po}$ , is also derived. The matrix represents the constraints between parts and operations. The parts and operations are placed along the columns and rows respectively. If the element  $M_{wp}(i, j) = 1$ , then the operation in the  $i^{th}$  row belongs to part in the  $j^{th}$  column.

$$M_{po} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



### 3.2.1. Chromosome Encoding

A chromosome is a symbolic representation of a feasible schedule. Most previously adopted representations are one-dimensional but a 2-dimensional operation-based encoding is used for the flexible scheduling problem. The chromosome representation has two components; *operation sequence* component (OS) and the *machine selection* component (MS). The OS component values represent the sequence of operation while the MS component values represent the machine that is selected to process the corresponding operation.

A valid chromosome for the problem in Table 1 is shown in Table 2. According to Table 1, there are 15 total operations, so the length of the chromosome is 15. The genes of the OS component are filled with random numbers generated from 1 to 15. The operations are scheduled in the order of their values. The MS component values are obtained by choosing a machine from the set of available machines for an operation during population initialisation. For instance,  $M_5$  has been selected to process operation  $O_{24}$  since the corresponding value is 5. The value could also be 6 since  $M_6$  is among the available machine set for the operation.

Table 2 Chromosome

	$O_{11}$	$O_{12}$	$O_{13}$	$O_{21}$	$O_{22}$	$O_{23}$	$O_{24}$	$O_{31}$	$O_{32}$	$O_{33}$	$O_{41}$	$O_{42}$	$O_{43}$	$O_{44}$	$O_{45}$
OS	15	3	5	4	11	12	7	10	9	8	2	14	1	6	13
MS	2	5	4	1	2	3	5	6	4	6	3	1	2	6	1

### 3.2.2. Chromosome Decoding

A decoding scheme is required to ensure that the chromosome produces a feasible schedule. The chromosome decoding is described in the following steps: -

*Step 1* Select the operation with the minimum value of OS and determine the part that the operation belongs to. For instance, the operation with the minimum OS value in Table 2 is  $O_{43}$ , which belongs to part 4.

*Step 2:* Search through the corresponding part column of matrix  $M_{po}$  and select the first operation with value equal to 1. This preserves the relative precedence constraints in the operations of the same part. For instance, the operations in column 4 of matrix  $M_{po}$  with values equal to 1 are  $O_{41}$ ,  $O_{42}$ ,  $O_{43}$ ,  $O_{44}$ ,  $O_{45}$ . The first operation is  $O_{41}$ , so it is selected

*Step 3:* According to the selected part and operation, set the corresponding element of matrix  $M_{po}$  to 0. This effectively removes the operation from the matrix so that it will not be considered in the subsequent iterations.

*Step 4:* Repeat step 1 to 3 for the other operations.

The sequence of operation after decoding the chromosome in Table 2 is given as follows:

$$O_{41} - O_{42} - O_{11} - O_{21} - O_{12} - O_{43} - O_{22} - O_{31} - O_{32} - O_{33} - O_{23} - O_{24} - O_{44} - O_{45} - O_{13}$$

The first operation is scheduled first, followed by the second operation, and so on. A schedule generated by the procedure is guaranteed to be feasible, so no repair mechanism is required for the chromosomes.

### 3.2.3. Initial Population Generation

Population initialisation is a crucial task in genetic algorithm because it affects the feasibility and quality of the final solution. The initial population generation for the OS and MS components of individuals are done in stages. The method for assigning machines to operations considers the cost to be minimised and the location of the machine. Since the machines are arranged in workstations and Micro-Flow cells, all cellular operations of a part are constrained to the same cell.

*Step 1:* Create an operation-machine matrix,  $M_{op}$ , and initialise based on the available machine set for each operation

*Step 2:* Create an array to hold the available time of all machines and initialise to 0

*Step 3:* Randomly fill the operation sequence component of the chromosome with values from 1 to 15 without any repetition

*Step 4:* Decode the operation sequence component as described earlier. For each operation, calculate the weighted sum of energy and tardiness costs for each machine in the available set as follows:

$$\alpha(\text{Available Time} \times \text{Unit Tardiness Cost}) + \beta(\text{Processing Time} \times \text{Unit Time Energy Cost}) \quad (1)$$

*Step 5:* Select the index  $k$  of the machine which has the lowest value. If there is a tie, a machine is randomly selected

*Step 6:* Add the processing time of the operation on the machine to the current available time of the machine

*Step 7:* If the operation is in a cell, then reduce the available machines for performing the remaining cellular operations of the part to the same cell by setting their values to 0 in the  $M_{po}$  matrix.

*Step 8:* Repeat step 3 to 7 until all operations have been selected.

### 3.2.4. Fitness Evaluation

Fitness evaluation involves the definition of an objective function to be used for determining the suitability of a chromosome for reproduction. The fitness function is defined as  $fit(c) = 1/fc$ , where  $fc$  is the weighted sum of total tardiness and energy costs for the schedule produced by a chromosome.

$$f_c = \alpha T_c + \beta E_c \quad (2)$$

$$T_c = \sum_{i=1}^n \max(0, C_i - D_i) \times U_i \quad (3)$$

$$E_c = \sum_{i=1}^n \sum_j^s t_{ijk} \times e_{ijk} \quad \text{for all } k \text{ in selected machines} \quad (4)$$

where

$\alpha$  is the weight of total tardiness cost  $T_c$

$\beta$  is the weight of total energy cost  $E_c$

$C_i$  is the completion time of the  $i^{th}$  part

$D_i$  is the due time of the  $i^{th}$  part

$U_i$  is the tardiness cost per unit time that the  $i^{th}$  part exceeds its due date

### 3.2.5. Genetic Operators

*Selection Strategy:* At each iteration, the best chromosomes are chosen for reproduction using the tournament selection method.

#### *Crossover Operator:*

The choice of crossover operator is very important in genetic algorithm, and consequently a wide range of crossover operators have been proposed for flexible job-shop problem. Crossover operators are application and chromosome dependent. The crossover operator is applied to the operation sequence chromosome only while the assignment of machines to operations is preserved in the offspring. An ordered crossover operator is used so that relative order information can be transmitted to the offspring. The crossover procedure is described as follows:

Parent 1

Operation	11	1	4	2	12	10	3	5	13	8	14	6	9	15	7
Machine	2	2	1	3	3	6	5	4	1	6	1	6	4	6	5

Parent 2

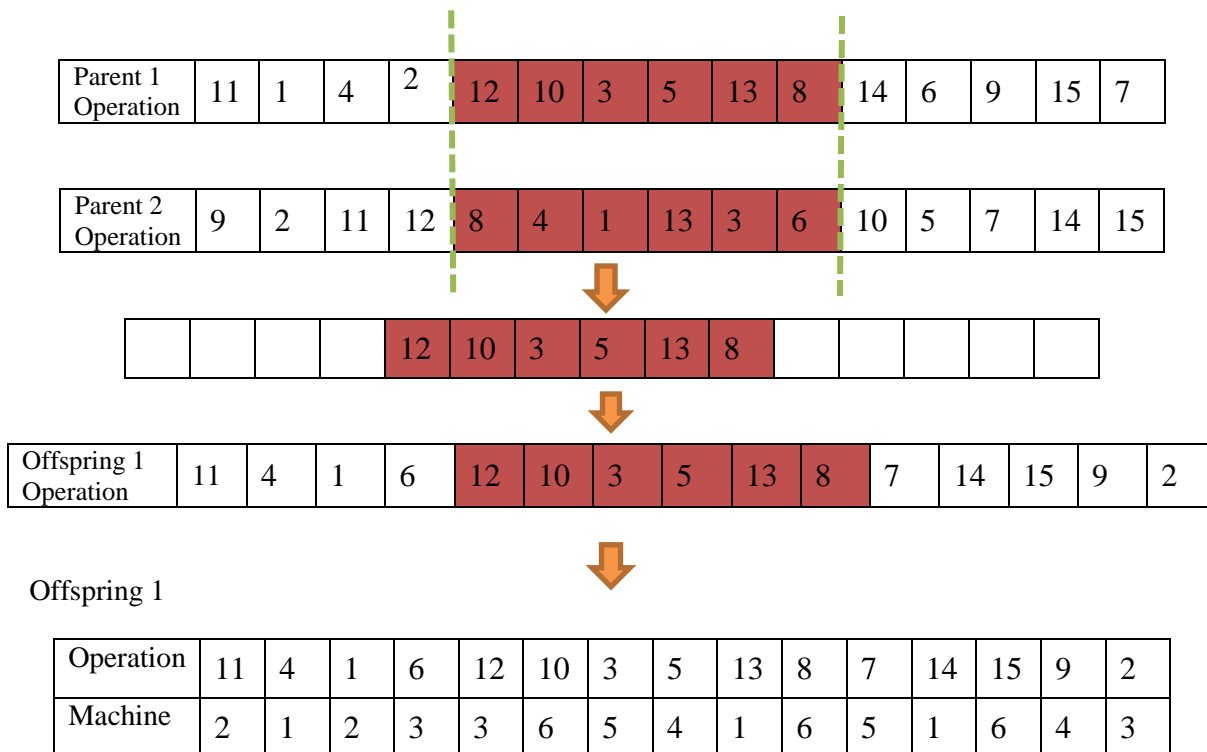
Operation	9	2	11	12	8	4	1	13	3	6	10	5	7	14	15
Machine	5	4	1	4	6	2	1	2	6	3	5	3	6	3	5

*Step 1:* Create two random crossover points and copy the consecutive alleles between the points from the operation chromosome of the first parent into the first offspring

*Step 2:* Starting from the second crossover point in the second parent, copy the remaining unused alleles from the second parent to the first offspring, wrapping around the list

*Step 3:* Copy the corresponding alleles from the machine selection chromosome of the first parent into the first offspring

*Step 4:* Repeat step 1 to 3 with parent roles reversed to get the second offspring



### ***Mutation Operator:***

Mutation operators are generally applied to introduce and maintain diversity from one generation of a population of chromosome to the next, thereby preventing the evolutionary process from being trapped in a local optimum. In this study, mutation is applied to the operation sequence and machine selection chromosomes as described in the following steps: -

*Step 1:* Choose two low probability threshold values for mutating the operation sequence and machine selection chromosomes respectively.

*Step 2:* Loop through the chromosome from left to right and generate a random probability value for each position.

*Step 2a:* If the randomly generated probability value is less than the machine selection chromosome mutation probability, set the machine allele in that position to an adjacent machine from the alternating machine set for the corresponding operation.

*Step 2b:* If the randomly generated probability value is less than the operation sequence chromosome mutation probability, then swap the operation and machine in that position with the ones in another randomly selected position. This should only be done for non-cellular operations.

#### 4. Agent-Based Scheduling Mechanism Design

Complex production systems are subject to unexpected disruptions that conventional monolithic planning and scheduling systems are not designed to deal with. Multi-agent systems have been proven to excel in such dynamic environments [7, 8, 9]. The shortcomings of static scheduling algorithms can be compensated for by using multi-agent systems to improve the overall performance of production systems. The real-time decision-making capabilities of agents provide a high degree of flexibility and adaptability to unpredictable changes that may occur during manufacturing process.

This section is focused on the design of a multi-agent control architecture that enables interaction and negotiation among various agents in a manufacturing shop floor to produce a feasible schedule in the face of stochastic events such as:

- Introduction of new parts into the manufacturing system during the execution of existing known parts
- Changes in parts due dates
- Dynamic changes in parts sequence of operation
- Removal of resources due to failures
- Addition of resources to minimise bottlenecks
- Correction of defective or non-conforming parts after inspection

The agent architecture in this task was designed following the “DACS- Designing Agent-based Control Systems” methodology for production control, developed at DaimlerChrysler’s research labs in Berlin [23]. The goal of the methodology is to produce a design that specifies the agents of the control system, and how each agent negotiates with the other agents to achieve desired objectives. The steps involved in the methodology are as follows:

- *Analysis of control decisions:* The control decisions that are necessary to operate the production process are identified and analysed through exercises conducted with GKN personnel. The aim of the exercises was to understand requirements and to establish common needs and expectations. The control decisions and dependencies are analysed and incorporated into a decision model.
- *Identification of agents:* The agents that form the multi-agent system, the decisions they are responsible for, and how they interact with other agents are all identified in this step. A system consisting of four agent types was designed in accordance with the overall objectives of the planning and scheduling system.
- *Selection of interaction protocol:* To facilitate the interaction and cooperation among the agents, four interaction protocols are used.

## 4.1. Agents and Characteristics

In agent-based modelling, software agents are used to represent the individual active components of a system. Although there is no agreed definition of software agents in the literature, most researchers agree on some characteristics of agents such as the ability to make decisions without external influence (autonomy), ability to communicate with other agents (sociable), ability to perceive its environment and respond to changes (reactive) and the ability to take initiatives in achieving its goals (proactive). The following requirements are considered in designing the agent-based architecture (Figure 10) :-

- **Autonomy and Social Ability:** Each agent has its own goal and must be able to make decisions based on local knowledge and in collaboration with other agents. There is no shared knowledge of state in distributed systems, so the overall behaviour of the system emerges from the interactions among the individual agents.
- **Flexibility and Adaptability:** The agents must be able to adapt automatically to unexpected changes in production environment. To achieve this, the agents need to be aware of their environment and then act proactively in response to whatever information they receive. For example, if a resource breaks down, the resource agent should be aware, and information should be sent to the concerned agents. If the breakdown occurs during the processing of a part, the part agent should be informed so that it can find an alternative resource that will achieve its goals.
- **Plug and Produce:** Plug and produce is a concept that allows resources to be quickly added and removed from a manufacturing process. To enable this functionality, the multi-agent system is designed to allow creation and destruction of agents living in a population at runtime. Agents are created dynamically as resources or parts are introduced into the manufacturing process and are destroyed when removed.

The framework developed in Task 4.2 consists of the following types of agents:

- **Resource Agent (RA):** Resource agents are used to represent the manufacturing devices on the shop floor. Although the machine tool and Micro-Flow cells could have been represented using separate agents, for the scenario in consideration, it suffices to represent them using a common agent. Resource agents hold information about the status and operational data (energy consumption and processing time) of resources. The agents are created when devices are introduced into the shop floor and are destroyed when removed.
- **Part Agent (PA):** Part agents represent the parts to be manufactured. Part agents hold information about the operations, workflow and other information that are required to manufacture the part they represent. The agent also keeps track of the parts status at any given time. The lifecycle of a PA begins when the part is released into the production process. During its existence, the agent negotiates with other agents to find suitable resources for executing operations during its manufacturing process. The agent will be destroyed when the part is completed or cancelled.

- **Schedule Optimisation Agent (SOA):** The Schedule Optimisation Agent is a broker that facilitates communication, collaboration and coordination among resource agents and part agents using the requirements and capabilities of the agents. It has a much global view of the overall system and its function is to optimise the scheduling plan to achieve the best possible overall performance.
- **Shop Management Agent (SMA):** One way to implement modular support in agent-oriented programming is to define middleware that are themselves implemented as software agents. The Shop Management Agent is a representation of the Directory Facilitator (DF) agent and the Agent Management Service (AMS) agent that are specified by FIPA. The SMA maintains a directory of all running agents and receives status information from. Agents can query the SMA to get information about the services offered by the other agents, including the agents that can provide the services required by the agents.

Table 3 Agents tasks and required information

Agent Type	Tasks	Required Information
<b>Resource Agent</b>	<ul style="list-style-type: none"> <li>• Inform SMA and PA about availability</li> <li>• Manage reservation queue</li> </ul>	<ul style="list-style-type: none"> <li>• Operational data (processing time, energy consumption and costs etc.)</li> <li>• Reasoning mechanism for preparing bids</li> <li>• Reservation queue</li> <li>• Status information (idle, busy, down)</li> </ul>
<b>Part Agent</b>	<ul style="list-style-type: none"> <li>• Manage parts processing and workflow</li> <li>• Negotiate with SMA and RA to find the best resources</li> <li>• Inform SMA about status</li> </ul>	<ul style="list-style-type: none"> <li>• Process operations and sequence (e.g. milling, cleaning, deburring, inspection etc.)</li> <li>• Due dates</li> <li>• Tardiness costs</li> </ul>
<b>Schedule Optimisation Agent</b>	<ul style="list-style-type: none"> <li>• Mediate between PA and RA</li> <li>• Match parts processing requirements appropriately with resources</li> </ul>	<ul style="list-style-type: none"> <li>• Initial statically optimised schedule</li> <li>• Operational data (processing time, energy cost)</li> <li>• Tardiness costs</li> <li>• Resources availabilities</li> </ul>
<b>Shop Management Agent</b>	<ul style="list-style-type: none"> <li>• Instantiate and destroy agents in population</li> <li>• Keeps track of all resources and parts status</li> </ul>	<ul style="list-style-type: none"> <li>• Objective function parameters</li> <li>• Resources and parts status via messages from RA and PA</li> </ul>



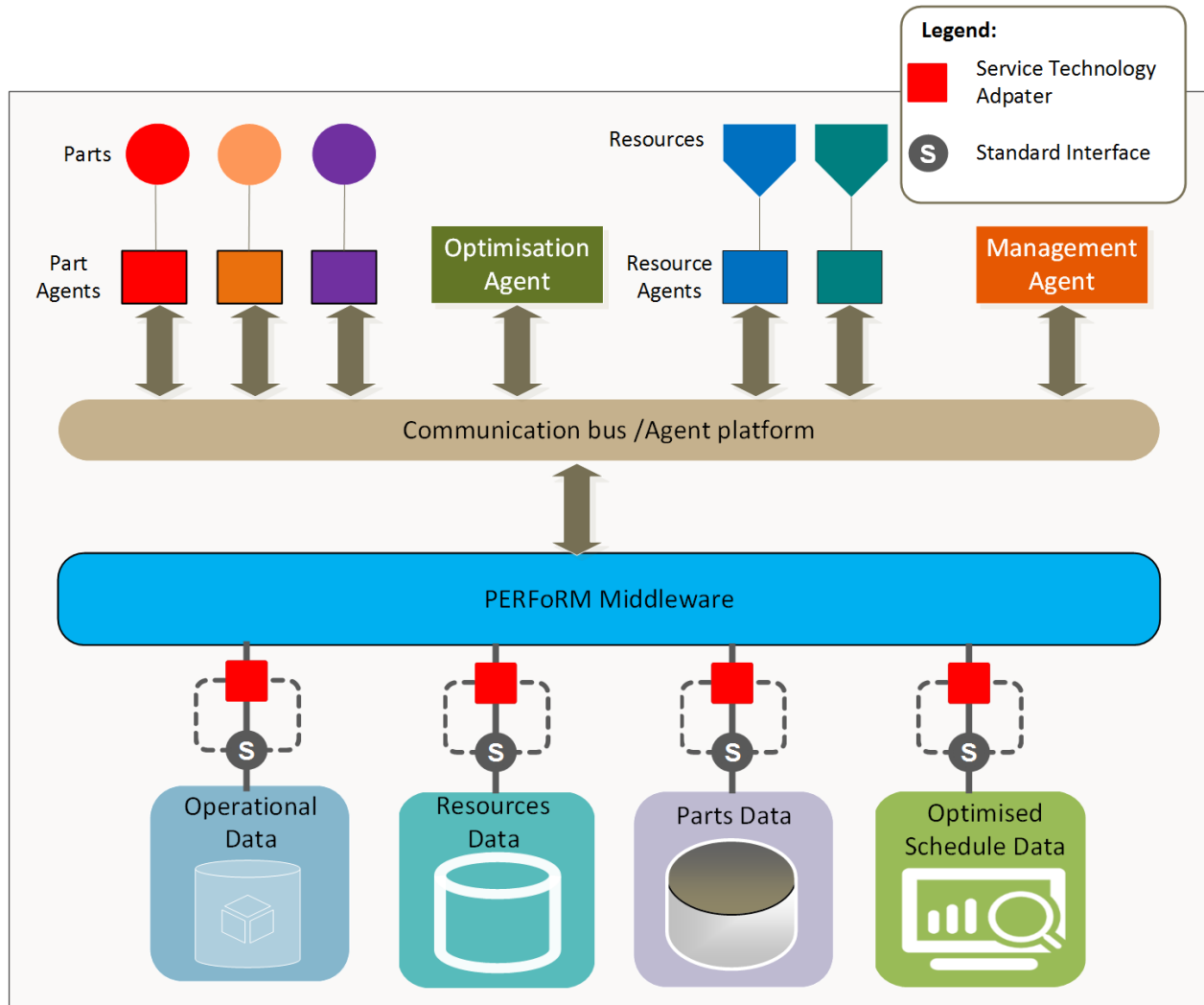


Figure 10 Multi-agent architecture for scheduling

## 4.2. Communication Acts

Coordination and cooperation are required among autonomous agents to efficiently complete a given task since no single agent has sufficient competence, resources and information to complete the task. Communications in a multi-agent system are achieved by exchanging messages. For agents to have any meaningful interaction, it is imperative that they use the same language conventions and vocabulary. Some communicative acts (performatives) have been defined by FIPA to achieve this. The communication between agents was implemented in accordance with the semantics defined in the FIPA Communicative Act Library. The following performatives are used in this task:

- *Request*: This performative is used by a sender to ask the receiver to perform an action. For instance, this is used by a PA to ask an RA to execute an operation as defined in the optimised schedule generated from genetic algorithm. The content of the message is the description of the

operation to be performed including the maximum cost (waiting time) that is acceptable to the PA. A *request* message is also sent by PA to SOA to ask for assistance in finding a resource.

- *Refuse*: A *refuse* message is sent by an RA if it is unable to execute a given operation requested by a PA. The reason for the refusal is contained in the message. For instance, the RA will respond with a *refuse* performative to the PA when the associated resource is out of service or when it is unable to meet the cost demanded by the PA.
- *Agree*: An *agree* message is sent by an RA to inform a PA that it can perform the operation requested by the PA.
- *Inform*: An *inform* message is used by a sender to inform the receiver that a given proposition is true. For instance, an *inform* message is sent by an RA or PA to communicate its current state to the SMA. It is also sent by an RA to inform a PA if it successfully completes an operation.
- *Call for Proposal*: This performative is used by a SOA to call for proposals from RAs who has the required skills and is available to execute given operations.
- *Propose*: A *propose* message is sent by an RA to SOA to submit a proposal for performing a certain operation, given certain preconditions (waiting time, processing time, energy consumption). This is in response to an earlier *call for proposal* message from SOA to the RA.
- *Accept Proposal*: An *accept proposal* message is sent by SOA to an RA to inform the RA of the acceptance of a previously submitted proposal to perform an operation.
- *Reject Proposal*: A *reject proposal* message is sent by the SOA to an RA to reject a proposal submitted by the RA for performing an operation. The reason for the rejection is included in the message.
- *Query Ref*: A *query-ref* message is used to ask another agent to inform the requester of the object identified by a descriptor. For instance, it is used by the SOA to query the SMA for available resources that can execute the operations required by a PA.
- *Proxy*: A *proxy* message is used when the sender wants the receiver to select target agents denoted by a given description and to send a message to them. For instance, a PA sends a *proxy* message to the SOA when it is unable to find a resource that meets its requirement on its own. The SOA acts on behalf of the PA to find the best possible resource to execute the operations requested by the PA.
- *Confirm*: A *confirm* message is sent by the SOA to inform a PA that it has been able to find a resource on its behalf. The details of the resource are sent to the PA.

- *Failure*: A *failure* message is used to inform the receiver that the sender could no longer fulfill a contract that was earlier agreed. For instance, an RA sends a *failure* message to a PA if a resource breakdown occurs during the processing a part or when the part is waiting on its queue.

### 4.3. Interaction Protocols

A clear policy that the agents will follow in their interactions with one another is required for them to achieve their shared goals. The protocols used in Task 4.2 are based on the interaction protocols defined by FIPA.

#### 4.3.1. Request

In this protocol, a part agent sends a *request* message to the specific resource agent it has been assigned to in the initial optimised schedule. The resource agent responds with either *agree* or *refuse* message depending on its operational state and whether it can perform the operation within the time constraints specified by the part agent. If the resource agent responds with an *agree* performative, then the part agent starts waiting on the queue until it receives further messages from the resource agent (Figure 11).

#### 4.3.2. Brokering

In the event that the resource agent responds with a *refuse* performative, then the responsibility of finding an alternative resource is passed on to the SOA. The part agent requests the SOA to act on its behalf by sending a *proxy* message, providing information on the time the required operation was due. If the operation is to be performed in a Micro-Flow cell, then the part agent also includes the list of the other cellular operations within its process flow. The SOA determines a set of agents to forward the request to by querying the SMA to get a list of all the operational resources that possess the skills required by the part agent. The SOA initiates a bidding process and then relays the outcome to the part agent. If the SOA finds a resource, a *confirm* message is sent to the part agent. Otherwise, a *failure* message is sent (Figure 12).

#### 4.3.3. Contract Net

The SOA solicits proposals from the capable and available resources by issuing a call for proposals with a description of the operations, but without placing any conditions on the execution of the operations. Since all the recipient resource agents can perform the required operations, they all respond with processing time, energy consumption and existing reservations information. The SOA decides the best resource and then communicate the result to the bidding agents using either *accept proposal* or *reject proposal* performatives (Figure 13).

#### 4.3.4. Inform

The inform-only protocol is used by resource and parts agents to inform the SMA of any changes in their states.

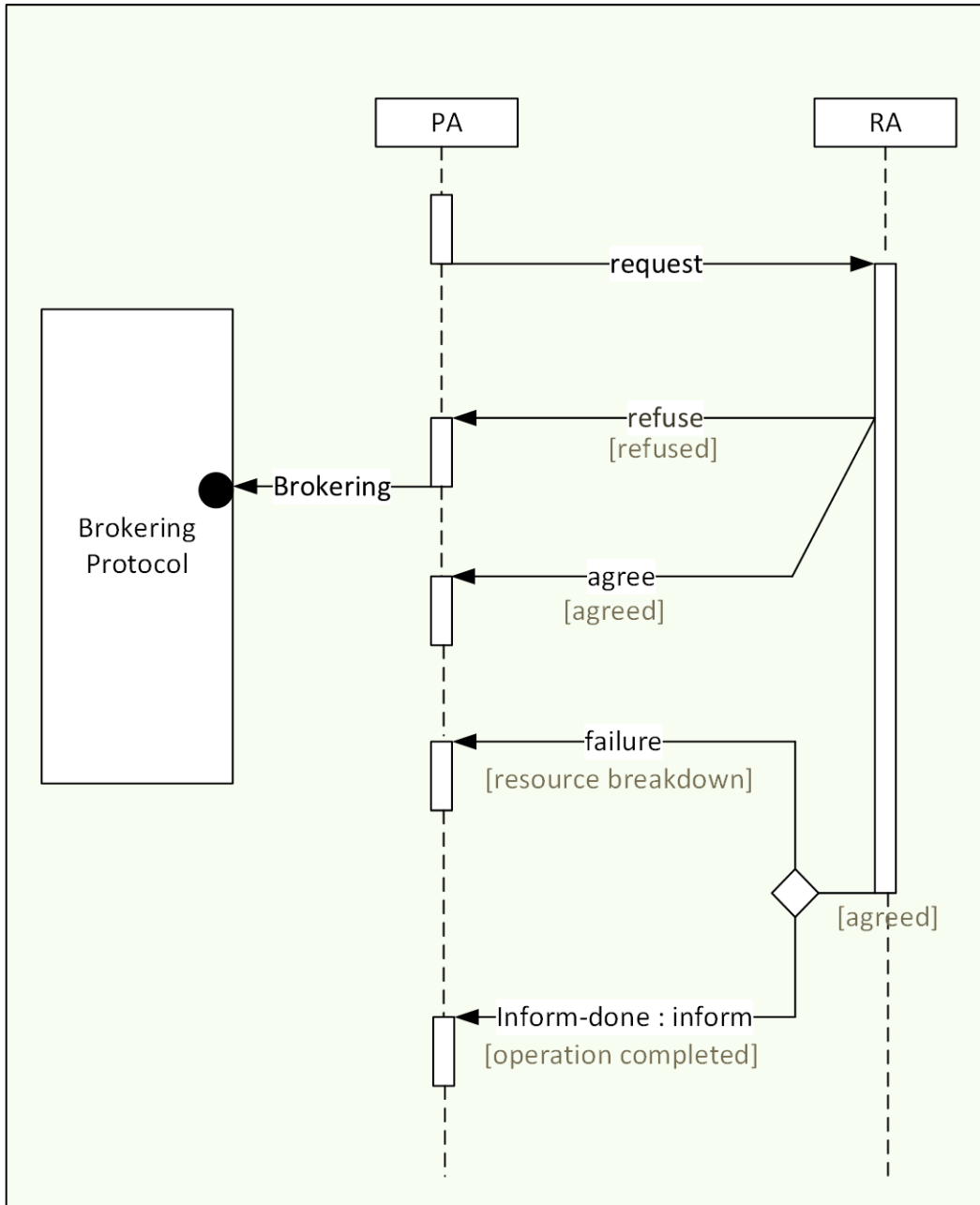


Figure 11 Request interaction protocol

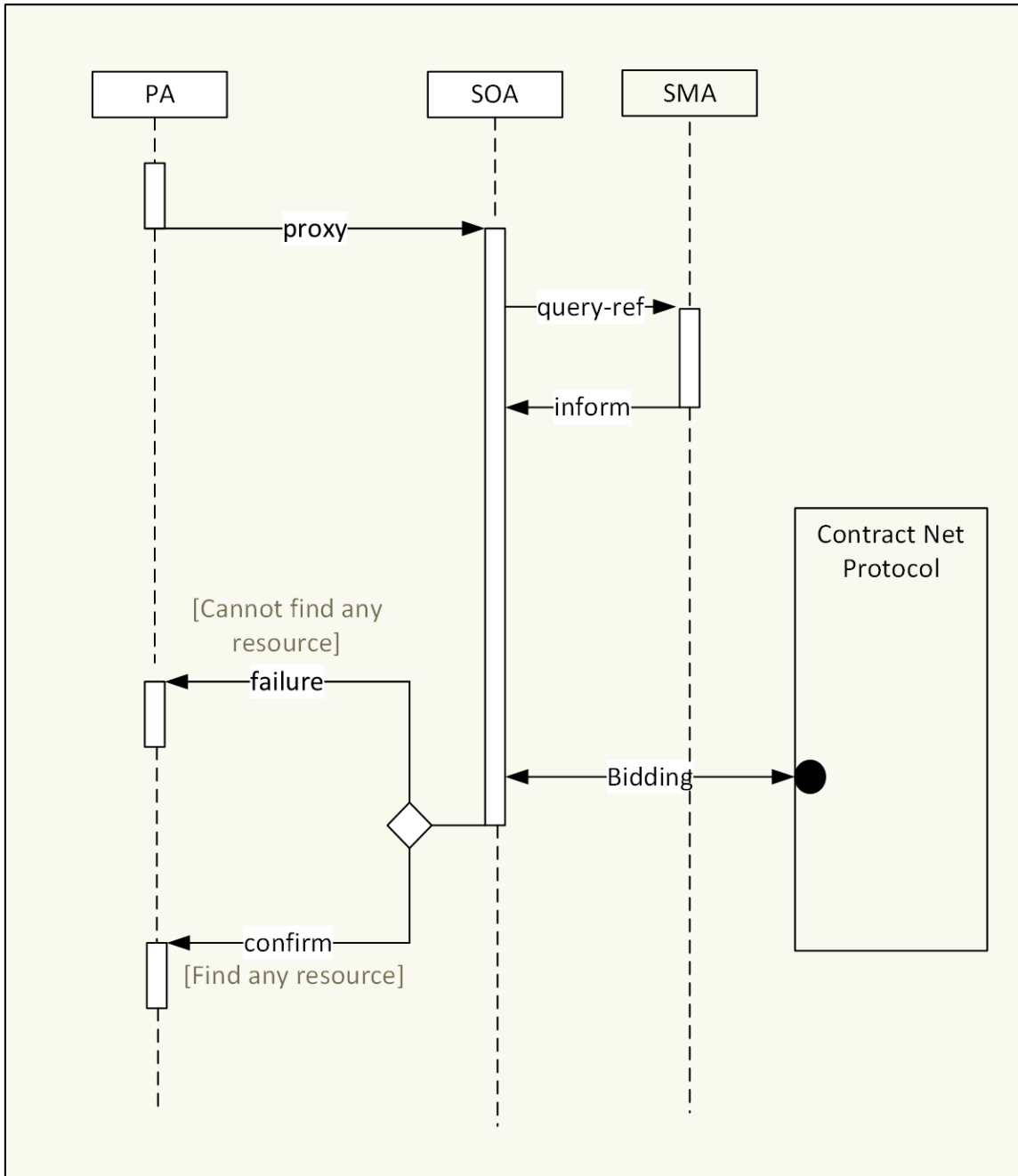


Figure 12 Brokering interaction protocol

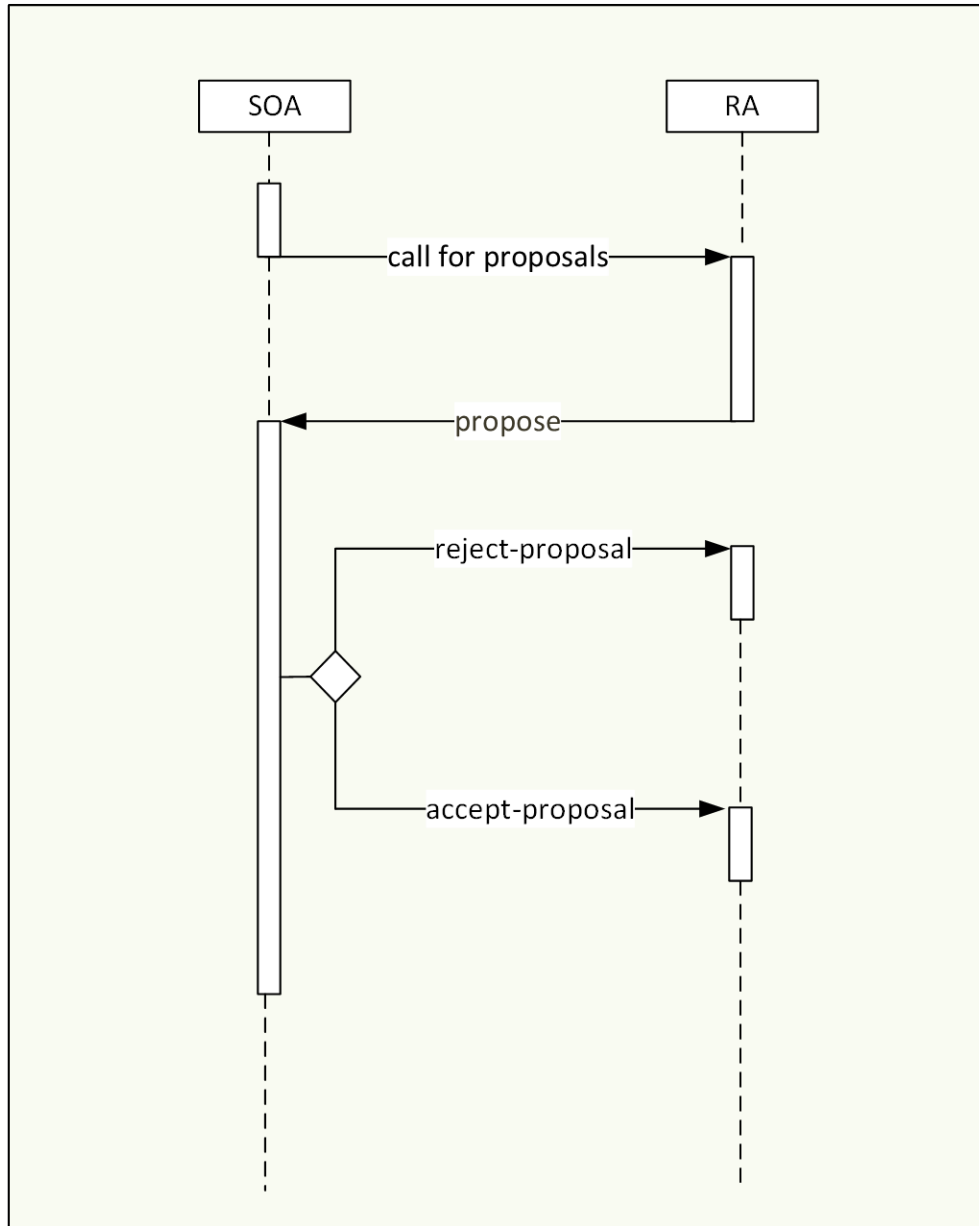


Figure 13 Contract net interaction protocol

#### 4.4. Agents States, Transitions, Actions and Events

The basic elements of discrete time simulation are states, transitions, actions and events. The global state of a manufacturing system is decided by the status of the distributed entities at various levels. A transition indicates that if a specified trigger event occurs, entities switch from one state to another and performs a specified action. Events are used to model delays and timeouts for actions scheduling. In this section, the possible states, transitions, actions and events of entities are described.

#### 4.4.1.States

The *Optimisation Agent* and *Shop Management Agent* are passive agents in that they only execute events and react to message arrivals, but do not maintain states.

A *Resource Agent* can be in *Idle*, *Busy* or *Down* states. The *Idle* and *Busy* are simple states that are embedded in *Operational* composite state (Figure 14). The states are explained as follows:

- *Idle*: The resource is in this state when it is waiting for a part to be processed.
- *Busy*: The resource in this state when it is processing a part.
- *Down*: The down state is the period from breakdown of the resource to its recovery

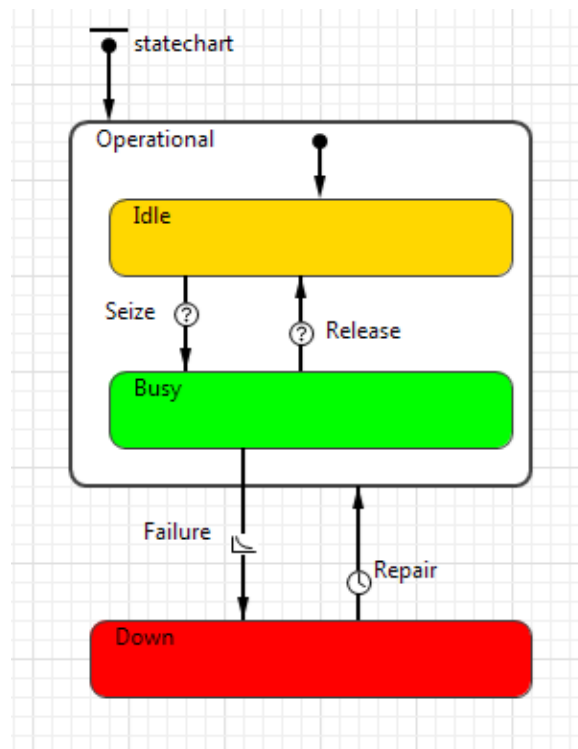


Figure 14 Resource agent state chart

A *Part Agent* can be in one of many states, but the key ones are *NotReleased*, *NegotiatingWithResourceAgent*, *NegotiatingWithOptimisationAgent*, *QueingForProcessing*, *OnResource* and *Complete* (Figure 15). The states are explained as follows:

- *NotReleased*: The agent is in this state when the simulation time has not reached its release time attribute value

- *NegotiatingWithResourceAgent*: The agent is in this state when it is negotiating directly with resource agents to find the best offers for executing specific operations. This is a composite state consisting of *WaitingForNextOperationToStart*, *SendingRequestToResourceAgent* and *WaitingForResourceAgentReply* sub-states.
- *NegotiatingWithOptimisationAgent*: The agent is in this state when it contacts the optimisation agent to find resources on its behalf. It consists of *SendingRequestToOptimisationAgent* and *WaitingForOptimisationAgentReply* sub-states.
- *QueingForProcessing*: The agent is in this state when an agreement has been reached with a resource, but the part cannot be processed immediately (i.e. waiting on queue).
- *OnResource*: The agent is in this state when it is being processed on a resource. It stays in this state for the duration of its processing time on that resource.
- *Complete*: The agent transitions to this state when all its operations have been completed.

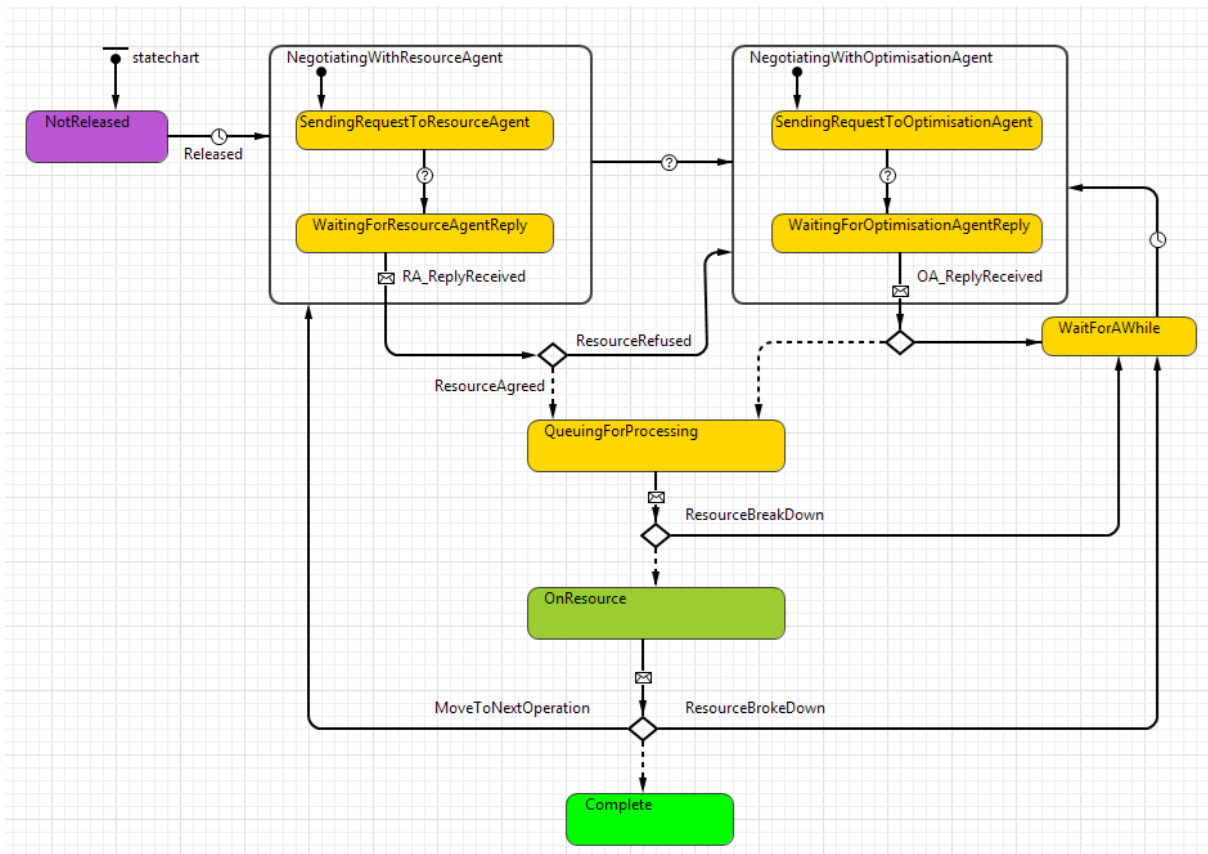


Figure 15 Part agent state chart



#### 4.4.2. Events, Actions and Transitions

A *Resource Agent* has *process messages* and *process reservations* background events, scheduled to run at specified rates. The *process messages* event is used to process incoming messages stored in its message queue by the messaging mechanism of the agent platform. A part is added to the resource reservation queue whenever there is an agreement to process the part. The *process reservations* event is used to process the resource reservation queue. The following events, actions and transitions are initiated by a resource agent:

- *Seize*: The *seize* transition is fired by the *process reservation* event when the resource agent is in *Idle* state, immediately after sending an invitation message to a selected part from its reservation queue based on a defined priority rule. When this transition occurs, the resource agent changes to *Busy* state and the concerned part changes from *QueingForProcessing* state to *OnResource* state.
- *Release*: The *release* transition occurs when a part exits a resource, the resource state changes from *Busy* to *Idle*. An *inform* message is sent to the part agent and its state changes from *OnResource* to either *NegotiatingWithResourceAgent* or *Completed* state depending on the process status of the part.
- *Failure/Repair*: Resource failure is modelled as a stochastic event with known probability distribution. It is assumed that failure occur only when a resource is in *Busy* state, changing its state to *Down*. A *repair* event is triggered after the time it takes to repair the resource and the resource transitions to *Operational (Idle)* state. The *failure* and *repair* transitions are triggered at rates defined using a triangular distribution function. Meanwhile, the interrupted part receives a message to this effect and then transitions to *NegotiatingWithOptimisationAgent* state to find an alternative resource. Part agents that are waiting in the resource queue also receive the failure message.

#### 4.5. Routing and Sequencing Rules

In flexible scheduling, routing and sequencing rules are needed to deal with machine selection from a set of alternative machines as well as determining the order in which machines should process the parts waiting in their queues to get the most optimum results. It is not always feasible to find optimal solutions in dynamic environments because greedy choices are made at each step to ensure that the objective function is optimised. Decisions are made on the fly and it impossible to reverse the decisions.

The priority sequencing rules that are commonly used in operations planning and scheduling are first-come, first-served (FCFS), shortest processing time (SPT), minimum number of operations (Min NoP) and earliest due date (EDD). There are other variants of processing time, number of operations and due dates related priority rules. To use these priority rules, information on each part's processing requirements, due dates, operations completed, and the remaining operations are required. Identifying the best priority rule to use at a particular stage in the process is a complex problem because the rule applied determines the sequence of operation, which in turn determines the sequences of operations downstream.

As explained in section 2, the initial schedule created by the genetic algorithm has machine assignment, start time and finish time for every operation in a part's process flow. The start time of every operation creates a new priority rule known as the earliest operation due time (EODT). When the optimisation agent receives brokering requests for an operation from multiple part agents, the part with the EODT is first pushed to the resource that provides the least weighted sum of operation tardiness and energy costs. The resources process the parts in their queue using FCFS rule to respect earlier commitments. Parts that are introduced after the initial schedule has been generated do not have operation due time. In this case, the optimisation agent uses the overall due time to establish priority.

## 5. Simulator Implementation and Results

A simulator is required for analyzing the system behavior and its performance, compare alternative system design and to determine the effects of alternative policies on system performance. This section focuses on the development of a simulator for evaluating the algorithms and solutions developed in Task 4.2. The simulator was developed in AnyLogic

### 5.1. Model Classes

Agents are main building blocks of AnyLogic models, which are created by extending the in-built *Agent* class. The base *Agent* class provides functionalities for sending and receiving messages, events handling and states maintenance. Within an agent, variables, parameters, events, state charts and process flow charts can be defined. An agent can execute multiple operations at the same time and the message passage mechanism routes messages appropriately to the connected state chart.

Agent can represent a single agent or population of agents. A population represents a collection of agents of the same type. For instance, the SMA and SOA are single agents whereas the RA and PA are population of agents. The following classes are defined in the AnyLogic simulator:

**Performatives:** The *Performative* is an enumeration type that defines the performative constants listed above.

**Message:** The *Message* class is a representation of the ACL message for communicating between agents. This is required because AnyLogic does not have a specific class for creating message objects. The attributes contained in the class are:

- Sender: The agent that sends the message
- Receiver: The recipient agent of the message
- Performative: The intention of the sender (e.g. *Request*, *Inform*)
- Content: The main content of the message
- ContentObject: This is used to attach a complex type to a message
- ConversationId: Is used to link messages in the same conversation. This enables agents to identify individual conversations and to reason across historical records of conversations

**PartAgent:** The *PartAgent* class inherits AnyLogic *Agent* class and its attributes are as defined in the data model. The class also has variables for status tracking.

**ResourceAgent:** The *ResourceAgent* class inherits AnyLogic *Agent* class and its attributes are as defined in the data model. The class also has variables for status tracking and part queue management.

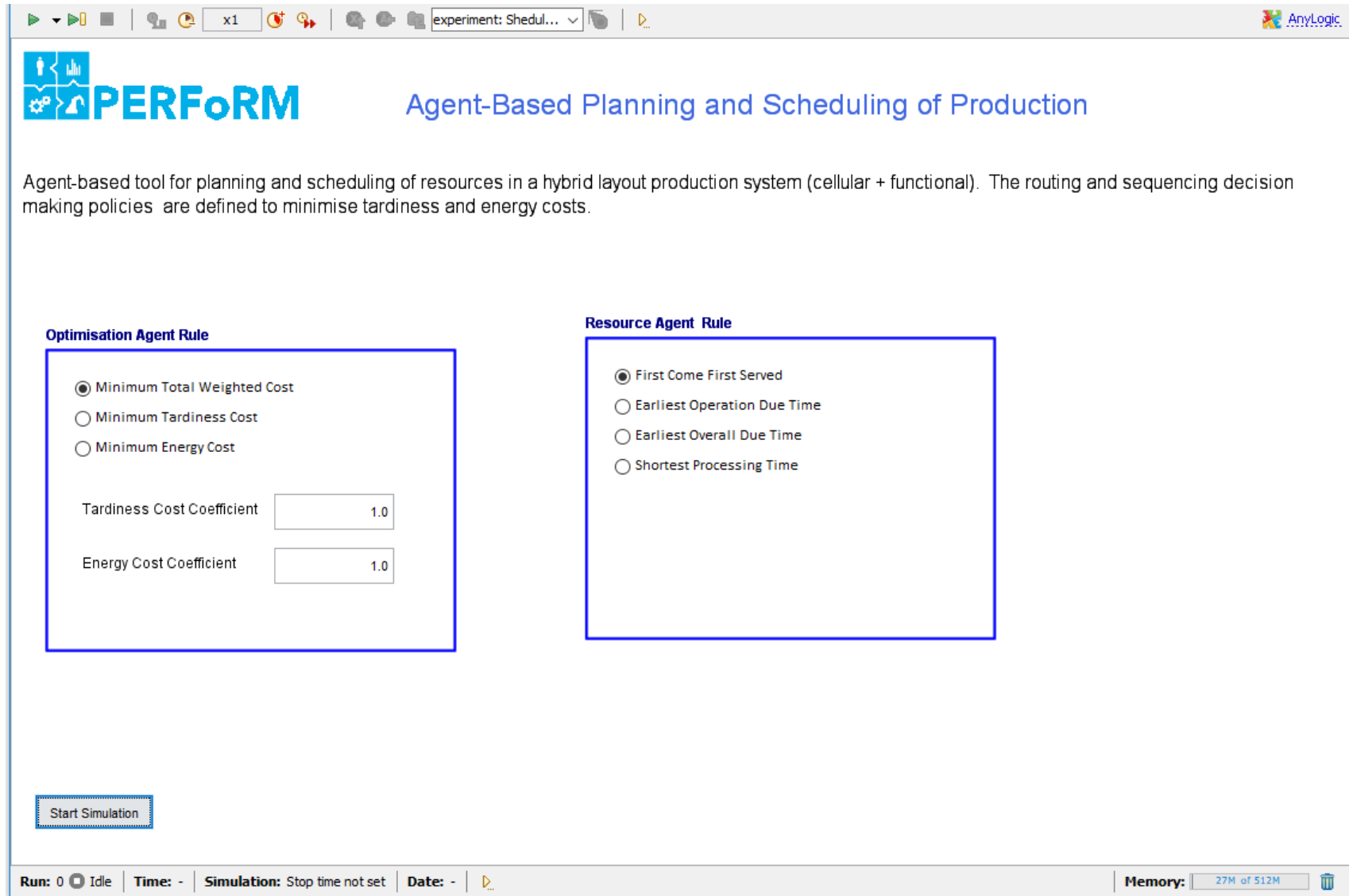
**OptimisationAgent:** The *OptimisationAgent* class inherits AnyLogic *Agent* class.

**ShopManagementAgent:** The *ShopManagementAgent* class inherits *AnyLogic Agent* class. It has parameters to maintain the list and states of all agents in the platform.

*State* represents a location of control with a set of reactions to conditions and/or events. *Events* are used to schedule actions in AnyLogic. All the classes have events that are fired at specified rates for processing incoming and outgoing messages. The interaction between different agents are being handled by the AnyLogic message passing mechanism. Sophisticated time-driven behaviour that cannot be defined using events are implemented using *Statechart*. State charts are used to visually capture the discrete states of parts and resources agents. Transitions in state charts are triggered by user-defined conditions.

AnyLogic models have in-built integrated relational database for reading input data and writing simulation outputs. The UML class diagram designed in Section 2 was transformed into a relational data model (database tables) for the AnyLogic simulator. The needed data are stored in the database and the population of parts and resources agents are created dynamically from the database during runtime.

The main page of the simulator is shown in Figure 16. It has been designed in such a way that allows a user to test various optimisation criteria and dispatch rules. Sample plant layout page generated based on the data stored in the database is shown in Figure 17. Sample state transitions for a resource and part agent during simulation are also shown in Figure 18 and Figure 19 respectively.



The screenshot shows the main interface of the PERFoRM simulator. At the top, there is a toolbar with various icons and a window title 'experiment: Schedul...'. The main header features the PERFoRM logo and the title 'Agent-Based Planning and Scheduling of Production'. Below the header, a descriptive paragraph states: 'Agent-based tool for planning and scheduling of resources in a hybrid layout production system (cellular + functional). The routing and sequencing decision making policies are defined to minimise tardiness and energy costs.'

The interface is divided into two main configuration panels:

- Optimisation Agent Rule:** This panel contains three radio button options:
  - Minimum Total Weighted Cost
  - Minimum Tardiness Cost
  - Minimum Energy Cost
 Below these options are two input fields:
  - Tardiness Cost Coefficient: 1.0
  - Energy Cost Coefficient: 1.0
- Resource Agent Rule:** This panel contains four radio button options:
  - First Come First Served
  - Earliest Operation Due Time
  - Earliest Overall Due Time
  - Shortest Processing Time

At the bottom left, there is a 'Start Simulation' button. The bottom status bar displays the following information: 'Run: 0', 'Idle', 'Time: -', 'Simulation: Stop time not set', 'Date: -', and 'Memory: 27M of 512M'.

Figure 16 Simulator main page

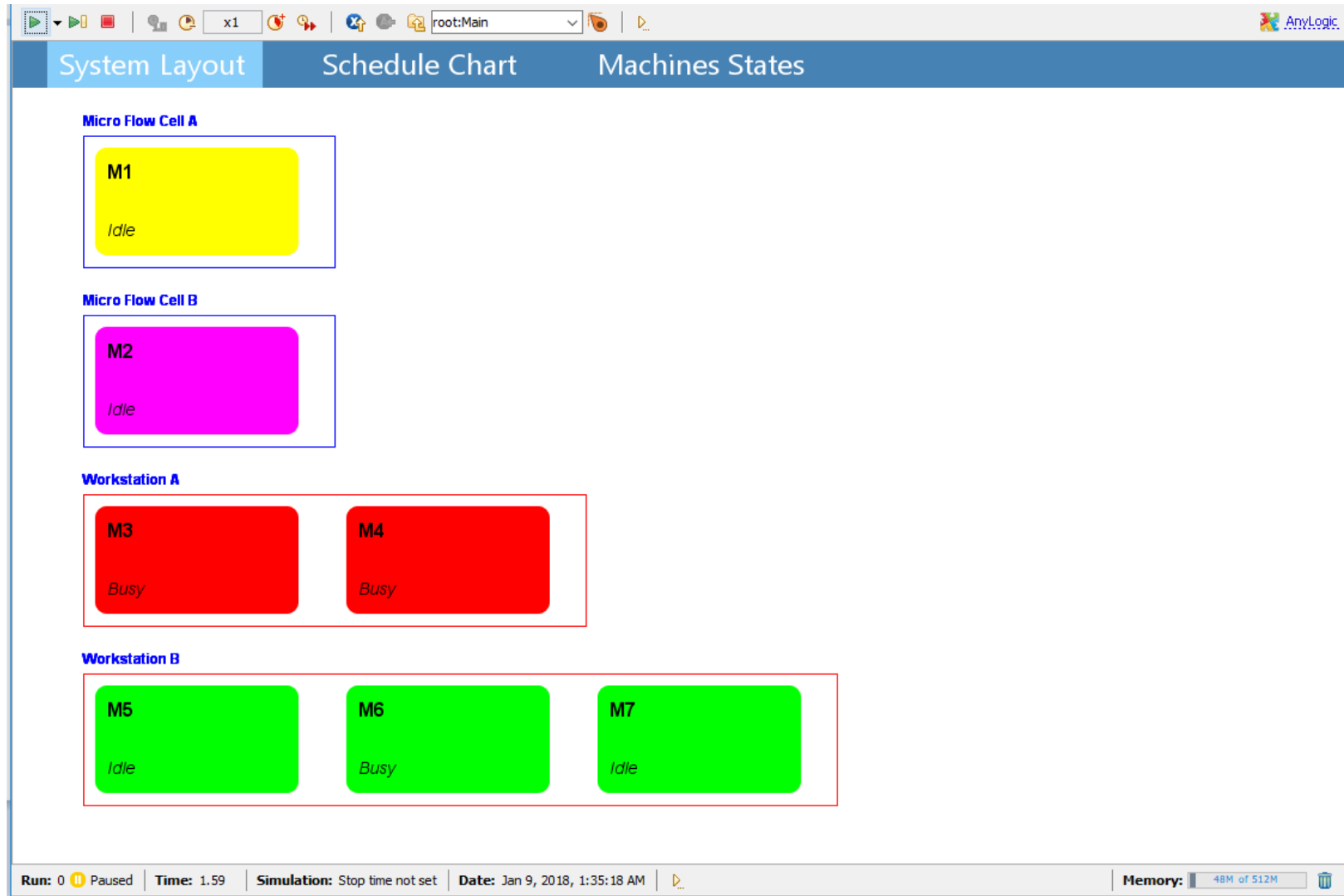


Figure 17 Plant layout in simulator

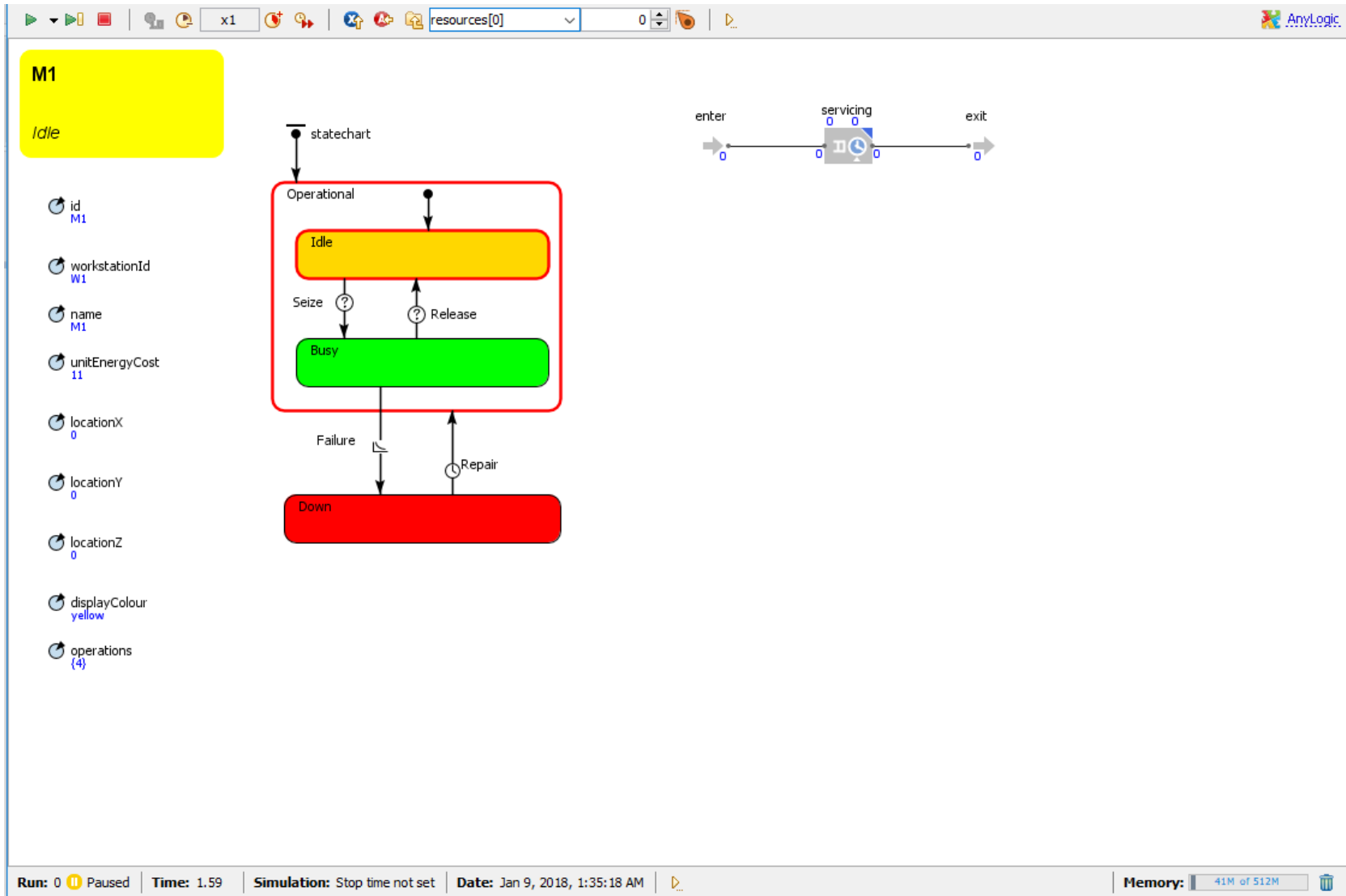


Figure 18 Resource agent state transition during simulation

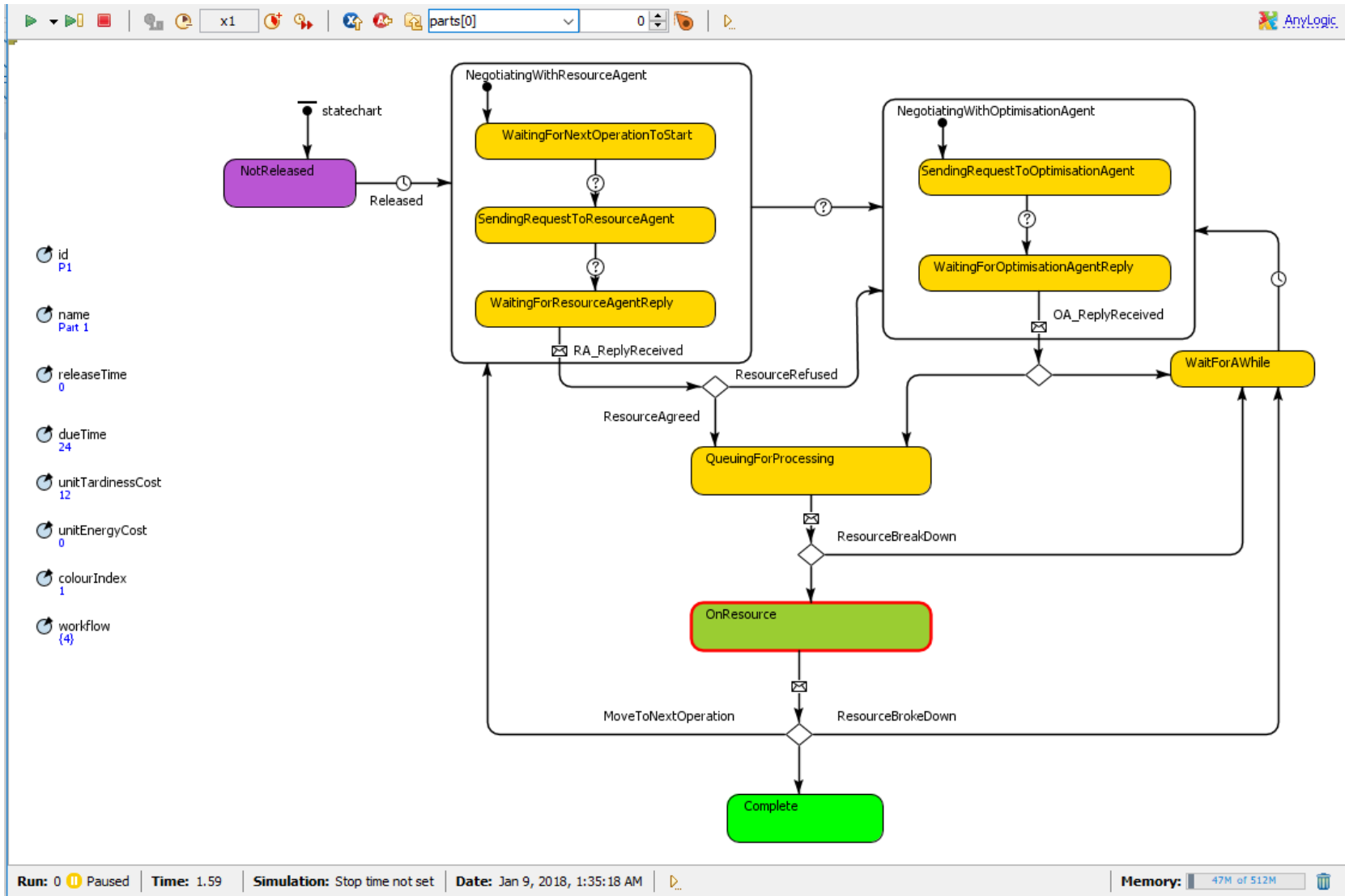


Figure 19 Part agent state transition during simulation



## 5.2. Simulation Results

The resources, operations and parts data used for test simulation are shown in Tables 4 to 9.

**Table 4 Resources, operations and energy costs data**

Resource	Operations	Location	Energy Cost /Unit Time
$M_1$	Cleaning, CMM, Deburring, Inspection	Micro-Flow Cell A	11
$M_2$	Cleaning, CMM, Deburring, Inspection	Micro-Flow Cell B	10
$M_3$	CNC/Milling A	Workstation A	13
$M_4$	CNC/Milling A	Workstation A	11
$M_5$	CNC/Milling B	Workstation B	9
$M_6$	CNC/Milling B	Workstation B	8
$M_7$	CNC/Milling B	Workstation B	12

**Table 5 Parts due time and tardiness costs data**

Part	Due Time	Tardiness Cost/Unit Time
Part 1	48	12
Part 2	58	15
Part 3	40	10
Part 4	52	20

**Table 6 Processing time of Part 1**

Part 1							
Operation	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$
CNC/Milling A ( $O_{11}$ )	-	-	6	6	-	-	-
CMM ( $O_{12}$ )	6	8	-	-	-	-	-
Inspection ( $O_{13}$ )	10	12	-	-	-	-	-
CNC/Milling B ( $O_{14}$ )	-	-	-	-	12	14	10

Table 7 Processing time of Part 2

Part 2							
<i>Operation</i>	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$
CNC/Milling B ( $O_{21}$ )	-	-	-	-	6	10	4
Deburring ( $O_{22}$ )	14	18	-	-	-	-	-
CMM ( $O_{23}$ )	8	12	-	-	-	-	-
CNC/Milling A ( $O_{24}$ )	-	-	22	24	-	-	-

Table 8 Processing time of Part 3

Part 3							
<i>Operation</i>	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$
CNC/Milling B ( $O_{31}$ )	-	-	-	-	16	18	12
Cleaning ( $O_{32}$ )	10	14	-	-	-	-	-
CNC/Milling A ( $O_{33}$ )	-	-	4	6	-	-	-

Table 9 Processing time of Part 4

Part 4							
<i>Operation</i>	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$
CNC/Milling A ( $O_{41}$ )	-	-	16	22	-	-	-
Cleaning ( $O_{42}$ )	4	8	-	-	-	-	-
CMM ( $O_{43}$ )	10	12	-	-	-	-	-
CNC/Milling B ( $O_{44}$ )	-	-	-	-	22	26	20

The data show in the above tables are used as inputs to the genetic algorithm optimiser developed in Section 3. The energy and tardiness cost coefficients are taken to be 1 in this case. The chosen population size is 100, which was evolved over 50 generations. The optimal allocation of resources to parts and the associated schedule generated from the algorithm are shown in Tables 10 and 11 respectively.

**Table 10 Optimal allocation of resources to parts operations generated from genetic algorithm**

<b>Resource</b>	<b>Operations</b>
$M_1$	$O_{22}, O_{42}, O_{43}, O_{23}$
$M_2$	$O_{12}, O_{32}, O_{13}$
$M_3$	$O_{41}, O_{24}$
$M_4$	$O_{11}, O_{33}$
$M_5$	$O_{44}$
$M_6$	$O_{31}$
$M_7$	$O_{21}, O_{14}$

**Table 11 Optimal schedule generated from genetic algorithm**

<b>Part</b>	<b>Operation</b>	<b>Resource</b>	<b>Start Time</b>	<b>Finish Time</b>
Part 1	$O_{11}$	$M_4$	0	6
Part 1	$O_{12}$	$M_2$	6	14
Part 3	$O_{31}$	$M_6$	0	18
Part 4	$O_{41}$	$M_3$	0	16
Part 2	$O_{21}$	$M_7$	0	4
Part 3	$O_{32}$	$M_2$	18	32
Part 2	$O_{22}$	$M_1$	4	18
Part 4	$O_{42}$	$M_1$	18	22
Part 1	$O_{13}$	$M_2$	32	44
Part 3	$O_{33}$	$M_4$	32	38
Part 4	$O_{43}$	$M_1$	22	32
Part 1	$O_{14}$	$M_7$	44	54
Part 4	$O_{44}$	$M_5$	32	54
Part 2	$O_{23}$	$M_1$	32	40
Part 2	$O_{24}$	$M_3$	40	62

The resources, operations, parts and optimised schedule data were supplied to the AnyLogic simulator. The first test involves running the simulator without any disturbances (resource breakdown). The schedule produced from the agents’ interaction is shown in Figure 20 and the corresponding states of the resources are shown in Figure 21. The parts and their corresponding colour keys are given in Table 12.

**Table 12 Schedule Gantt chart color keys**

<b>Part</b>	<b>Colour</b>
Part 1	
Part 2	
Part 3	
Part 4	

As seen in Figure 20, the agents followed the optimised schedule without any deviations. The results of the schedule are shown in Table 13. The test was repeated under the same conditions but without providing the agents with any initial optimised schedule. The output schedule and the corresponding states of resources are shown in Figure 22 and 23 respectively. The completion times and costs of the schedule are shown in Table 14. As seen from the result, the hybrid scheduler produced better solution with respect to the performance criteria than the greedy decision-making scheme. Due to the greedy decision-making scheme of the agents in the non-optimised schedule, Part 3 has no tardiness costs (early) but Part 4 has a very high tardiness costs. Although the chosen performance criteria are energy and tardiness costs, the hybrid solution is also better with respect to other measures such as average total completion time, makespan, average tardiness and maximum tardiness.

**Table 13 Optimised schedule costs**

<b>Part</b>	<b>Completion Time</b>	<b>Tardiness</b>	<b>Tardiness Cost</b>
Part 1	59	11	121
Part 2	70	12	180
Part 3	42	2	20
Part 4	61	9	180
<i>Total Tardiness Cost: 501</i>			
<i>Total Energy Cost: 1872</i>			
<i>Total Cost: 2373</i>			

**Table 14 Non-optimised schedule costs**

<b>Part</b>	<b>Completion Time</b>	<b>Tardiness</b>	<b>Tardiness Cost</b>
Part 1	62	14	168
Part 2	63	5	75
Part 3	39	0 (early by 1)	0
Part 4	75	23	460
<i>Total Tardiness Cost: 703</i>			
<i>Total Energy Cost: 2048</i>			
<i>Total Cost: 2751</i>			

Another test was conducted under resource failure conditions; the resultant schedule and the corresponding states of the resources are shown in Figure 24 and 25 respectively. As seen in Figure 24, the agents followed the optimised schedule until the first failure occurred on  $M_1$ . After this time, the optimisation agent is called upon to find alternative resources when a part is unable to make a reservation with the resource that has been assigned to it in the initial schedule.

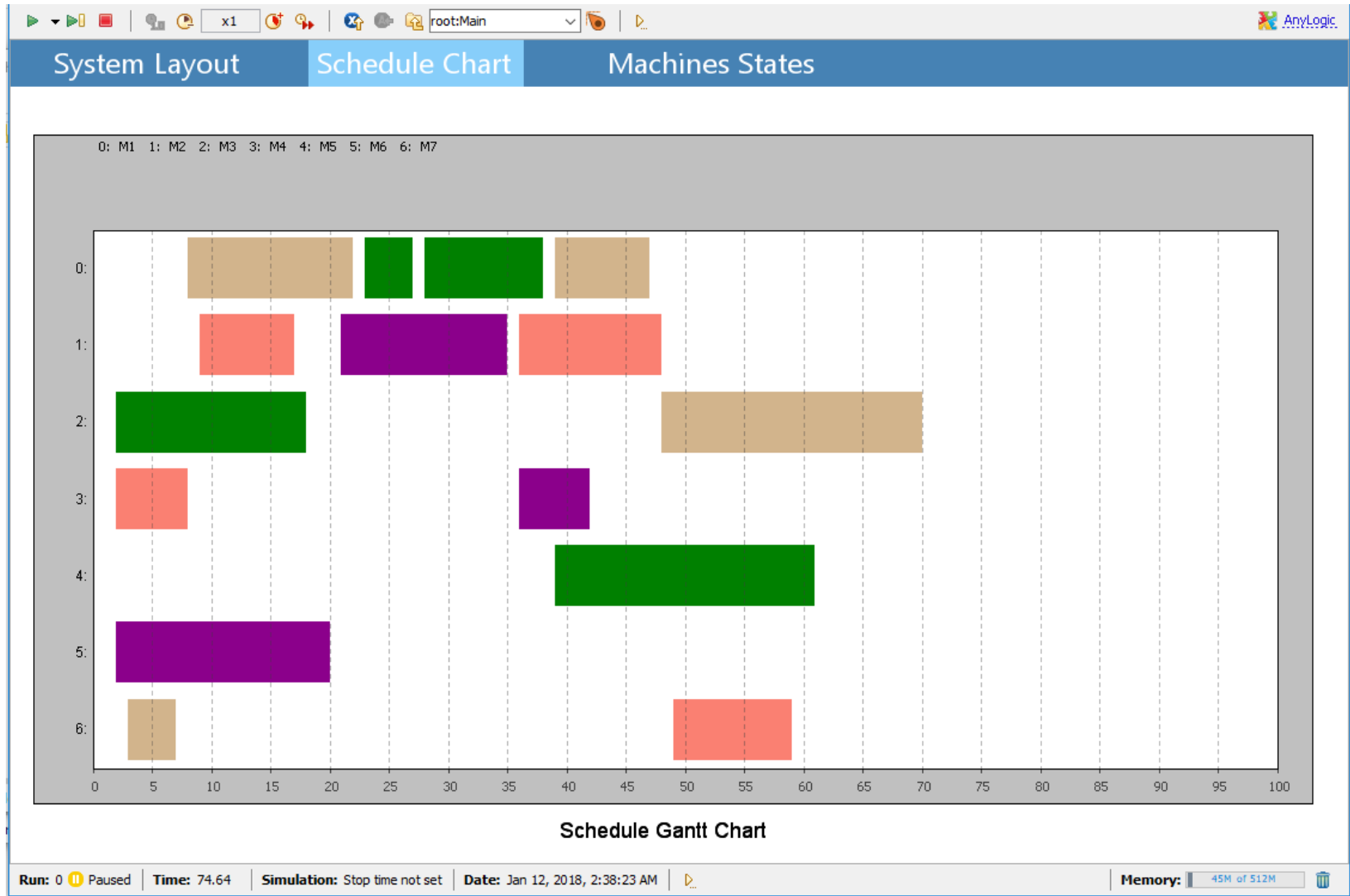


Figure 20 Gantt chart for optimised schedule

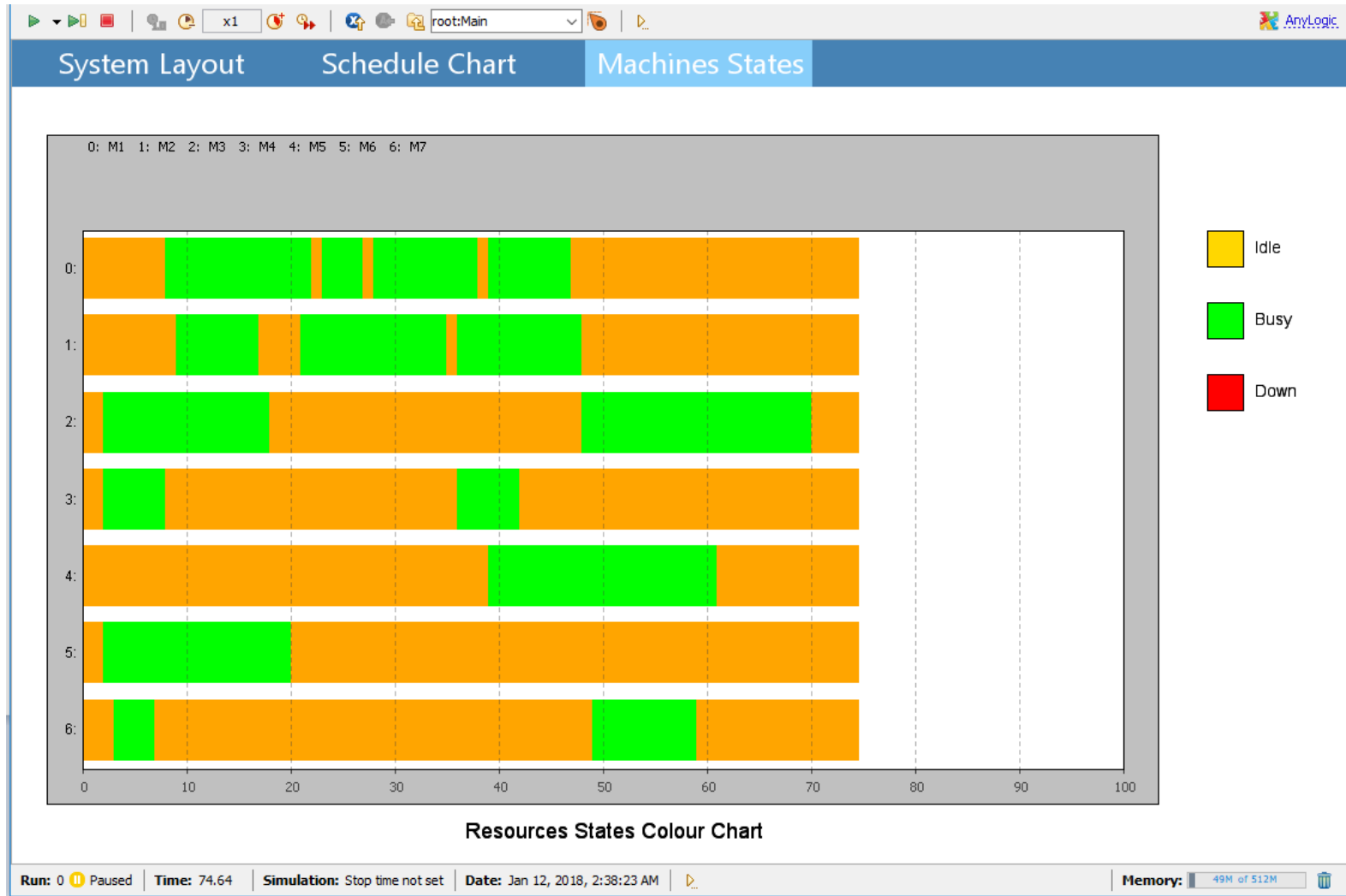


Figure 21 Resources states chart for optimised schedule

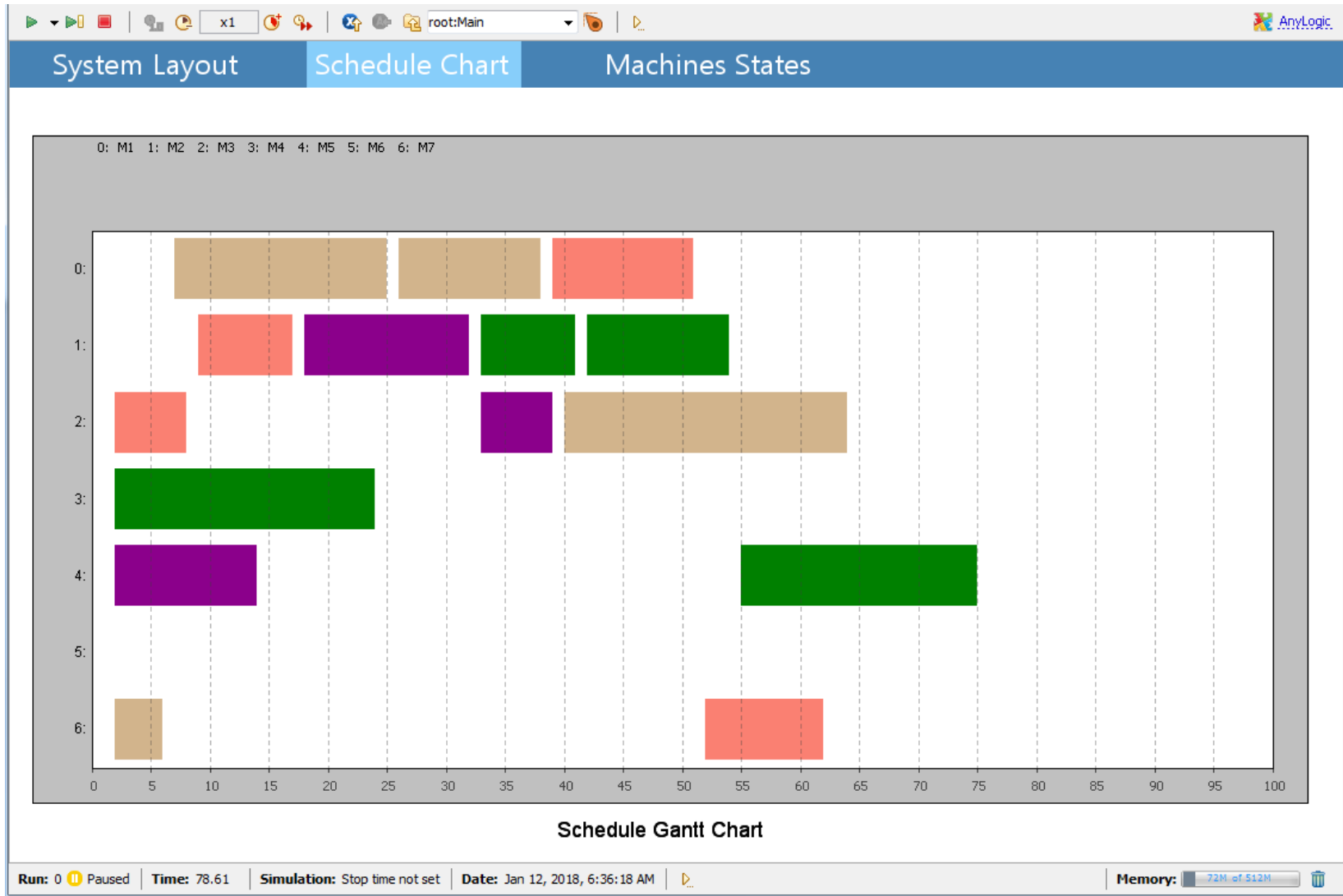


Figure 22 Gantt chart for non-optimized schedule



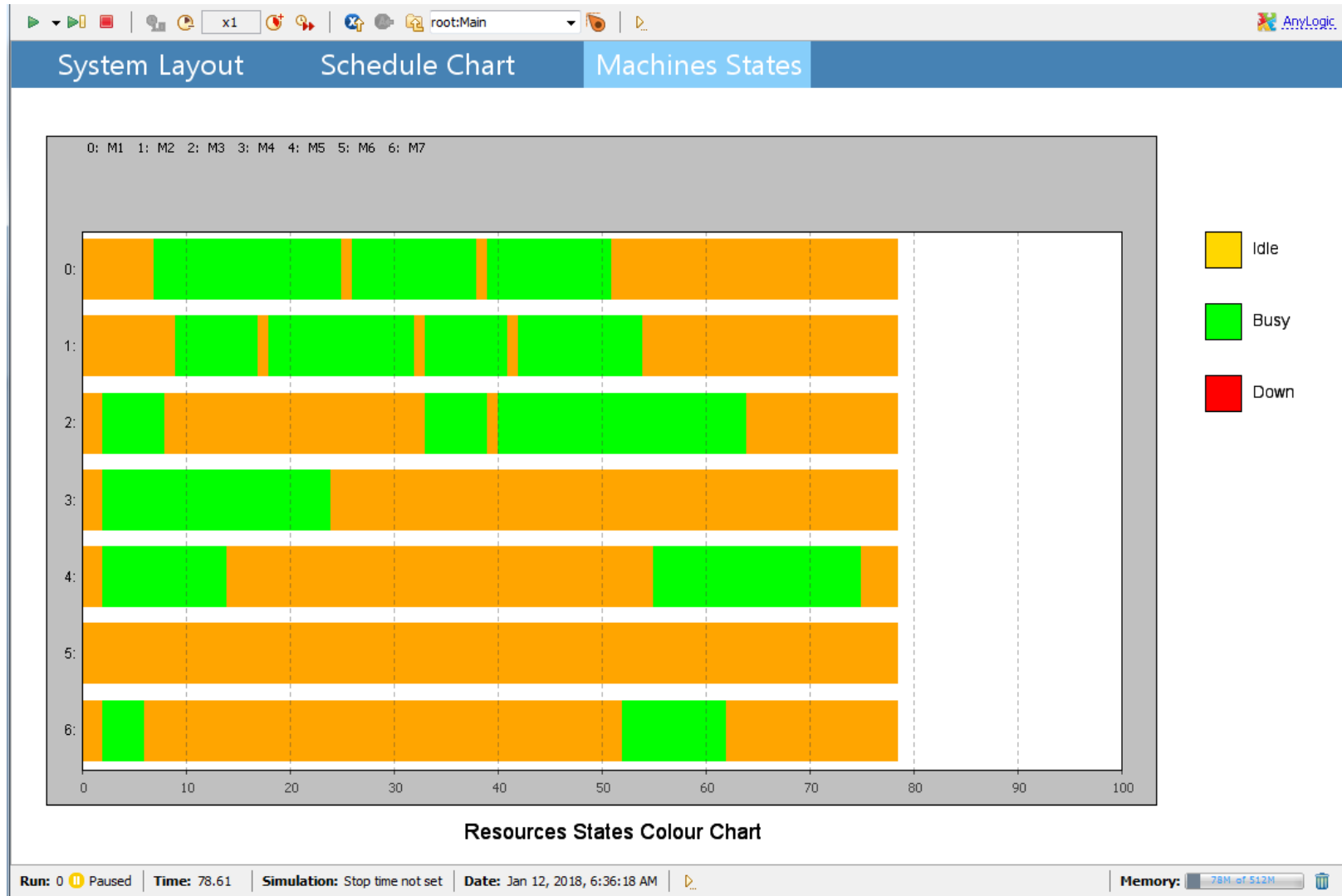


Figure 23 Resources states chart for non-optimised schedule

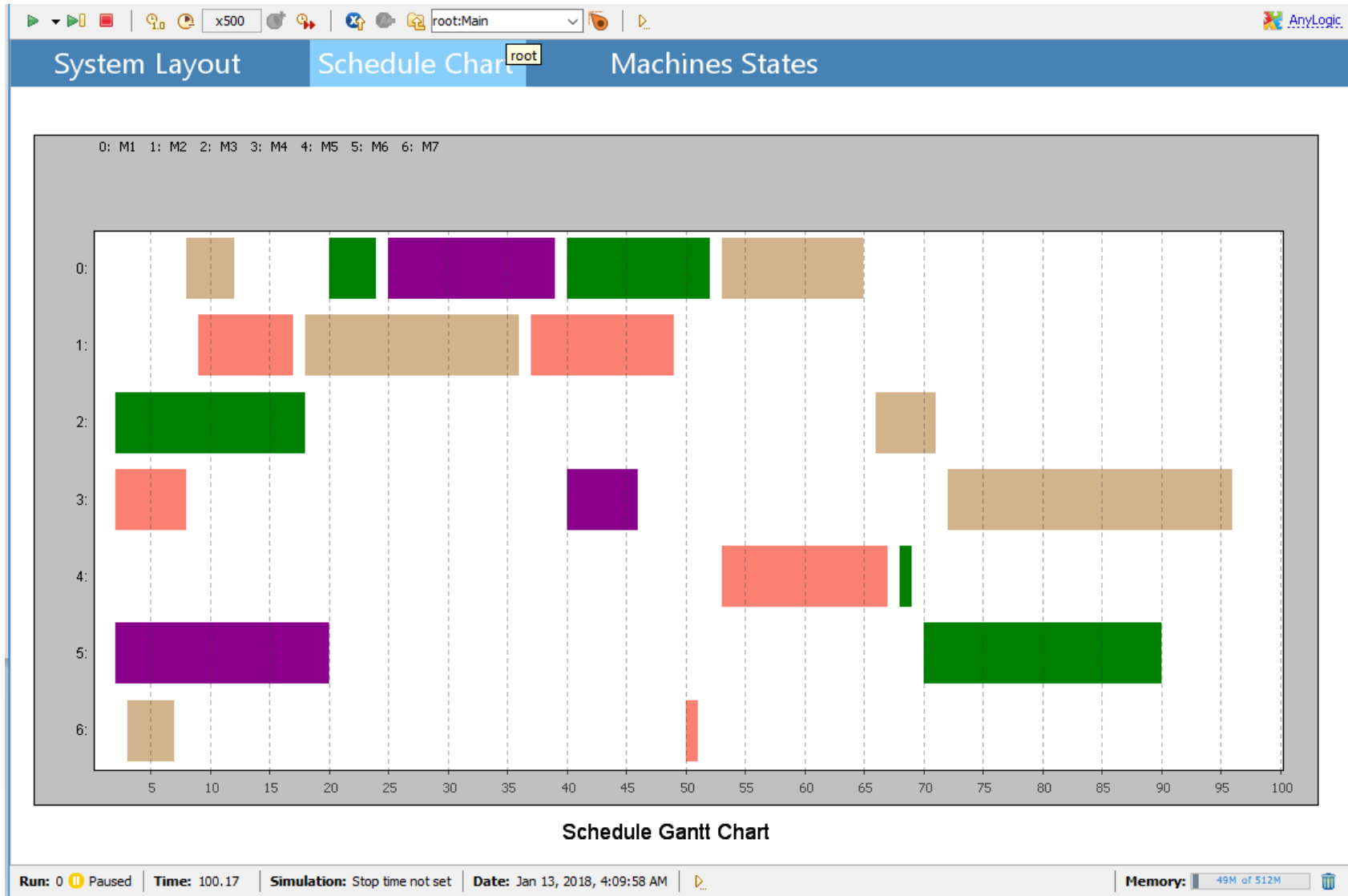


Figure 24 Gantt chart for schedule under dynamic resource failure conditions

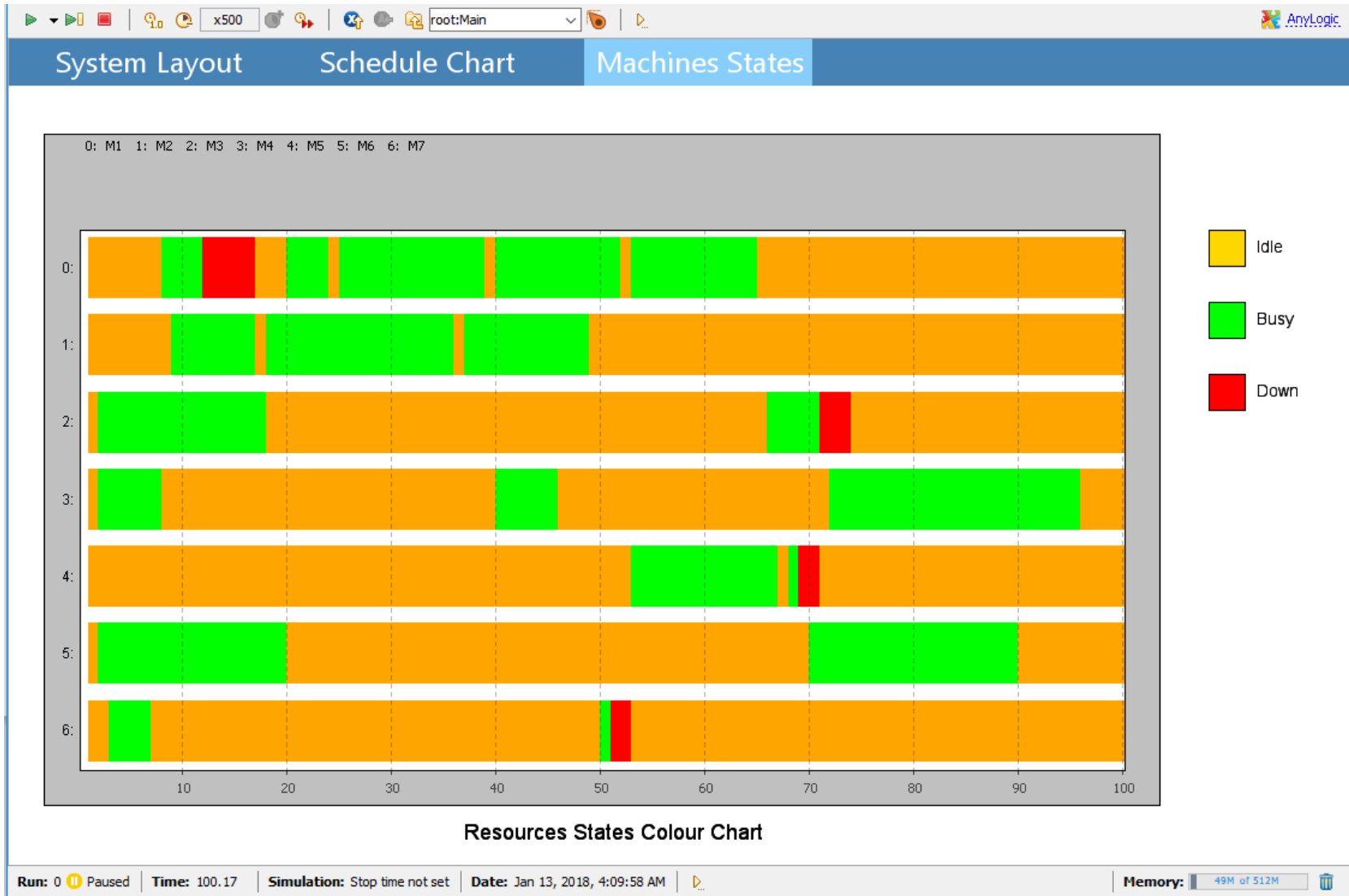


Figure 25 Resources states chart for schedule under dynamic failure conditions

## 6. Conclusions

Traditional centralised scheduling solutions produce optimal results, however, it is well-known that they cannot cope with dynamic disturbances such as machine failure. Multiagent systems provide solutions to this problem but they also suffer from lack of optimisation due to their greedy decision-making schemes. A multi-agent based scheduling system that is enhanced using a central optimisation technique has been developed in this task for a combination layout (functional and cellular) flexible job-shop system.

In the proposed method, a genetic algorithm is used to generate an initial optimised schedule to be executed by the agents. The weighted sum of total energy and tardiness costs was used as the performance objective. During simulation, the agents attempt to follow the schedule unless a disruption occurs, in which case, the part agents find alternative resources to fulfil their needs by interacting with the relevant agents in the system. A simulator for evaluating the performance of the proposed concept was created using AnyLogic software. Although the software is not compatible with FIPA standards for developing multi-agent systems, a custom framework was developed to conform to the standard.

Some simulation tests were conducted, and the results show that the output schedule has lower total costs when the agents are provided with an initial optimised schedule (hybrid solution), compared to when the agents make greedy decisions throughout. It was also shown that the agents interact to ensure that a feasible schedule is always produced when resource breakdown occurs.

The developed concept will be validated and demonstrated in WP10. To achieve this, a production-ready system developed using JADE and WADE frameworks will be integrated with the solutions developed in other WPs.

## 7. References

- [1] Gen, M., and Lin, L. (2014) “Multi-objective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey,” *Journal of Intelligent Manufacturing*, 25 (5), 849 – 866
- [2] Liu, Y., Dong, H., Lohse, N. Petrovic, S. and Gindy N. (2014). “An investigation into minimising total energy consumption and total weighted tardiness in job shops”. *Journal of Cleaner Production*, 65, 87 – 96
- [3] Wang, L., Cai, J., Li, M. and Liu Z. (2017). “Flexible job shop scheduling problem using an improved ant colony optimisation”. *Scientific Programming*, 9, ISSN: 1058-9244
- [4] Zhao, B., Gao, J., Chen, K., and Guo, K. (2015) “Two-generation Pareto ant colony algorithm for multi-objective job shop scheduling problem with alternative process plans and unrelated parallel machines,” *Journal of Intelligent Manufacturing*, 26, (3), 1–16
- [5] Shel, L., Dauzere-Peres, S. and Neufeld, J. S. (2018) “Solving the flexible job shop scheduling problem with sequence-dependent setup times”, *European Journal of Operational Research*, 265, (2), 503-516
- [6] Ouelhadj, D. and Petrovic, S. (2009). “A survey of dynamic scheduling in manufacturing system”. *Journal of Scheduling*, 12, 417- 431.
- [7] Zhou, R., Lee, H. P., and Nee, A. Y. C. (2008). “Simulating the generic job shop as a multi-agent system”. *International Journal of Intelligent Systems and Applications*, 4, (1/2), 5 - 33.
- [8] Erol, R., Sahin, C., Baykasoglu A. and Kaplanoglu V. (2012) “A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems”. *Journal of Applied Soft Computing*, 12, (6), 1720 -1732
- [9] Hsu, C., Kao, B., Ho, V. L. and Lai, K. R. (2016) “Agent-based fuzzy constraint-directed negotiation mechanism for distributed job shop scheduling”. *Engineering Applications of Artificial Intelligence*, 53, 140 - 154
- [10] Yang, B. and Geunes, J.. (2008) “Predictive-reactive scheduling on a single resource with uncertain future jobs”. *European Journal of Operations Research*, 189, (3), 1267 - 1283
- [11] Zhang, S., Wong T. N. (2017) “Flexible job-shop scheduling/rescheduling in dynamic environment: a hybrid MAS/ACO approach”. *International Journal of Production Research*, 55, (11), 3173 - 3196
- [12] Sahin, C., Demirtas M., Erol, R., Baykasoglu A. and Kaplanoglu V. (2017) “A multi-agent based approach to dynamic scheduling with flexible processing capabilities”. *Journal of Intelligent Manufacturing*, 28, (8), 1827 - 1845
- [13] Nedwed F.Y., Zinnikus I., Nukhayev M., Klusch M., Mazzola L. (2017). “shopST: Flexible Job-

Shop Scheduling with Agent-Based Simulated Trading”. In: *Berndt J., Petta P., Unland R. (eds) Multiagent System Technologies. MATES 2017. Lecture Notes in Computer Science, vol 10413.* Springer, Cham, ISBN 978-3-319-64797-5

- [14] Deliverable D10.1 – “GKN Use Case goals, requirements and KPIs – Specifications of applications, functions and requirements for the Micro-Flow Cell”. *PERFoRM Project*, 2016
- [15] Deliverable D10.2 – “Concept development of the Micro-Flow Cell”. *PERFoRM Project*, 2017
- [16] Jain, A., Jain, P. K., Chan, F. T. S. and Singh, S. (2013) “A review on manufacturing flexibility”. *International Journal of Production Research*, 51 (19), 5946 – 5970
- [17] Liu, Y., Dong, H., Lohse, N. and Petrovic, S. (2016) “A multi-objective generic algorithm for optimisation of energy consumption and shop floor production performance”. *International Journal of Production Economics*, 179, 259 – 272.
- [18] He, Y., Li, Y., Wu, T. and Sutherland, J. W. (2015) “An energy-responsive optimisation method for machine tool selection and operation sequence in flexible machining job shops”. *Journal of Cleaner Production*, 87, 245 -254.
- [19] Liu, G., Zhou, Y. and Yang H. (2017) “Minimising energy consumption and tardiness penalty for fuzzy flow shop scheduling with state-dependent setup time”. *Journal of Cleaner Production*, 87, 470 - 484.
- [20] Ahmadi, E., Zandieh, M., Farrokh, M., and Emami, S. M. (2016) “A multi objective optimisation approach for flexible job shop scheduling problem under random machine break down by evolutionary algorithms”. *Computers and Operations Research*, 73, 56 -66.
- [21] Bellifemine, F., Poggi, A. and Rimassa, G. (2001) “Developing multi-agent systems with a FIPA-compliant agent framework”. *Software Practice and Experience*, 31, 103-128.
- [22] Du, X., Li, Z. and Xiong, W. (2008) “Flexible job shop scheduling problem solving based on genetic algorithm with model constraints, *IEEE International Conference on Industrial Engineering and Engineering Management*, Singapore, Singapore, Dec. 8 -11, 2008
- [23] Bussmann S., Jennings, N. R. and Wooldridge, M. J. (2004) “Multiagent systems for manufacturing control: A design methodology, *Springer Series on Agent Technology*, Springer-Verlag, Berlin Heidelberg, ISBN: 978-3-662-08872-2