Production harmonizEd Reconfiguration of Flexible Robots and Machinery

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

Deliverable 4.4

Guidelines and recommendations for reconfigurability mechanisms for machinery and robots

Lead Author: Loughborough University

Dissemination level:	PUBLIC
Date:	27/07/2018
Revision:	1.0

Version history

Version	Date	notes and comments
0.1	01/04/2018	Report structure & table of content by LBORO
0.2	10/05/2018	Completion of Chapter One by LBORO
0.3	04/05/2018	Completion of Chapter Four by IPB
0.4	25/06/2018	Completion of Chapter Two and Three by LBORO
0.5	05/07/2018	First draft ready for proofreading and internal reviews (TUBS & IPB)
0.6	09/07/2008	Abstract and conclusion completed by LBORO
1.0	27/07/2018	Incorporation of feedback from TUBS

Author List:

Phil Ogun (LBORO)
Niels Lohse (LBORO)
Lennart Bueth (TUBS)
Sebastian Thiede (TUBS)
José Barbosa (IPB)
José Dias (IPB)
Nelson Rodrigues (IPB)
Adriano Ferreira (IPB)

Abstract

A reconfigurable manufacturing system (RMS) is designed at the outset to enable rapid change in configuration (hardware, software and control components) in order to quickly and cost-effectively adjust production capacity and functionality in response to dynamic market demands. RMSs are usually designed around part families, meaning that each part family corresponds to a specific configuration and the configurations evolve over time to provide the specific functions and capacity needed to produce all the parts in a family within a given demand period.

In make-to-order environments, manufacturing systems will need to cope more with volume fluctuations and product mix variations. The sequential approach to the manufacturing of parts in different families may not be efficient over multiple generations of small batch sizes due to increased idle time and changeover costs. Higher efficiencies can be gained through concurrent manufacturing of multiple families. However, the configuration of the resources at every instant of time could either impede or facilitate the system’s operability, productivity and responsiveness. Concurrent manufacturing of parts from multiple families creates the problem of when to reconfigure and which machines should be reconfigured to guarantee schedule feasibility and improved performance of the system with respect to chosen performance measures.

The work reported in this report is focused on the development of a reconfiguration mechanism for machines and robots in a make-to-order manufacturing environment. The method developed in this task combines genetic algorithm (GA) optimisation with distributed multi-agent systems (MAS). GA is used to generate an optimal or near-optimal operations execution and reconfiguration schedule for a given demand period. The schedule contains information on allocation of parts and operations to machines, as well as information on which machines should be reconfigured and when the reconfiguration should take place. It is assumed that each part has a due date and the tardiness costs per unit time is known. Also, the efforts required to change a machine from one configuration to the other is known. The goal of the model is to minimise weighted sum of reconfiguration and tardiness costs. The reconfiguration cost can be adjusted to control the nervousness and the frequency of reconfiguration.

An agent-based reconfiguration mechanism for the logical re-organization of micro-flow production cells is also developed. Reconfigurations are triggered according to the schedule generated by the planning and scheduling model. The multi-agents direct the actual execution of the reconfiguration processes to achieve resiliency. Although the developed tool and methods have been designed with consideration for the GKN use case, they can be generalised to any kind of plant layout.

Table of Contents

1. List of Figures

2. List of Tables

1. Introduction

1.1. Structure of the Report

This report contains the outcome of Task 4.4, titled “Guidelines and recommendations for reconfigurability mechanism for machinery and robots”. There are four 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 reconfiguration tools and methods developed in this task using genetic algorithm and multi-agent systems are presented in Section 3 and 4 respectively.

1.2. Problem Definition

In today’s manufacturing environment, changes are inevitable and every agile manufacturing system must be able to respond quickly to such dynamic and unpredictable changes without significant impact on production efficiency, quality and overall costs. Achieving manufacturing agility is very crucial in highly competitive and customer-driven markets, where minor deterioration in production performance and product delivery may affect the manufacturers reputation and long-term survival. The class of systems that are designed at the outset for rapid adaptability in response to market or intrinsic system changes is known as the reconfigurable manufacturing systems (RMS).

The aim of RMS is to achieve rapid modification and quick integration of new functions into existing systems using basic process modules [1]. One of the key enabling characteristics of a reconfigurable system is modularity. In RMS, modular blocks are used to achieve the required system functionality to produce a part family. A distinguishing feature of an RMS is that its configuration evolves over time to provide the specific functionalities and capacity needed for every demand period. Customised flexibility is provided through scalability and reconfiguration as and when needed to meet market requirements instead of the fixed general flexibility provided by flexible manufacturing systems (FMS) [2]. An RMS can be easily reconfigured at system, cell machine or controller levels. At system level and cell levels, the layout of the plant can be changed completely by adding or removing machines from the plant. At machine or controller levels, the structure and process capabilities of a machine may be adjusted by simply removing, adding or changing the constituent hardware and software modules of the machine (e.g. spindles and axes, tool magazines, controllers) [3].

In RMS, parts are grouped into families, and a certain mix and volume of parts within a family corresponds to a specific configuration of the RMS. By designing RMSs around part families, the manufacturing system can respond to changes in families in a timely and cost-effective fashion. The fundamental idea behind part families is to group components that require similar operations into the same family, so they can be manufactured in a fixed system configuration. The configuration of an RMS is fixed during a demand period and is reconfigured at the end of the demand period based on changes in the market demand and the capacity management policy adopted by the enterprise [4]. That is, the system is initially configured to manufacture parts in a given family within a certain time horizon. Once it is finished, the system is reconfigured to manufacture parts from the next family, and so forth. For every change in configuration, changeover costs are incurred, and certain length of production time could be loss depending on how long it takes to complete the changeover operation.

The effectiveness of an RMS depends on the optimal grouping of parts into families. Several techniques have been designed for grouping parts into families and to determine the corresponding systems' configurations [5, 6, 7]. The key factors that are generally considered in part grouping are changeover costs, costs per unit time of idle and/or underutilised resources [5]. In make-to-order environments, the sequential approach to the manufacturing of parts in different families may not be efficient over multiple generations of small batch sizes (short cycle time) due to increased idle time and changeover costs. Higher efficiencies can be gained through concurrent manufacturing of multiple families. However, the configuration of the resources at every instant of time could either impede or facilitate the system's operability, productivity and responsiveness. Concurrent manufacturing of parts from multiple families creates the problem of when and which resources should be reconfigured to guarantee schedule feasibility and improve the performance of the system with respect to chosen performance measures.

The reconfiguration mechanism developed in Task 4.2 is an extension of the planning and scheduling mechanism developed in Task 4.2 [8] for a combination layout (functional + cellular) production system. The reconfiguration tool combines genetic algorithm (GA) optimisation with distributed multi-agent systems (MAS) to achieve optimal reconfiguration plan and resilient changeover process. GA is used to generate a system-level operations execution and reconfiguration schedule, which defines the changeover actions that should be carried out and the time of execution. Distributed multi-agents representing robots and process modules are then used to direct the actual execution of the reconfiguration plan at cell and/or machine level. One of the key issues to be addressed in this task is how to define the thresholds and nervousness control parameters for balancing the frequency of reconfiguration.

1.3. Aims and Objectives

The aim of Task 4.4 is to develop mechanisms for system level and machine level reconfiguration. The objectives towards the achievement of this aim are:

- To define a structure for linking reconfiguration planning and scheduling layers of production systems to higher and lower level layers
- To develop a strategy for optimal assignment of production tasks to resources and reconfiguration planning to meet the needs of coevolving part families in a reconfigurable manufacturing system
- To define thresholds for triggering reconfiguration, which will guarantee that reconfiguration occurs only when needed and not make the system too nervous. The key issues to be addressed are to determine the resources to be reconfigured and when the reconfigurations should take place
- To develop a distributed agent-based architecture control architecture, which uses negotiations among agents to achieve seamless and dynamic reconfiguration (quick changeover) at cell and/or machine level.

2. Requirements Specification and System Analysis

2.1. The PERFoRM System Architecture

The PERFoRM system architecture for seamless production system reconfiguration is based on a network of distributed hardware devices and software applications [9]. This architecture addresses different ISA-95 levels, exposing their functionalities as services and are interconnected in a transparent manner by using an industrial middleware (Figure 1). The middleware is a distributed, configurable and extendable service-based integration platform that aims to guarantee a transparent, secure and reliable interconnection of the heterogeneous hardware devices and software applications developed in the PERFoRM ecosystem. An important innovation of this platform is its distributed nature instead of the centralized ones that are found nowadays and can act as a single point of failure as well as a limitation for the system scalability.

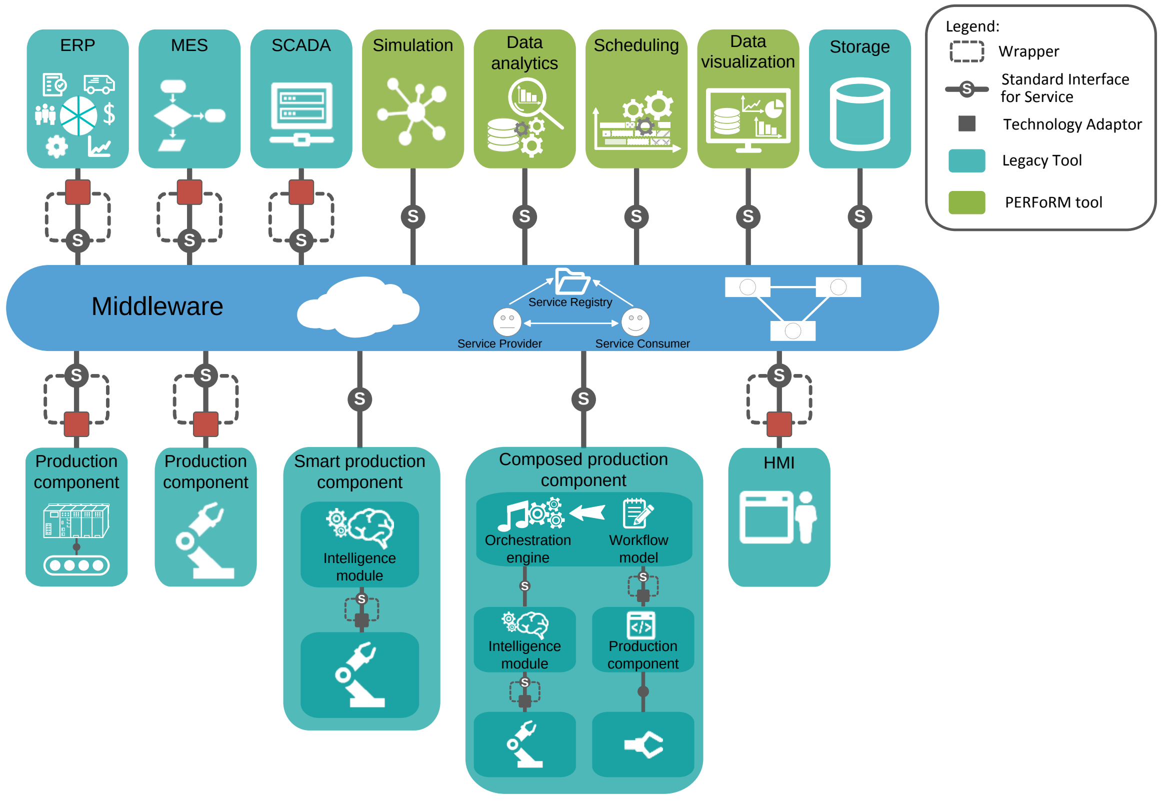


Figure 1 Overall PERFoRM System Architecture

The PERFoRM architecture uses standard integration interfaces as the main drivers for pluggability and interoperability, to enable seamless and transparent connection between devices and software applications. Manufacturing companies currently use legacy and heterogeneous systems for the management and the execution of their production process. The innovative architecture proposed in the PERFoRM project can gain wide-spread adoption in the industry only if it is possible to integrate with legacy systems. For this reason, technology adapters (Figure 1) are key elements to connect legacy systems to the PERFoRM middleware and to transform the legacy data model into the standard interface data model defined in Task 2.3 [10]. In this way, the technological adapters are only necessary when there is the need to connect a legacy component (e.g. an existing database or robot) to the PERFoRM system.

In addition to the standard interfaces, a standardized manufacturing data model is adopted in PERFoRM. The data model covers the semantic needs associated to each manufacturing entity. In this context, machinery level and the backbone level abstraction layers are considered. The machinery level covers mainly L1 (automation control) and L2 (supervisory control) layers. The data backbone level covers L3 (manufacturing operations management) and L4 (business planning and logistics).

At a lower level, robots and automation machinery need to be empowered with intelligence and higher processing capabilities to run more complex algorithms, allowing them to process higher amount of data to support seamless reconfiguration of the system and the achievement of self-* properties (e.g. self-adaptation, self-diagnosis). The amount of data being generated in shop floor plants is increasing at a very high rate. The proper analysis, locally (at the edge) and globally (at the cloud), of the collected data assumes a crucial aspect, generating new knowledge that can be used to detect trends, deviations and possible problematic situations in a timely manner.

2.2. Hybrid GA and Agent-Based Reconfiguration

Reconfiguration planning in concurrent manufacturing of parts from multiple families is an NP-hard combinatorial optimisation problem. GA is effective in solving such problems especially when the amount of input data is large, and the rate of exponential data growth for discrete optimisation grows linearly [11]. Although GA cannot guarantee a globally optimum solution in polynomial time, it is able to provide near optimal solutions within a shorter space time compared to other deterministic algorithms for search space optimisation. One of the characteristics of RMS is convertibility, which is the ability to quickly adjust the functionality of the system and controls to suit new production requirements. However, decisions regarding how to deal with exceptions in the reconfiguration process are complex. Multi-agent systems have shown great potentials in dealing with dynamic situations. In this task, agents are used to manage the reconfiguration schedules produced by the optimiser because complex decisions can be made through the synergy resulting from the reasoning and negotiation mechanisms of the agent (Figure 2).

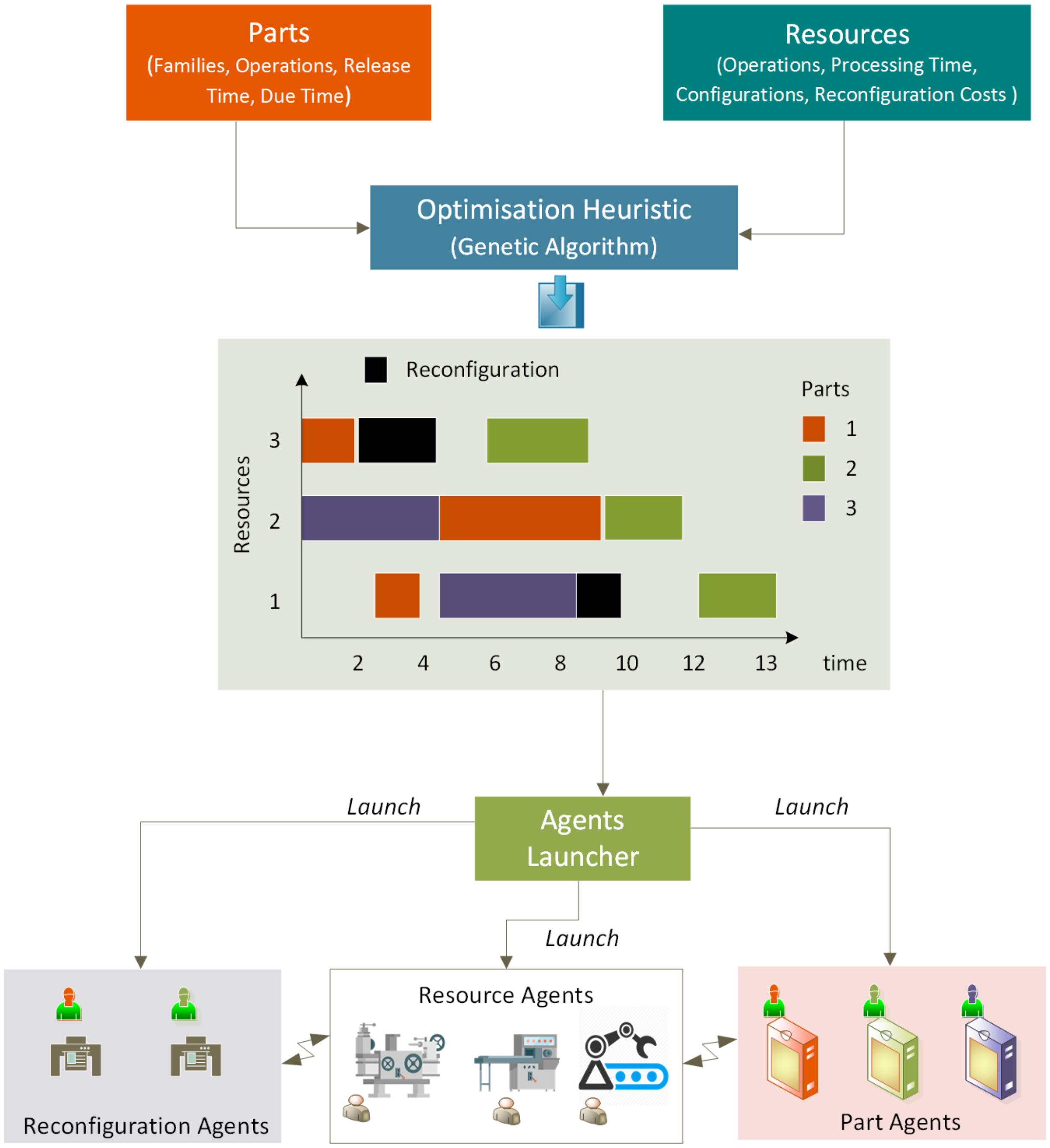


Figure 2 Hybrid centralised and agent-based reconfiguration planning and execution system

2.3. Framework for Linking Enterprise Business Logic and Control Layers to Operations Management Layer

The manufacturing industry is now flooded with innovative ideas and concepts (Industry 4.0, Industrial Internet of Things, cyber-physical systems), which have brought new challenges and opportunities across the industry. ISA-95 framework defines a series of interconnecting functional modules, each with a specific role and clearly defined interfaces for inter-module communication. However, big data and IIoT are poised to render the traditional ISA-95 framework for linking enterprise business applications, operations management and process control layers of manufacturing systems obsolete. Therefore, the traditional manufacturing execution |systems (MES) are evolving to manufacturing operations management (MOM) technologies with built-in intelligent decision-making capabilities. MOMs are designed to address planning and scheduling problems, execution and control, performance analysis and operational intelligence.

One of the limitations of the current MOMs is that they are designed for centralised planning and scheduling systems. It is generally known that centralised planners and schedulers are 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. 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 [12, 13, 14].

Although it has been proven that agent-based schedulers provide better overall performance in environments that require reconfigurability, flexibility and reliability, the technology has not gained much industrial adoption and deployment. This is due in part to the lack of support for distributed systems in current MESs and MOMs. In order to facilitate the adoption of agent-based technology, a new structure for linking the reconfiguration planning and scheduling layers of production systems (MOM or MES) to higher and lower level layers needs to be defined. The proposed framework for linking MOMs to the enterprise business logic and control layers is shown in Figure 3.

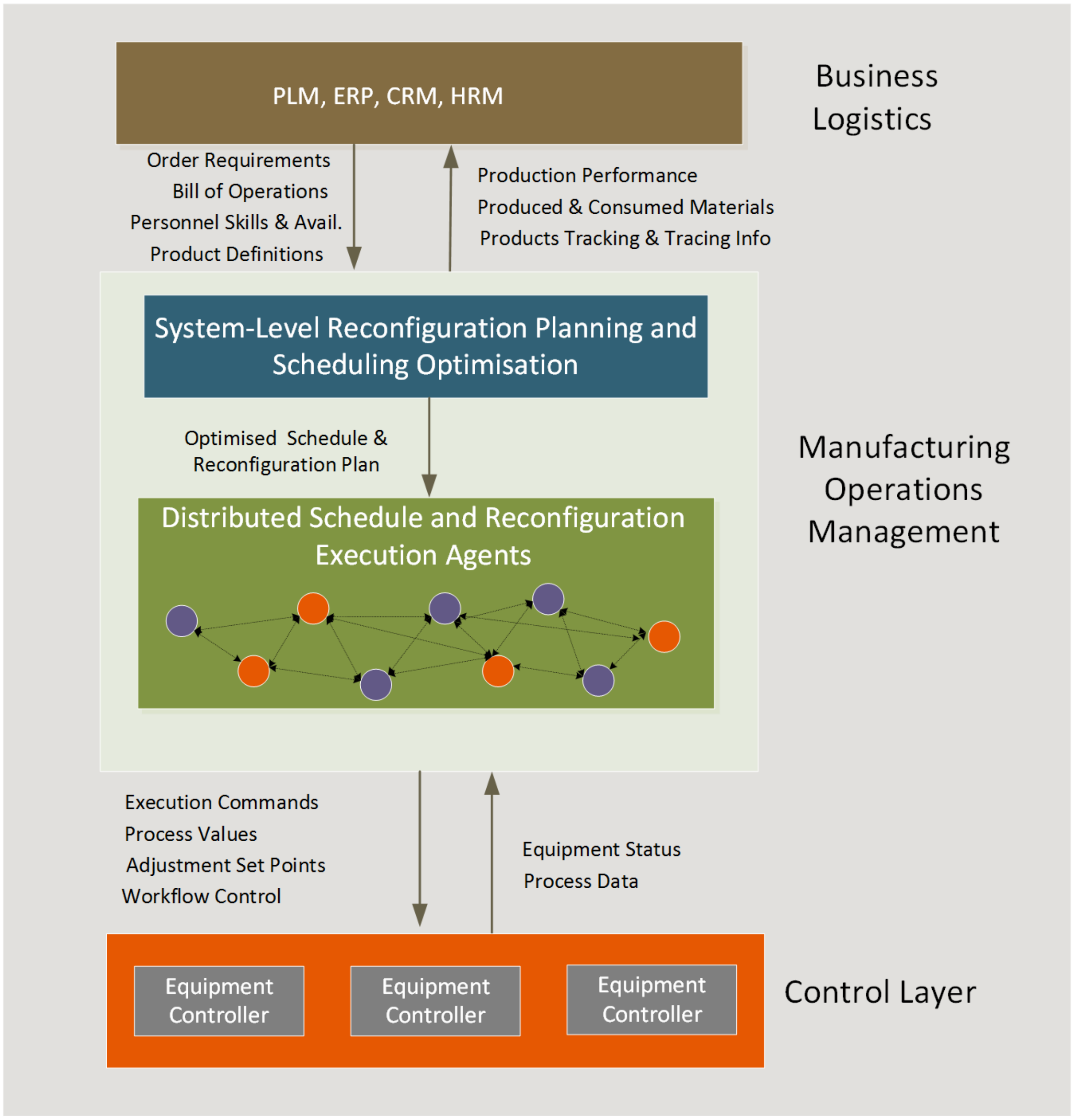

PLM, ERP, CRM, HRM
Business Logistics
Order Requirements
Bill of Operations
Personnel Skills & Avail.
Product Definitions
Production Performance
Produced & Consumed Materials
Products Tracking & Tracing Info
System-Level Reconfiguration Planning and Scheduling Optimisation
Optimised Schedule & Reconfiguration Plan
Distributed Schedule and Reconfiguration Execution Agents
Manufacturing Operations Management
Execution Commands
Process Values
Adjustment Set Points
Workflow Control
Equipment Status
Process Data
Equipment Controller
Equipment Controller
Equipment Controller
Control Layer

Figure 3 Framework for linking MOM to business logic and control layer

2.4. PERFoRMML Data Model Extension

A standardised data model for the PERFoRM project has already been created in WP2. However, the model is not adequate for the requirements of Task 4.4. The following data model is an extension to the default PERFoRMML. The class diagram for the model is shown in Figure 4. Data is supplied to the reconfiguration service through the middleware (Figure 5).

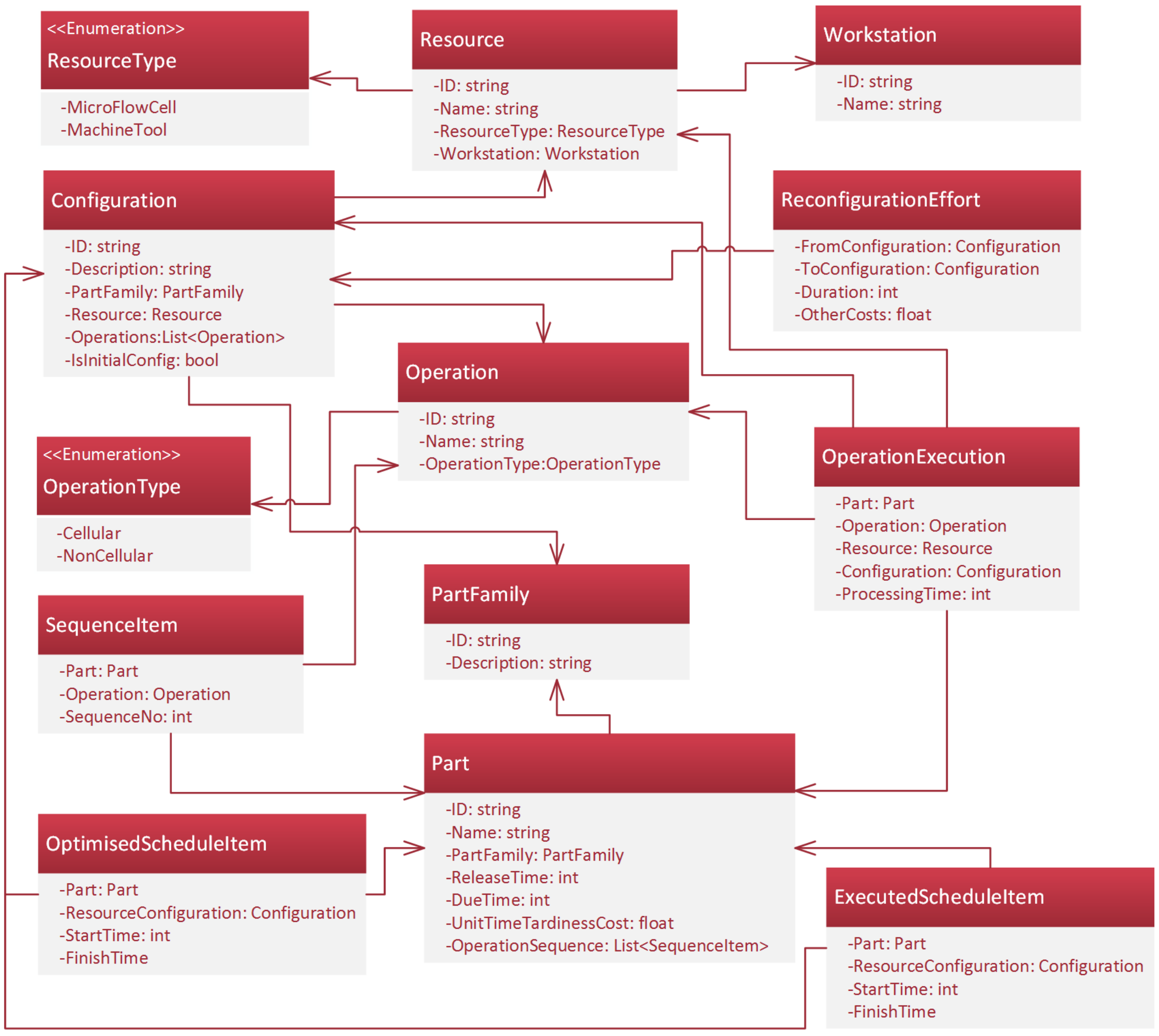


Figure 4 PERFoRMML Data model extension for Task 4.4

Operation: The *Operation* class is a representation of a manufacturing operation that can be performed in the shop floor

- 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 of an actual machine or equipment located in a workstation

- 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

PartFamily: The *PartFamily* class is a representation of a part family

- ID: A unique identifier for a part family
- Description: Meaningful description of the part family

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
- PartFamily: The associated family the part belongs to
- ReleaseTime: The time the part enters the system
- DueTime: The time the part is due
- UnitTardinessCosts: The unit time lateness costs of the part
- OperationSequence: The ordered list of operations for producing the part

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

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

Configuration: The *Configuration* class is a representation of a possible configuration of a resource

- ID: A unique identifier for a configuration
- Description: Meaningful description of the configuration
- PartFamily: The associated part family for the configuration
- Resource: The associate resource for the configuration
- Operations: The list of operations that the associated resource can perform when it is in the configuration

ReconfigurationEffort: The *ReconfigurationEffort* class defines the time and costs of changing a resource from one configuration to the other.

- FromConfiguration: The current configuration of a resource
- ToConfiguration: The target configuration of a resource
- Duration: The time it takes from start to finish of the configuration process
- OtherCosts: Other costs incurred during reconfiguration.

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
- Configuration: The associated configuration of the resource
- ProcessingTime: The time for processing the part on the resource in the specified configuration

ReconfiurationItem: The *ReconfiurationItem* class is a representation of a reconfiguration item generated by the optimiser

- Resource: The associated resource

- FromConfiguration: The associated current configuration
- ToConfiguration: The associated target configuration
- StartTime: The start time of the reconfiguration process
- FinishTime: The finish time of the reconfiguration process

ScheduleItem: The *ScheduleItem* class is a representation of a schedule item generated by the 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

The data required for reconfiguration is provided by the middleware as shown in Figure 5. The reconfiguration plan generated by the planning tool is used by the agents to direct reconfiguration at the micro-flow cell and machine levels.

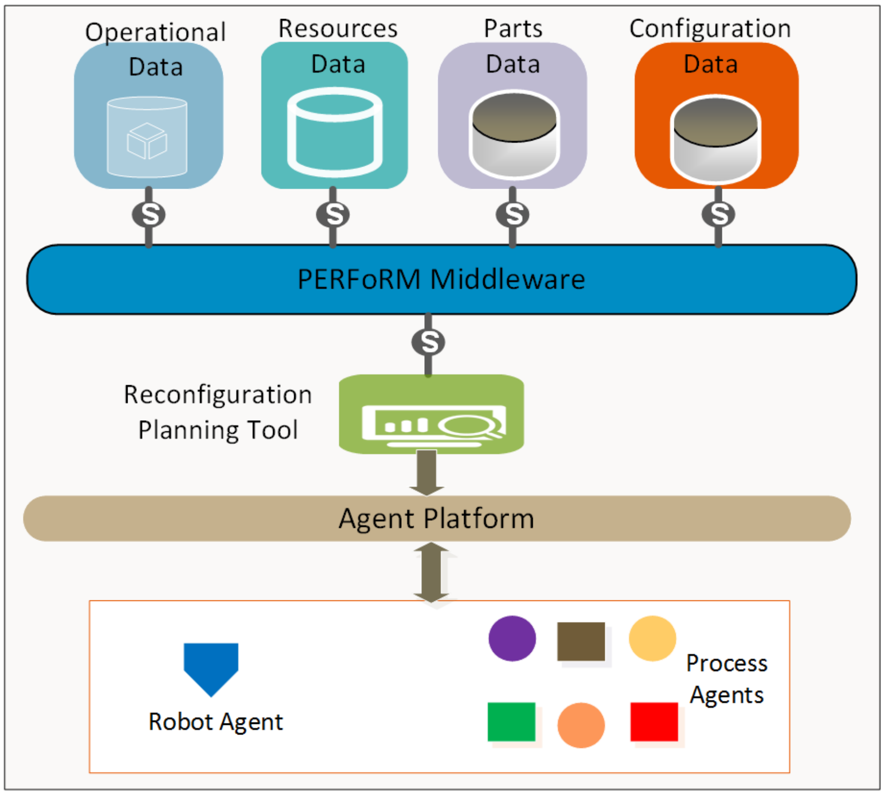


Figure 5 Data consumption through the middleware

2.5. Plant Layout

Three categories of manufacturing flexibilities are identified in T4.2 namely *routing*, *process* and *machine* flexibilities. However, the machine flexibility, which is the ability of a machine to perform a variety of operations, was not addressed in Task 4.2 [8]. It was assumed that once machines have been configured for a set of part families, they cannot be changed during a schedule execution. In addition to the process and routing flexibilities, the reconfigurability of the machines is also considered in Task 4.4.

The key characteristics of an RMS are modularity, integrability, customised flexibility, scalability, convertibility and diagnosability. These characteristics have been considered in the design of the micro-flow cell concept presented by the GKN use case. The micro-flow cell (Figure 6) consists of a Programmable Logic Controller (PLC) for communication and control, a Human Machine Interface (HMI), a robot system for part handling and a number of process modules. Each process module is controlled by a PLC. The arrangement of resources in the plant is shown in Figure 7 and the flows of parts through the resources are shown in Figure 8.

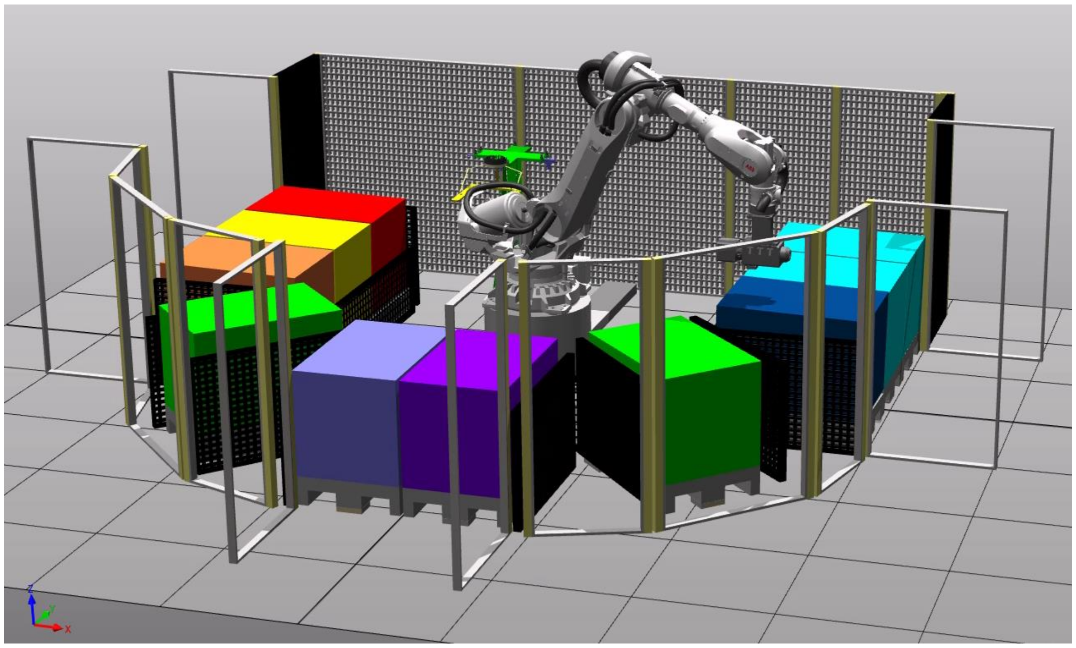


Figure 6 Micro-flow cell

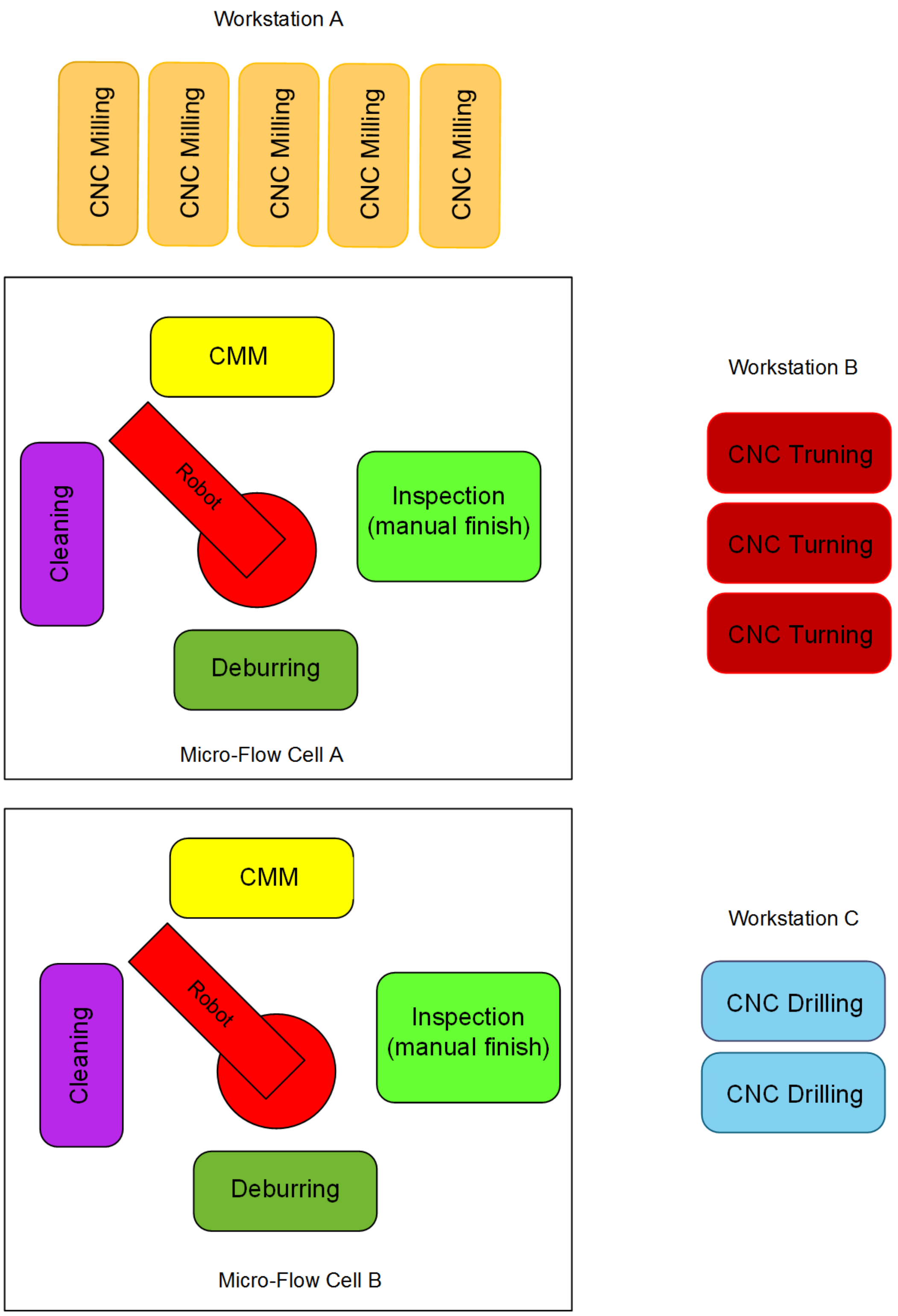


Figure 7 Hybrid micro-flow cell and functional plant layout

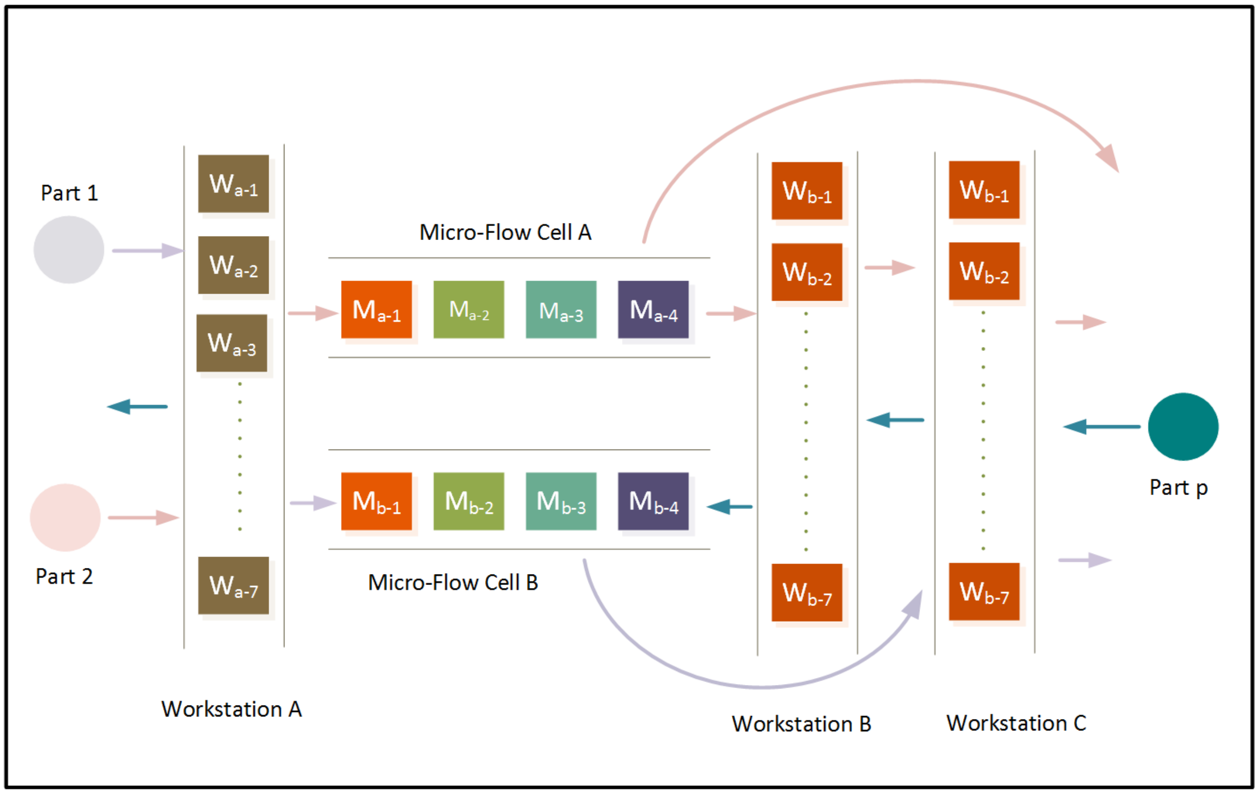


Figure 8 Part flow through the micro-flow cells and workstations

3. Reconfiguration Planning and Scheduling Model

This chapter is focused on the design of the planning and scheduling model for the reconfigurable manufacturing system using genetic algorithm.

3.1. Problem Formulation

The goal is to find the best production and reconfiguration schedules that will minimise weighted sum of reconfiguration and tardiness costs. The threshold level for triggering reconfiguration and nervousness control is indirectly defined using the cost of changing a resource from one configuration to the other. A low reconfiguration cost compared to the tardiness cost will make the system too nervous and vice-versa.

3.1.1.Model Inputs

The following inputs are required by the reconfiguration planning and scheduling model:

Plant Data

R:	a set of resources in the plant
r:	number of resources in the resources set
P:	a set of parts to be manufactured
p:	number of parts in the parts set
O:	a set of operations that can be performed in the plant
F:	a set of all possible part family grouping

Resource Data

i:	index of resource, $i = 1 \dots\dots\dots r$
C:	the set of possible configurations of resource *i*
m:	the index of the m^{th} configuration
OC_m:	the set of operations in the m^{th} configuration $OC_m \subseteq O$
t_{mn}:	the reconfiguration time from m^{th} to n^{th} configuration
c_{mn}:	the reconfiguration cost from m^{th} to n^{th} configuration

Parts Data

j:	index of part, $j = 1 \ldots\ldots p$
f_j:	the family that part j belongs to, $f_j \in F$
r_j:	release time of part j
d_j:	due time of part j
c_j:	unit time tardiness cost of part j
O_j:	ordered set of operations for manufacturing part j, $O_j \subseteq O$
o_{kj}:	the k^{th} operation of part j, $o_{kj} \in O_j$

Operation Execution Data

t_{kjim}: the processing time of the k^{th} operation of the j^{th} part on the i^{th} resource in m^{th} configuration

3.1.2.Model Outputs

The model produces the following outputs:

Operation Schedule

The schedule is a list of operations to be performed on a specific resource, at a specific time and in a specific configuration. Each item in the list has the following parameters:

P_j:	part
R_i:	resource
C_m:	configuration
t_1:	start time of the operation
t_2:	finish time of the operation

Reconfiguration Schedule

The reconfiguration schedule is a list of reconfigurations that are required for the feasibility of the operation schedule.

R_i:	resource
C_1:	from configuration

C_2: target configuration
t_1: start time of the reconfiguration
t_2: finish time of the reconfiguration

3.1.3. Assumptions

The following assumptions are mode in the model:

- All resources (micro-flow cells and machine tools) are reconfigurable
- All resources are available at any time
- All parts are released at time $r_j = 0$
- Parts visit micro-flow cells only once (Figure 8).
- A resource can only execute one operation 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_{kj} must be completed before operation $o_{(k+1)j}$
- No operation preemption. i.e. an operation cannot be interrupted on a machine once started until it is completed
- The system is not affected by stochastic events such as machine breakdowns (100% availability), dynamic introduction of new parts, order cancellation, changes in parts priorities, changes in parts mix and reworks due to quality issues
- There are no ramp-up and ramp-down periods during reconfiguration (Figure 9)

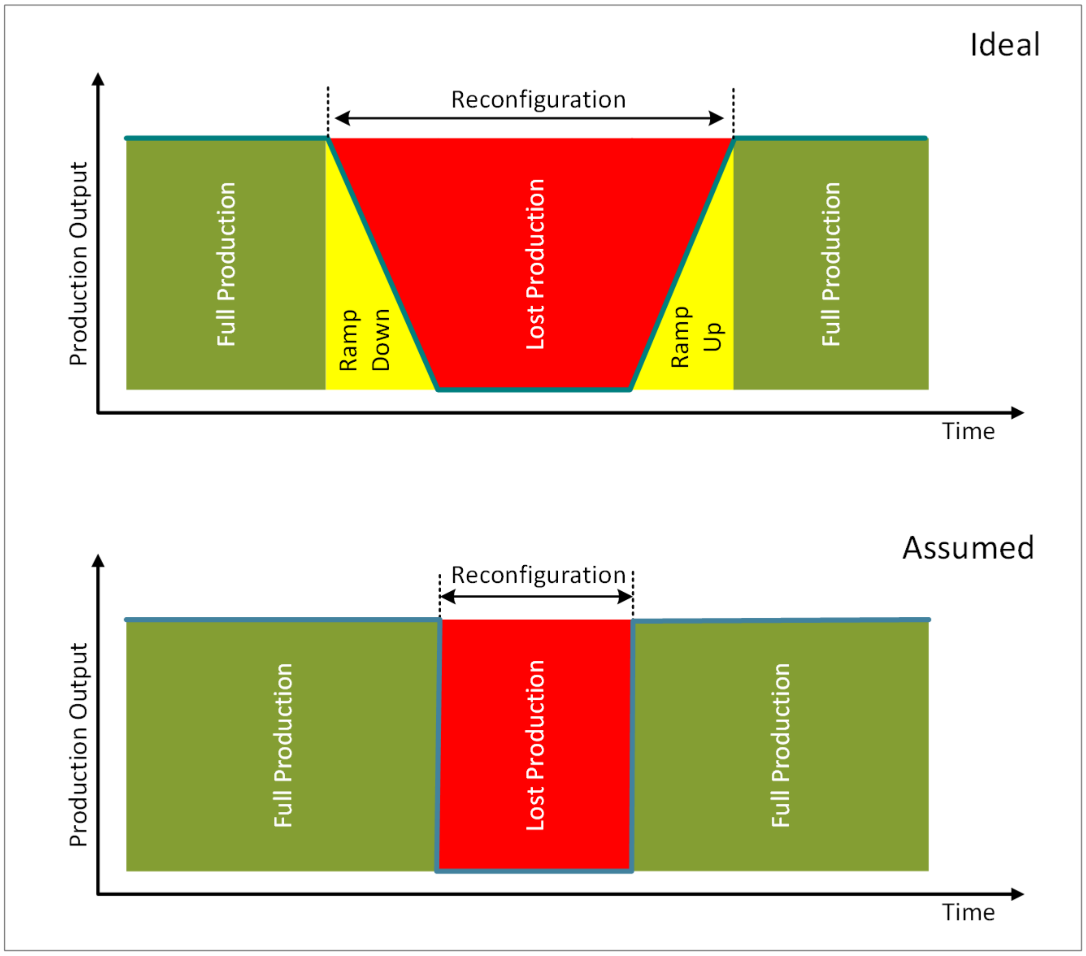


Figure 9 Ideal versus assumed reconfiguration time

3.2. Genetic Algorithm Model

Genetic algorithm is a heuristic search method used 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 model adopted in this task is based on the algorithm proposed by Du & Xiong [15] for a flexible job-shop scheduling problem. The algorithm has been modified to account for the hybrid layout scenario described in Section 2. The model is described using the information shown in Table 1.

Table 1 Part operations and resource configurations data

Part	Operation	Resources													
		R_1		***R_2***		***R_3***		***R_4***		***R_5***		***R_6***		***R_7***	
		C_{11}	C_{12}	C_{21}	C_{22}	C_{31}	C_{32}	C_{41}	C_{42}	C_{51}	C_{52}	C_{61}	C_{62}	C_{71}	C_{72}
1	O_{11}	-	-	-	-	1	-	1	-	-	-	-	-	-	-
	O_{12}	1	-	1	-	-	-	-	-	-	-	-	-	-	-
	O_{13}	1	-	1	-	-	-	-	-	-	-	-	-	-	-
2	O_{21}	-	-	-	-	-	-	-	-	-	1	-	1	-	-
	O_{22}	-	1	-	1	-	-	-	-	-	-	-	-	-	
	O_{23}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
	O_{24}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
3	O_{31}	-	-	-	-	-	-	-	-	-	1	-	1	-	-
	O_{32}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
	O_{33}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
	O_{34}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
4	O_{41}	-	-	-	-	1	-	1	-	-	-	-	-	-	-
	O_{42}	1	-	1	-	-	-	-	-	-	-	-	-	-	-
	O_{43}	1	-	1	-	-	-	-	-	-	-	-	-	-	-
	O_{44}	-	-	-	-	-	-	-	-	1	-	1	-	-	-
5	O_{51}	-	-	-	-	-	-	-	-	-	-	-	-	-	1
	O_{52}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
	O_{53}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
	O_{54}	-	1	-	1	-	-	-	-	-	-	-	-	-	-
	O_{55}	-	-	-	-	-	1	-	1	-	-	-	-	-	-
6	O_{61}	-	-	-	-	-	-	-	-	-	-	-	-	1	-
	O_{62}	1	-	1	-	-	-	-	-	-	-	-	-	-	-
	O_{63}	1	-	1	-	-	-	-	-	-	-	-	-	-	-

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

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

A second matrix called the *operation-resource matrix*, M_{or}, is also derived. It represents the constraint model of relation between part operations and resources. The resources and operations are arranged along the rows and columns of the matrix respectively. If the element $M_{or}(i, k) = 1$, then the operation in the k^{th} column can be processed by the resource in the i^{th} row.

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

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 resource-configuration column indicates that the corresponding part operation can be performed by the resource when it is in the corresponding configuration. From Table 1, there are seven parts, grouped into two families. Each resource can be configured to process parts from the two families (C_{R1} and C_{R2}).

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 and reconfiguration problem. The chromosome representation has two components; *operation sequence* component (*OS*) and the *resource-configuration selection* component (*RS*). The *OS* component values represent the sequence of operation while the *RS* component values represent the resource that is selected to process the corresponding operation and its configuration.

A valid chromosome for the problem in Table 1 is shown in Table 2. According to Table 1, there are 23 total operations, so the length of the chromosome is 23. The genes of the *OS* component are filled with random numbers generated from 1 to 23. The operations are scheduled in the order of their values. The *RS* component values are obtained by choosing a resource configuration from the set of available resource configurations for an operation during population initialisation. For instance, C_{11}, which is the part family 1 configuration of resource R_1, has been selected to process operation O_{44}. C_{21}, which corresponds to the second configuration of resource R_1 could also be chosen.

Table 2 Chromosome representation for operation execution and resource configuration

	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{24}	O_{31}	O_{32}	O_{33}	O_{34}	O_{41}	O_{42}	O_{43}	O_{44}
OS	11	2	20	8	3	19	10	15	7	14	1	18	17	12	9
MS	C_{31}	C_{21}	C_{21}	C_{62}	C_{12}	C_{12}	C_{12}	C_{52}	C_{22}	C_{22}	C_{22}	C_{41}	C_{11}	C_{11}	C_{61}

	O_{51}	O_{52}	O_{53}	O_{54}	O_{55}	O_{61}	O_{62}	O_{63}
OS	21	13	6	5	22	23	16	4
MS	C_{72}	C_{22}	C_{22}	C_{22}	C_{42}	C_{71}	C_{11}	C_{11}

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: Create and initialise the M_{po} matrix

Step 2: 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_{34}, which belongs to part 3

Step 3: 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 3 of matrix M_{po} with values equal to 1 are O_{31}, O_{32}, O_{33} and O_{34}. The first operation is O_{31}, so it is selected

Step 4: 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 5: Get the resource the operation has been assigned to and its configuration. For instance, operation O_{31} is assigned to resource 5 in configuration 2. If the most recent configuration of the resource is different from the currently required configuration (i.e. configuration 2), then a reconfiguration is required. Update the available time of the resource and the total reconfiguration costs accordingly.

Step 6: Schedule the operation and update the resource available time and part ready time

Step 7: Repeat step 2 to 6 for the other operations in the chromosome.

A schedule generated by the above procedure is guaranteed to be feasible, so no repair mechanism is required for the chromosomes. The sequence of operations and reconfigurations after decoding the chromosome in Table 2 are given as follows:

<u>Sequence of Operation</u>

$O_{31} – O_{11} – O_{21} – O_{61} – O_{51} – O_{52} – O_{32} – O_{22} – O_{41} – O_{23} – O_{12} – O_{42} – O_{53} – O_{33} – O_{34} – O_{62} – O_{43} – O_{44} – O_{24} – O_{13} – O_{54} – O_{55} – O_{63}$

<u>Reconfigurations</u>

$C_{71} – C_{72}, C_{22} – C_{21}, C_{12} – C_{11}, C_{21} – C_{22}, C_{62} – C_{61}, C_{11} – C_{12}, C_{22} – C_{21}, C_{21} – C_{22}, C_{41} – C_{42}, C_{12} – C_{11}$

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 *RS* components of individuals are done in stages. The steps for creating each individual in the initial population are as follows:

Step 1: Create an operation-machine matrix, M_{or}, and initialise based on the available resource sets for each operation

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

Step 3: Randomly fill the operation sequence component of the chromosome with values from 1 to 23 without any repetition

Step 4: For each gene in the chromosome, choose the resource with the shortest availability out of the set of resources that can perform the operation represented by the gene. If there is a tie, randomly select one

Step 5: Add the processing time of the operation on the resource to the available time of the resource

Step 6: If the operation is in a micro flow cell, then reduce the available resources for performing the remaining cellular operations of the part to the chosen micro-flow cell by setting the values of the other resources in the same column to 0 in the M_{or} matrix.

Step 7: Repeat step 4 to 6 until all operations in the chromosome have been assigned.

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 reconfiguration costs for the schedule produced by the chromosome. Given a set of reconfigurations with associated costs

$$\mathrm{Z} = \{z_1, z_2 \ldots\ldots\ldots\ldots z_y\} \tag{1}$$

$$f_c = \alpha T_c + \beta R_c \tag{2}$$

$$T_c = \sum_{j=1}^{p} max(0, C_i - D_i) \times U_i \tag{3}$$

$$R_c = \sum_{a=1}^{|Z|} z_a \tag{4}$$

where

α is the weight of total tardiness cost T_c

β is the weight of total reconfiguration cost R_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 elite selection method.

Crossover Operator:

The choice of crossover operator is very important in genetic algorithm. Crossover operators are application and chromosome dependent. The crossover operator is applied to the operation sequence components of a chromosome only, while the assignment of resources 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

	O_{11}	**O_{12}**	**O_{13}**	**O_{21}**	**O_{22}**	**O_{23}**	**O_{24}**	**O_{31}**	**O_{32}**	**O_{33}**	**O_{34}**	**O_{41}**	**O_{42}**	**O_{43}**	**O_{44}**
OS	6	12	7	13	8	3	15	11	4	10	5	2	9	1	14
MS	C_{31}	C_{21}	C_{21}	C_{62}	C_{12}	C_{12}	C_{12}	C_{52}	C_{22}	C_{22}	C_{22}	C_{41}	C_{11}	C_{11}	C_{61}

Parent 2

	O_{11}	**O_{12}**	**O_{13}**	**O_{21}**	**O_{22}**	**O_{23}**	**O_{24}**	**O_{31}**	**O_{32}**	**O_{33}**	**O_{34}**	**O_{41}**	**O_{42}**	**O_{43}**	**O_{44}**
OS	3	9	1	13	4	7	12	14	11	10	6	15	8	2	5
MS	C_{41}	C_{11}	C_{11}	C_{52}	C_{22}	C_{22}	C_{22}	C_{62}	C_{12}	C_{12}	C_{12}	C_{41}	C_{21}	C_{21}	C_{61}

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 resource selection chromosome of the first parent into the first offspring

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

Parent 1

	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{24}	O_{31}	O_{32}	O_{33}	O_{34}	O_{41}	O_{42}	O_{43}	O_{44}
OS	6	12	7	13	8	3	15	11	4	10	5	2	9	1	14
MS	C_{31}	C_{21}	C_{21}	C_{62}	C_{12}	C_{12}	C_{12}	C_{52}	C_{22}	C_{22}	C_{22}	C_{41}	C_{11}	C_{11}	C_{61}

Parent 2

	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{24}	O_{31}	O_{32}	O_{33}	O_{34}	O_{41}	O_{42}	O_{43}	O_{44}
OS	3	9	1	13	4	7	12	14	11	10	6	15	8	2	5
MS	C_{41}	C_{11}	C_{11}	C_{52}	C_{22}	C_{22}	C_{22}	C_{62}	C_{12}	C_{12}	C_{12}	C_{41}	C_{21}	C_{21}	C_{61}

Offspring 1

	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{24}	O_{31}	O_{32}	O_{33}	O_{34}	O_{41}	O_{42}	O_{43}	O_{44}
OS	1	13	7	12	14	3	15	11	4	10	6	8	2	5	9
MS	C_{31}	C_{21}	C_{21}	C_{62}	C_{12}	C_{12}	C_{12}	C_{52}	C_{22}	C_{22}	C_{22}	C_{41}	C_{11}	C_{11}	C_{61}

Offspring 2

	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{24}	O_{31}	O_{32}	O_{33}	O_{34}	O_{41}	O_{42}	O_{43}	O_{44}
OS	13	8	3	15	4	7	12	14	11	10	5	2	9	1	6
MS	C_{41}	C_{11}	C_{11}	C_{52}	C_{22}	C_{22}	C_{22}	C_{62}	C_{12}	C_{12}	C_{12}	C_{41}	C_{21}	C_{21}	C_{61}

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. Mutation is applied to the operation sequence and resource selection components of chromosomes as described in the following steps: -

Step 1: Choose two low probability threshold values for mutating the operation sequence and resource selection components 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 resource selection chromosome mutation probability, set the resource allele in that position to another randomly selected resource from the available resources lists for that operation

Step 2b: If the randomly generated probability value is less than the operation sequence chromosome mutation probability, then swap the operation sequence value in the position with that of another randomly selected position. This should only be done for operations that are not performed in a micro-flow cell. Otherwise, the whole part will have to be swapped.

4. Agent-Based Reconfiguration Mechanism

The agent-based reconfiguration focuses on the logical re-organization of the reconfigurable micro-flow production cells. This chapter detail the architecture as well as the description of the reconfiguration mechanism through the dynamic and automatic plug-in and plug-out of the modular processes. The reconfiguration is triggered according to the schedule received from the planning and scheduling logic. The process modules are plugged and unplugged to fulfil the work orders defined in the schedule. When an unpredictable event occurs, decisions are made by the agents in real-time to establish corrective mechanisms for maintaining the stability and feasibility of the pre-generated schedule.

4.1. Multi-Agent System (MAS) to Support Reconfiguration

This tool is built upon MAS principles, considering a society of distributed and autonomous agents to regulate the re-organization procedure of the micro-flow cell. The MAS paradigm [16, 17], derived from the distributed artificial intelligence field, is pointed out as a suitable approach to support flexibility, robustness and reconfigurability. The inherent characteristics of MAS can easily support the micro-flow cell requirements, namely in terms of reconfigurability, scalability and pluggability. The use of MAS principles in this tool can bring the following benefits:

- **Scalability**: the addition of new processes in the micro-flow cell is very simple in logical terms, simply requiring the instantiation of developed agent class on the fly (with its proper customization); note that there is no need for additional code from the point of view of the agents, or also the need to stop, re-program and re-start the tool.

- **Distributed data collection**: the MAS approach allows to implement a distributed approach for the collection of data from the process modules and robots supporting the implementation of performance monitoring.

- **Data persistence**: this solution guarantees the persistence of the data in case of failure. The agents store individually and locally the current cell configuration and its own status. In case of breakdown, when the system is turned on, each individual agent accesses to the local database to update the cell configuration and its own status.

- **Cooperation**: the interaction among multiple robots and process modules to exchange information, regarding schedule and to execute their operations after a reconfiguration procedure, is simplified by using MAS. Additionally, in case of multiple micro-flow cells, the creation of a network of interconnected agents increases the seamless reconfiguration, namely by reacting in real time to failures, rescheduling dynamically the system and overcoming problems related to performance bottlenecks.

- **Plug and Produce**: Plug and produce is a concept that allows resources to be quickly added and removed from a manufacturing. The multi-agent system allows 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.

4.2. Agent-based Architecture

The designed MAS-based system is composed by two types of agents, namely the "Robot Agent" representing robot resources and the "Process Agent" representing process modules. The creation of two types of agents increases the modularity and flexibility of the system, since the agents have distinct functions, establishing in this way a similarity between the logical and the physical levels, allowing the agents to have a better representation of the system. The robot agent is linked with the cell configuration and safety procedures, while the process agent only cares with its associated physical process. Aiming to keep the cell configuration updated during the reconfiguration procedure, the process and robot agents need to interact, namely to exchange information regarding the plug-in and plug-out of the process modules. The internal architecture of these agents is illustrated in Figure 10.

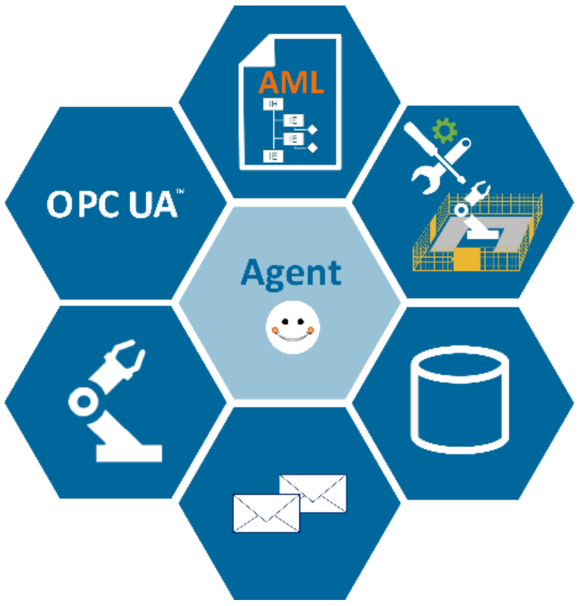


Figure 10 Internal architecture of the agents

Each agent is managing the logical re-organization of its physical device, contributing for the re-organization of the micro-flow-cell. The agents interconnect their physical counterparts, i.e. robot controllers and PLCs, by getting data from OPC-UA (Open Platform Communications - Unified

Architecture) servers, acting as adapters to be PERFoRM compliant. The information collected from the physical resources, regarding current configuration status and performance, is stored in their local databases. The agents reason using the historical and current data to perform the reconfiguration procedure and share information by exchanging messages among themselves according to the FIPA-ACL (Foundation for Intelligent Physical Agents – Agent Communication Language) communication language.

The behaviour of each type of agent is dependent of its role and objectives, being in this work formalized using the Petri nets formalism [18]. Petri nets is a formal modelling tool adequate to model and to analyse the behaviour of complex event-driven systems characterised as being concurrent, asynchronous, stochastic and with high-level distribution. The Petri net behaviour model for the Process agent is illustrated in Figure 11.

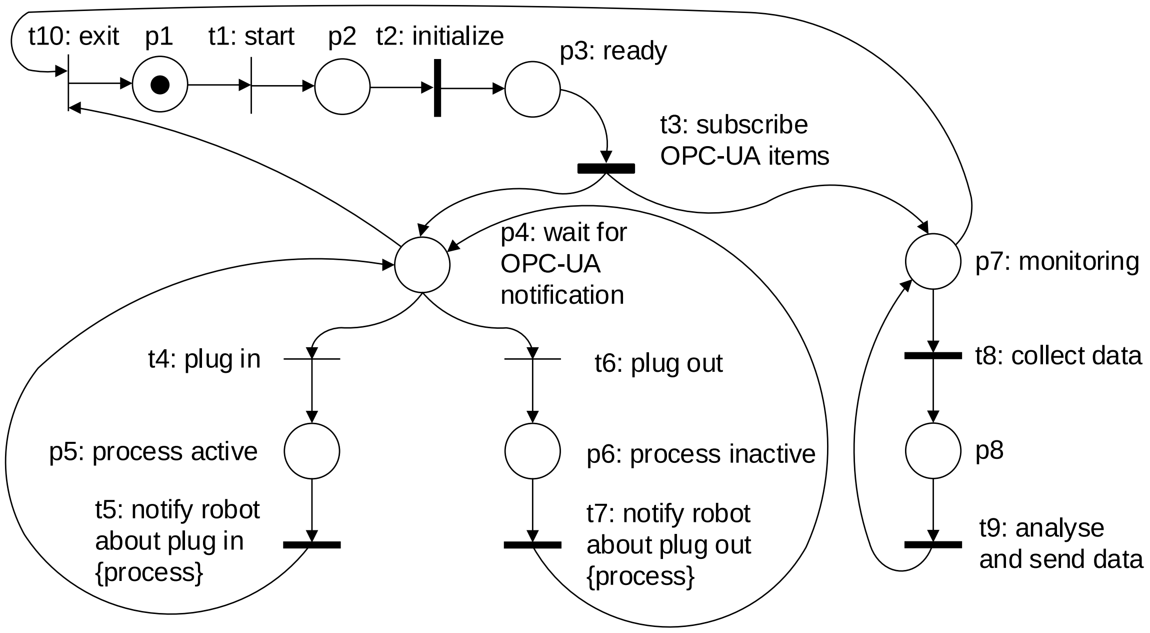

t10: exit
p1
t1: start
p2
t2: initialize
p3: ready
t3: subscribe OPC-UA items
p4: wait for OPC-UA notification
p7: monitoring
t4: plug in
t6: plug out
t8: collect data
p5: process active
p6: process inactive
p8
t5: notify robot about plug in {process}
t7: notify robot about plug out {process}
t9: analyse and send data

Figure 11: Behaviour model for the Process Agent

The process agents are created according to the catalogue of modular process modules that can be used in the micro-flow production cell, being created one process agent for each physical process module.

After the initialization phase, where the agent is parametrized according to the details of the process module it represents, namely its name and OPC-UA server address, and subscribes some process's parameters in the OPC-UA server, the agent enters in a sleep mode, waiting for an event (as result of the initial subscription) notifying that its associated process is plugged-in. At this stage, the agent switches to an active mode, updates its position and location within the micro-flow production cell and notifies the robot agent responsible for this cell about its new state, providing information regarding the process and its position. The same procedure is executed when the process agent receives an event notifying that the process is plugged-out. This mechanism allows the seamless reconfiguration of the cell in a distributed manner, since the process agents are individually detecting when are plugging-in and -out and exchanging information with each other and with the robot agents to maintain the proper knowledge about the cell configuration.

In active mode, each process agent is also continuously collecting data related to the performance of its process module, e.g., processing time, number of processed parts and status (idle or execution). This data is analysed and aggregated and sent to an external Key Performance Indicator (KPI) monitoring tool.

The Petri net behaviour model for the Robot agent is illustrated in Figure 12.

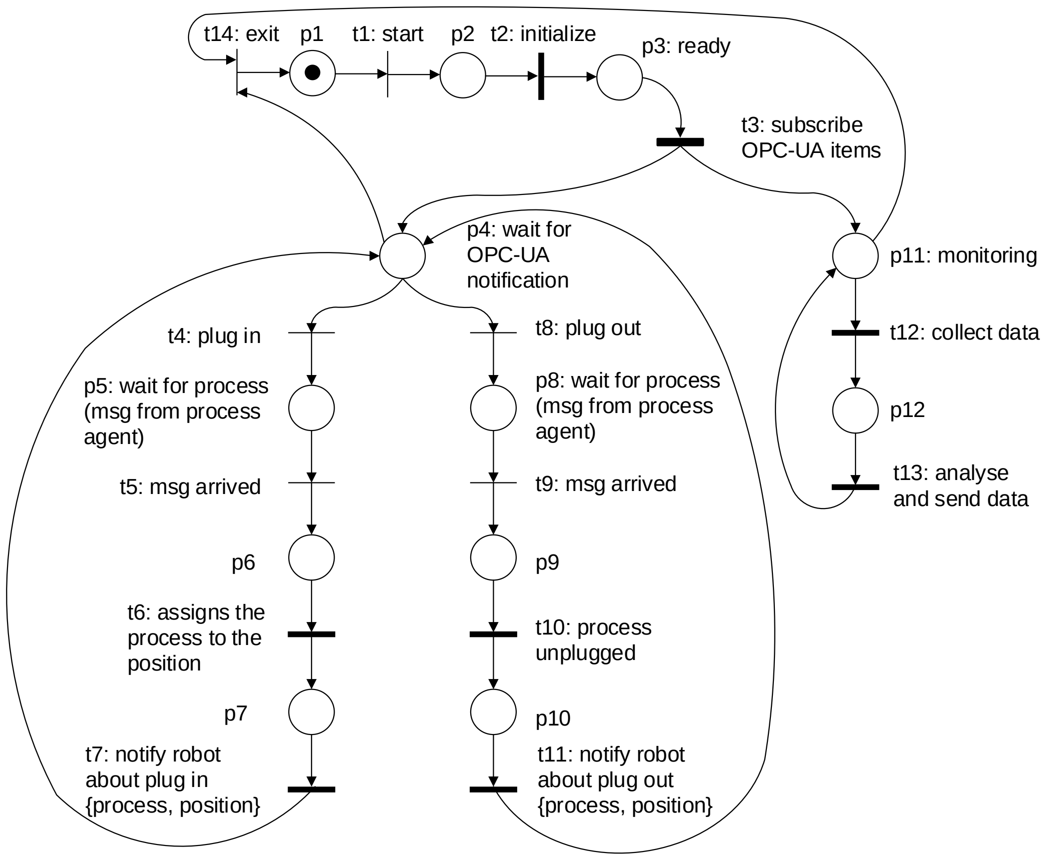

t14: exit
p1
t1: start
p2
t2: initialize
p3: ready
t3: subscribe OPC-UA items
p4: wait for OPC-UA notification
p11: monitoring
t12: collect data
p12
t13: analyse and send data
t4: plug in
p5: wait for process (msg from process agent)
t5: msg arrived
p6
t6: assigns the process to the position
p7
t7: notify robot about plug in {process, position}
t8: plug out
p8: wait for process (msg from process agent)
t9: msg arrived
p9
t10: process unplugged
p10
t11: notify robot about plug out {process, position}

Figure 12: Behaviour model for the Robot Agent

The robot agent is launched at the same time as the process agents, being created one robot agent for each physical robot placed in the micro-flow production cell (usually, a micro-flow cell has only one robot and only one robot agent is created, but if it contains more robots, additional robot agents are created). After the initialization phase, similar to the one described for the process agent, where the agent is parametrized according to the robot type, the agent remains in active mode, monitoring the performance of its physical counterpart and waiting for an event, as result of the initial subscription: a plug-in or plug-out of a modular process. When a process module is plugged-in, the robot agent is automatically notified about the change in the cell configuration, storing the info related to the plugged-in process and its location in its local database. This information is assigned to the robot controller that should adapt itself to work with this new modular process at this location, namely downloading the proper robot program and changing its tools. A same procedure occurs when the robot agent receives an event notifying that a process is plugged-out. In this case, the agent updates the unavailability of the process and informs the robot controller that this process is not anymore active. After the reconfiguration of the micro-flow production cell, the cell can return to the processing state, if the cell safety procedures are guaranteed. In fact, aiming to ensure the safety, robot agents should check the information related to the cell safety systems, e.g., light curtains and emergency stops, before allowing to return to the normal processing operation.

4.3. Plug-in and Plug-out Mechanisms

As previously referred, process and robot agents need to interact to reach the seamless reconfiguration procedure. For this purpose, the reconfiguration can be triggered by plugging in and/or plugging out modular processes. Figure 13 describes the sequence diagram to reconfigure the micro-flow cell due to the plug-in of a process module.

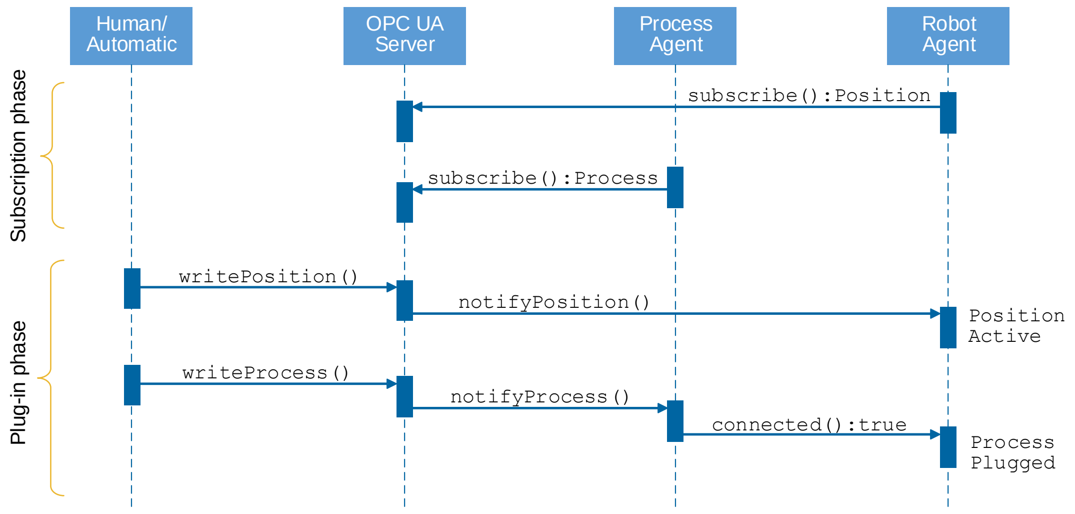


Figure 13: Sequence diagram for the plug-in of a process module

This procedure is divided in two phases, namely the subscription phase and the plug-in phase. On the subscription phase, the robot agent subscribes the cell positions related to its operation scope, while the process agents subscribe their available process modules.

On the plug-in phase, the system is waiting for a reconfiguration, that occurs when the operator plugs-in a process module into a specific position. When a position becomes active, by plugging-in a process module, the robot agent is automatically notified (since it had subscribed this event) that there is a process module incoming into that position. The process agent is also notified that the process module associated to it is plugged, switching its status from *sleep mode* to *active mode* and informing the robot agent (by sending a message), that the process it is connected. In this way, the robot agent knows which process is plugged in each position.

This connectivity can be done in two ways: automatically or manually. If the cell is equipped with sensors that allows to know which positions are active, and the process modules have RFiD (Radio-Frequency IDentification) tags that identifies themselves when attaching a position (e.g., using a RFiD reader), the process identification can be performed in automatic way. Otherwise, this identification can be performed by the operator that plugs-in physically the process module and inserts the position and the process identification through an HMI.

Independently of the physical connection approach, the process identification information is stored in an OPC-UA server, which variables are subscribed by the several robot and process agents. This allows separation between the agent-based system and the physical technological implementation of the micro-flow production cell.

Regarding the plug-out of a process, a similar interaction is performed by the process and robot agents, with the robot agent subscribing the position and the process agent the process modules (Figure 14). When a process module is plugged-out, the robot agent is notified that there is a process out-coming from that position and the process agent is notified that the process is being unplugged. After receiving the notification, the process agent informs the robot agent that the process is disconnected and switches its status to *sleep mode*. In this way, the robot knows automatically which process module is not available anymore and which position becomes inactive.

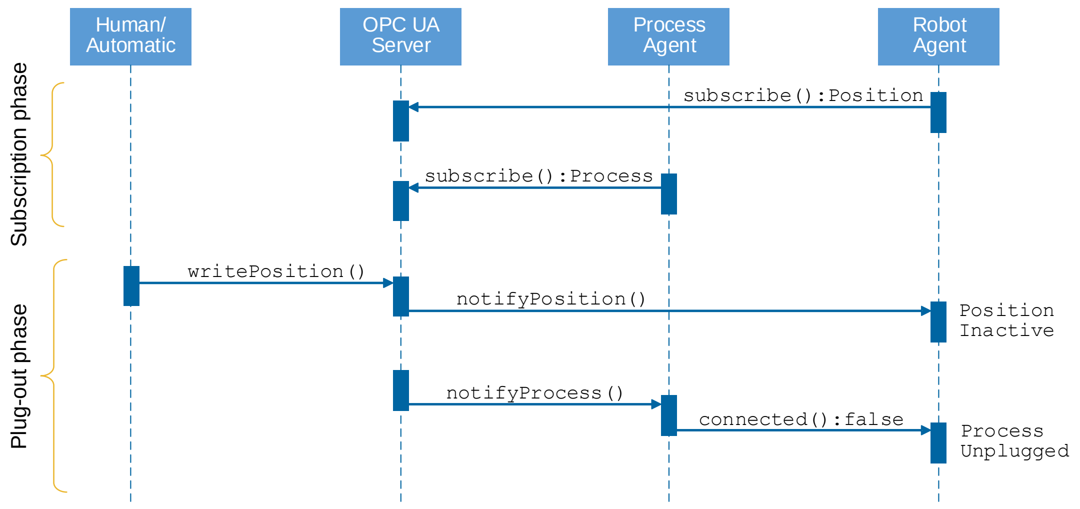


Figure 14: Sequence diagram for the plug-out of a process module

With this distributed and loose coupled behaviour, the agent-based solution can discover in a dynamic and automatic manner the plug-in and plug-out of process modules, allowing to support the seamless reconfiguration of the micro-flow production cell composition.

As mentioned before the reconfiguration occurs according to the production orders coming to the micro-flow production cell as a schedule of jobs to be performed. The need to change the cell reconfiguration comes to the operator as a need to provide the required skills to execute the desired schedule.

4.4. Deployment of the Agent-based Reconfiguration Mechanism

The designed agent-based reconfiguration tool was deployed to support the dynamic reconfiguration of the production cell.

4.4.1. Agent-based Infrastructure

The agent-based infrastructure was implemented using the Java Agent Development Environment (JADE) framework [19]. 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 most of the FIPA specifications, making it an ideal choice for agent simulation and development.

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. Figure 15 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).

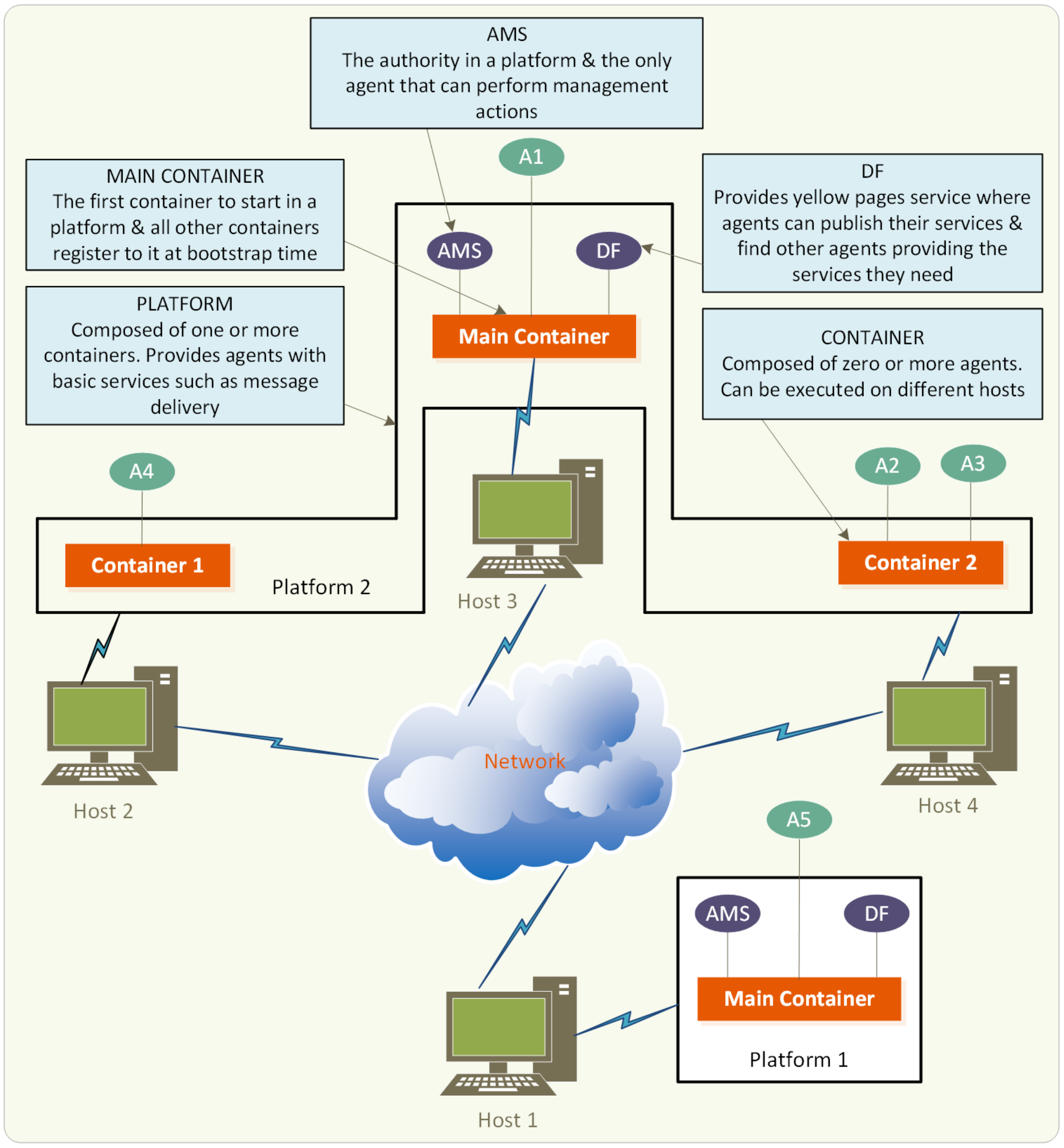

AMS
The authority in a platform & the only agent that can perform management actions
A1
MAIN CONTAINER
The first container to start in a platform & all other containers register to it at bootstrap time
DF
Provides yellow pages service where agents can publish their services & find other agents providing the services they need
AMS
DF
Main Container
PLATFORM
Composed of one or more containers. Provides agents with basic services such as message delivery
CONTAINER
Composed of zero or more agents. Can be executed on different hosts
A4
A2
A3
Container 1
Platform 2
Host 3
Container 2
Host 2
Network
Host 4
A5
AMS
DF
Main Container
Platform 1
Host 1

Figure 15 The JADE architectural elements

The agents communicate to implement their cooperation patterns using messages formatted according to the FIPA-ACL specification. The agents are created according to the number of available process modules and robots. The several instantiated agents are parametrized by parsing an AML (Automation Markup Language) file (see Section V.C for more details), which contains, for each agent, the name, type of device (i.e. process or robot), characteristics of the process/robot, and connection info to the physical device. In this work, seven process agents (brushing, dimension, marking, roughness, grinding, polish and deburring) and one robot agent (managing an ABB industrial robot) were created to manage the reconfiguration of the micro-flow production cell.

All the agents register their skills in the Directory Facilitator (DF), which works like yellow pages services for the agents. It maintains an accurate, complete and timely list of the agents, that can be easily and on-line discovered by the other agents to support the seamless reconfiguration.

4.4.2. Interconnecting the Physical Devices

The agents are interconnecting robots and machinery, by using the OPC-UA standard (IEC62541) [20], which is aligned with the industry trends in factory automation to exchange real-time data information (Figure 16). The connection between the OPC-UA server and the PLC is established through Modbus TCP/IP Ethernet and with the robot through Ethernet.

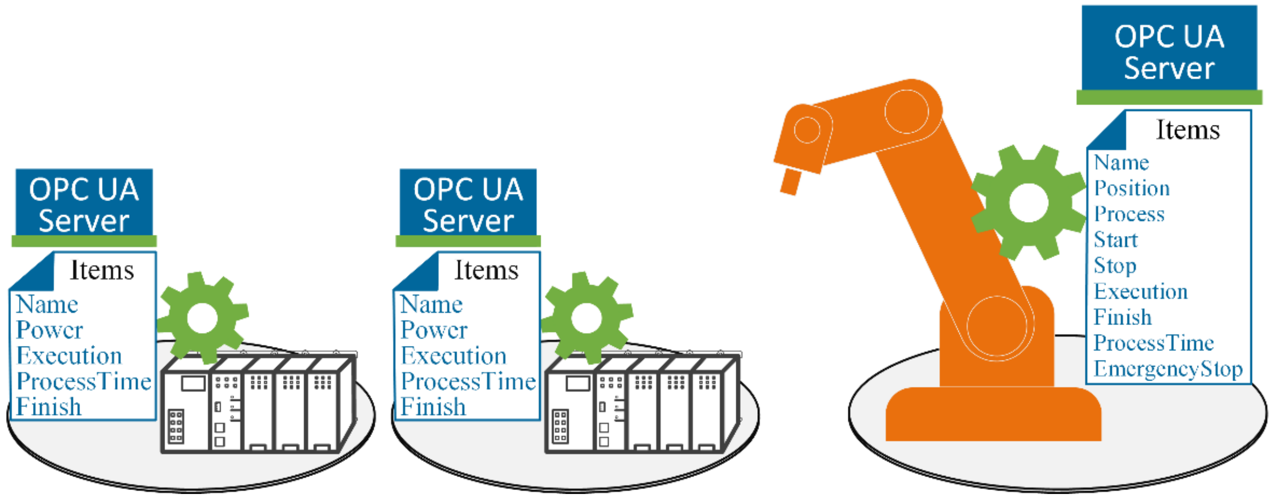


Figure 16 Interfacing agents with physical hardware devices using OPC-UA

The base modelling of the OPC-UA is the concept of Nodes, which represent instances. A *NodeId* uniquely identifies a Node in the OPC-UA server and is used to address the Node. An example of a used *NodeId* is listed below, namely for the position "P1" of the micro-flow cell (other nodes types).

```
NodeId nodeIdCellP1 = NodeId.parseNodeId("ns=2;s=Robot.ABB.IRB1400.bool.P1");
```

The agents are running in the cell computer and exchange data with the OPC-UA server by means of standard interfaces and using mainly three methods, namely, *read*, *write* and *subscribe*. The subscribe method is of particular importance to ensure the event notification of desired events. In this case, a subscription can monitor multiple Items, where a Monitored Item is used to define the attribute of a Node that should be monitored for data changes.

The monitoring of the changes on the Node related to the position P1 is described in the next excerpt of code.

```
public   void   uaSubscribe()   throws   ServiceException,   StatusException   {
subscription = new Subscription();

       MonitoredDataItem              itemPlugP1              =              new
MonitoredDataItem(opcUaClient.nodeIdPlugP1,                  Attributes.Value,
MonitoringMode.Reporting);

       subscription.addItem(itemPlugP1);

       itemPlugP1.setDataChangeListener(dataChangeListenerPlugP1);

    opcUaClient.client.addSubscription(subscription);

}
```

```
public void uaSubscribe() throws ServiceException, StatusException {
             subscription = new Subscription();

MonitoredDataItem itemPlugP1 = new MonitoredDataItem(opcUaClient.nodeIdPlugP1,
Attributes.Value, MonitoringMode.Reporting);
       subscription.addItem(itemPlugP1);
       itemPlugP1.setDataChangeListener(dataChangeListenerPlugP1);

       opcUaClient.client.addSubscription(subscription);
}
```

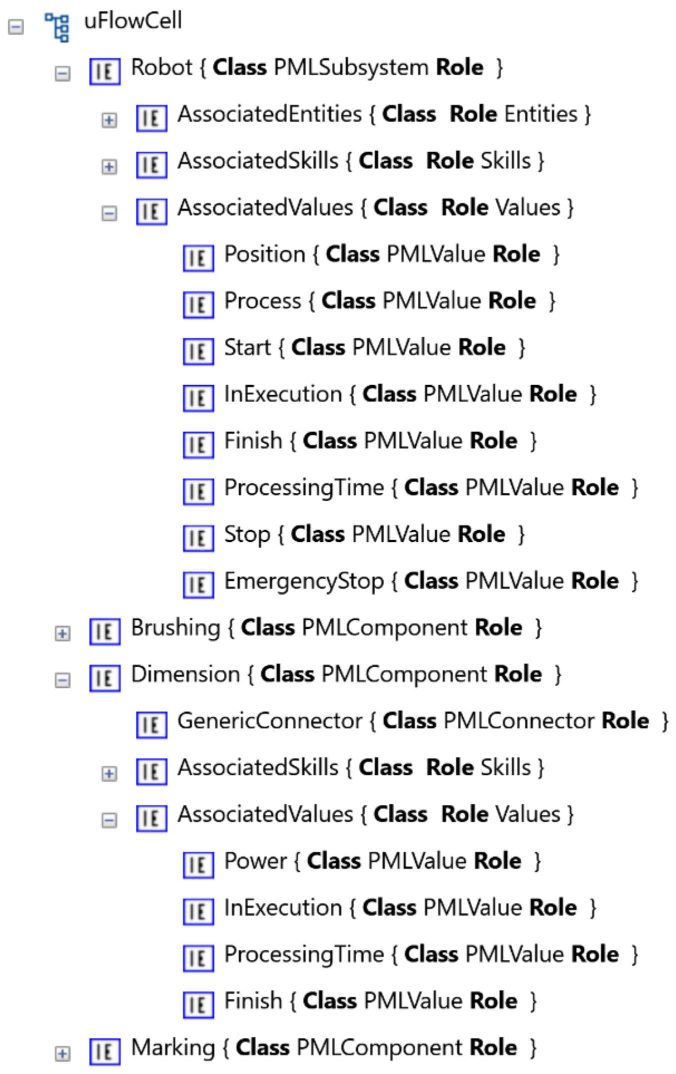
In this way, the Monitored Item is used to subscribe for data changes on the Attribute value of a Node. According to a publish interval, when the value changes, the client who has subscribed the event will receive a notification.

4.4.3.Data Model Instantiation

The part of the data manipulated by the agent-based reconfiguration tool is described using the PERFoRMML (PERFoRM Markup Language) data model specified in [21], which is based on the AutomationML [22]. PERFORMML uses OPC-UA as data format \slash transport protocol. The *PMLEntity* class is the generic representation of shop floor entities, encapsulating the information of components and subsystems. On the *PMLComponen*t or *PMLSubsystem* it is defined the associated entities, skills and values. The *PLMValue* elements enable the basic representation of information pertaining to data machinery level, namely in terms of parameters, e.g., ***ID*** - a string that serve as unique identifier for this element, ***Description*** - a string containing the description of the element, ***Address*** - a string containing the address of a value (OPC-UA).

The part of the data model containing the description of the topology of the micro flow cell (Figure 17) results in an AML file. AML is an XML-based data format built upon other well established, open standards spanning several engineering areas, e.g., the Computer Aided Engineering Exchange (CAEX) that serves as the basis of hierarchical plant structures aiming at interconnecting them [23].

In this AML file, the Instance Hierarchy comprises the topology of the cell, being built by adding the "elements" from the System unit class, which contains all the needed elements to describe the micro-flow cell.



```
uFlowCell
  Robot { Class PMLSubsystem Role }
    AssociatedEntities { Class  Role Entities }
    AssociatedSkills { Class  Role Skills }
    AssociatedValues { Class  Role Values }
      Position { Class PMLValue Role }
      Process { Class PMLValue Role }
      Start { Class PMLValue Role }
      InExecution { Class PMLValue Role }
      Finish { Class PMLValue Role }
      ProcessingTime { Class PMLValue Role }
      Stop { Class PMLValue Role }
      EmergencyStop { Class PMLValue Role }
  Brushing { Class PMLComponent Role }
  Dimension { Class PMLComponent Role }
    GenericConnector { Class PMLConnector Role }
    AssociatedSkills { Class  Role Skills }
    AssociatedValues { Class  Role Values }
      Power { Class PMLValue Role }
      InExecution { Class PMLValue Role }
      ProcessingTime { Class PMLValue Role }
      Finish { Class PMLValue Role }
  Marking { Class PMLComponent Role }
```

Figure 17 Topology of the micro-flow cell represented using PERFoRMML

5. Conclusions

There is a growing number of companies entering into the manufacturing industry, meaning that for manufacturers to achieve and maintain competitiveness, they must adapt to rapid changes in the market, which is the goal of RMSs. Traditional RMSs are operated around part families, so parts within a particular family can be produced using a specific configuration of the resources.

In make-to-order short life-cycle environments, sequential production of part families will lead to frequent reconfiguration and hence excessive costs and significant loss of production time.

A reconfiguration planning and execution mechanism for concurrent production of parts from multiple families has been designed in this task. In the proposed method, genetic algorithm is combined with distributed multi-agent systems (MAS) to achieve optimal reconfiguration plan and resilient reconfiguration process. The output of the optimiser is the reconfiguration schedule, which defines the details of the reconfigurations that should be executed at a particular point in time. Weighted sum of total tardiness and reconfiguration costs was used as the measure of performance in generating the reconfiguration plan. The reconfiguration costs can be adjusted to control the nervousness and the frequency of reconfiguration in the system. The agents manage the actual reconfiguration process, thereby ensuring that the system can deal with unforeseen situations by negotiating with another. The GA algorithm solution was implemented using the Accord.Net Machine Learning Framework and the multi-agent system was implemented using JADE. The developed concept will be validated and demonstrated using the GKN use case in WP10.

References

[1] Elmaraghy, H. A. (2006) “Flexible and reconfigurable manufacturing paradigm” *International Journal of Flexible Manufacturing Systems*, 17, 261-276

[2] Mehrabi, M. G., Ulsoy, A. G., Koren, Y. (2000) "Reconfigurable manufacturing systems: key to future manufacturing" *Journal of Intelligent Manufacturing*, 11, 403-419

[3] Koren, Y., Heisel, U., Jovane, F. Moriwaki, T. Pritshow G., Ulsoy G., and Van Brussel H. (1999) "Reconfigurable machining systems" *CIRP Annals*, 48 (2), 527 – 540

[4] Dou, J., Dai, X. and Meng, Z. (2010) "Optimisation for multi-part flow-line configuration of reconfigurable manufacturing system using GA" *International Journal of Production Research*, 48 (14), 4071 – 4100

[5] Gupta, A., Jain, P. K. and Kumar, D. (2014) "A novel approach for part family formation for reconfiguration manufacturing system" *OPSEARCH,* 51 (1), 76-97

[6] Wang, G, Huang, S., Shang, X and Du, J. (2016) "Formation of part family for reconfigurable manufacturing systems considering bypassing moves and idle machines" *Journal of Manufacturing Systems*, 41, 120-129

[7] Kashkoush, M. and Elmaraghy, H. (2014) "Product family formation for reconfigurable assembly systems" *Procedia CIRP*, 17, 302-307

[8] Deliverable D4.2 – "Planning procedure for energy and agent-based planning and rescheduling". *PERFoRM Project*, 2018

[9] Deliverable D2.2 – "Definition of the system architecture". *PERFoRM Project*, 2016

[10] Deliverable D2.3 – "Specification of the generic interfaces for machinery, control systems and data backbone". *PERFoRM Project*, 2017

[11] Diveev, A. I and Bob, O. V. (2017). "Variational genetic algorithm for NP-hard scheduling problem solution". *Procedia Computer Science*, 103, 52-58

[12] 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.

[13] 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

[14] 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

[15] 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

[16] Ferber, J. (1999) “Multi-agent systems, an introduction to distributed artificial intelligence*”, Addison Wesley.*

[17] Wooldridge, M. (2002) “An introduction to multi-agent systems*”, John Wiley and Sons.*

[18] Muratta, T. (1989) “Petri Nets: Properties, analysis and applications”, IEEE, 77 (4), 541-580.

[19] Bellifemine, F., Caire, G. and Greenwood, D. (2007) “Developing multi-agent systems with JADE”, Wiley.

[20] Mahnke, W., Leitner, S. H. and Damm, M. (2009) “OPC unified architecture”, Springer Verlag.

[21] Peres, R. S., Parreira-Rocha, M., Rocha, A. D., Barbosa, J., Leitao, P. and Barata, J. (2016) “Selection of a data exchange format for industry 4.0 manufacturing systems”, 42nd Annual Conference of the IEEE Industrial Electronics Society, Oct. 23-26, 2016, Florence, Italy.

[22] Drath, R., Luder A., Peschke, J. and Hundt L. (2008) “AutomationML – the glue for seamless Automation Engineering”, *IEEE International Conference on Emerging Technologies and Factory Automation*, Sept. 15-18, 2008, Hamburg, Germany.

[23] Faltinski, S., Niggermann, O., Moriz, N. and Mankowski A. (2012) “AutomationML: From data exchange to system planning and simulation” *IEEE International Conference on Industrial Technology*, March 19-21, 2012, Athens, Greece.