University Utrecht

Emulation of an Agent-Based Reconfigurable Manufacturing Grid

Author:
Laurens van den Brink

Supervisors:
J. J. C. Meyer
D.H. Telgen

November 7, 2015

Abstract

Agile manufacturing becomes more important to provide high-mix, low-volume production for more customized products. Reducing time to market and upscaling from R&D is required to meet changing market demand to stay competitive. Grid manufacturing is agile manufacturing paradigm that uses low-cost equipment, so called equiplets. This is based on the paradigms of Reconfigurable Manufacturing Systems with Agent Technology. Reconfiguring an equiplet will change its capability to adapt based on the manufacturing demand, without downtime of the manufacturing process. An agent-based architecture is presented where autonomous agents coordinate the manufacturing of products through communication. The agents need to handle fluctuations in product demand and disturbance in the manufacturing process. To effectively counter the consequences of disturbances and uncertainties during production, strategies are investigated in a stochastic model. A simulation is built to investigate the currently unexplored aspect of reconfiguration equipment during production. The simulation imitates external events to validate the systems' functionality.

Preface

This thesis is part of the master *Technical Artificial Intelligence*. During the master it is mandatory to attain a number of colloquiums. An interesting talk was given by Daniel Telgen, one of my supervisors during my bachelor graduation project. The subject and practical approach appealed to me so I contacted him to see if there was a possibility to do my master thesis on the subject. The research at the *Research Centre for Technology and Innovation HU University of Applied Sciences Utrecht* ranges from low-cost manufacturing equipment to controlling production with agent technology. I had the freedom to compose the research questions within their project. This gives me the opportunity to explore the most interesting problems, but at the same time it was difficult to set the boundaries of the research.

There are a number of people that I want to thank for the contribution on my thesis. First of all, I want to thank my supervisor Daniel Telgen for giving me the opportunity and support throughout the project. Three articles were written for which I was able to contribute as a co-author. The third article, *Adding Reconfiguration to an Agile Agent Based Production Grid* presented a part of the research in this thesis. I am very grateful for the opportunity to present the article at the FAIM conference in Wolverhampton (UK). I have learned a lot from this experience.

I want to thank my supervisor John Jules Meyer of the *University Utrecht* for allowing me to work on this project. Also, Erik Puik from the Research Centre for Technology and Innovation HU University of Applied Sciences Utrecht.Besides I want to thank Leo van Moergestel, who has laid the scientific basis of a great part of this thesis.

During my work at the HU, several students worked on a research project or as intern during their bachelor *Technische Informatica*. From these students, I would like to thank Tommas Bakker for a lot of discussions related to the project. Roy Scheefhals for discussing ideas of possible errors relating to scheduling in the developed system. I would like to thank Auke de Witte and Lars Veenendaal for checking the grammar of my thesis. Furthermore, I want to thank Alexander Hustinx, Tom Oosterwijk and the others for the great time I had during the time working on my thesis.

Contents

Chapter 1: Introduction

Manufacturing has come a long way since the early production lines. As technology evolves, more operations are automated in the production process. In current manufacturing processes, dedicated equipment is used for product assembly. To stay competitive there is a paradigm shift towards more upgradable and flexible production to provide more customized products. Industrialization of new product developments usually starts with R&D prototypes to serve mainly for testing the functionality of the product. After this phase more dedicated platforms are required to handle higher volumes and increase manufacturing speeds. The manufacturing strategies often lack a clear public roadmap, for example for micro assembly. Therefore, more agile manufacturing strategies are needed to deal with unplanned changes [1]. This has the most impact on high-mix, low volume production.

Changing manufacturing environments are characterized by competition on a global scale. Rapid changes in market demand and process technology require production systems that are easily upgradeable. In addition, these systems require to be prepared for integration of new technologies and functions [2]. Changing market demands are handled by increasing uptime and the speed of dedicated production machines. Invested capital and project risk would increase if new equipment is required to be purchased. Unexpected delays will lead to delayed market introduction. Replacing equipment or changing product design ceases actual production, which leads to loss of income and increasing project risks [3].

Overall trends in various manufacturing sectors are the following: restructuring in all levels of organizations due to globalization of markets; moving management systems from hierarchical structures to more levelled systems reducing middle management; putting more emphasis on decentralized teamwork; acquiring more skills and knowledge to make more intelligent decisions at a lower level. These are trends towards more modularity, autonomy, and self-sufficiency at the lowest possible levels [4].

Manufacturing strategies are improved by adopting more flexible and reusable manufacturing technologies. The equipment would preferably remain unchanged when scaling the production from R&D. In this way scalability issues are prevented and technology is reused. Furthermore, upgrading existing equipment reduces investments in new equipment significantly.

This research continues on the manufacturing paradigm, Grid manufacturing, where low-cost manufacturing equipment is used with agent technology to provide an agile manufacturing system [5]. Industrial applications are still rare as most prototypes lack robustness that is required by the industry [7]. More proven techniques need to be developed for the industry to adopt flexible and reusable manufacturing [6]. This thesis presents an implementation of an architecture for a production grid that is based on Grid Manufacturing. Furthermore, a simulation that enables the software to be validated and the performance to be tested in a dynamic manufacturing environment.

In this thesis, the first chapter describes the concept of intelligent manufacturing system and outlines different manufacturing paradigms to show context of the research. In the second chapter the problem description and research questions are given. Chapter 3 describes the architecture of the system and the Grid Manufacturing paradigm which is applicable on the architecture. In chapter 4 a part of the architecture is described in more detail and optimization strategies to improve the production are explained. In chapter 5 the simulation model is described and the results of the simulations are given. Chapter 6 reflects on this research where-after in chapter 7 related work is reviewed. In chapter 8 conclusions are drawn and chapter 9 describes opportunities for future work.

1.1 Intelligent Manufacturing Systems

Intelligent manufacturing goes beyond classical manufacturing as the latter uses a dedicated production line where machinery or craftsmen are used to do one task of the production process. Intelligent manufacturing systems aim to be more flexible and reactive to uncertainty and changes in production.

1.1.1 Manufacturing Paradigms

Manufacturing systems can be categorized in the related paradigms:

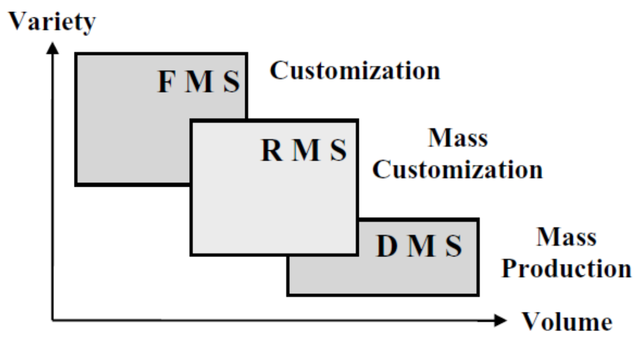


Figure 1.1: Manufacturing Paradigms [8]

Dedicated Manufacturing Systems Dedicated Manufacturing Systems (DMS) prescribes the use of inexpensive, fixed automation equipment to produce high volumes over long periods of time. The systems are usually designed to produce a single part in a production process with a high production rate [9].

Flexible Manufacturing Systems Flexible Manufacturing Systems (FMS) facilitate changes to produce a variety of products with the same manufacturing system in a short time. FMS describes a generalized flexible design to anticipated variations in products which are built-in a priori [10]. Although production rate is lower it provides more flexibility compared to dedicated systems [9].

Reconfigurable Manufacturing Systems Reconfigurable Manufacturing Systems (RMS) aims to combine the high throughput of dedicated systems and the flexibility of FMS [4] [11]. The objective of RMS is to provide the functionality and capacity depending on demand by rapid adjustment of production capacity and functionality. The system structure, hardware, and software components are capable to change. Important properties that come with the RMS paradigm are modularity, integrability, customization, and scalability [2].

1.1.2 Manufacturing Control Systems

An industrial control network is a system of interconnected equipment used to monitor and control physical equipment in industrial environments. This differs from traditional enterprise networks due to the specific requirements of their operation [12]. The industry requires more reliable and real-time networks than for usage by consumers. Industrial networks are composed of more independent layers and specialized components as Programmable Logic Controllers (PLCs), Supervisory Control and Data Acquisitions (SCADA) and Distributed Control Systems (DCSs).

Production control can be divided into layers, see figure 1.2. The agility of production process is controlled within the business layer. The direct control of hardware is done in the process control layer. Two intermediate layers in these industrial systems are of interest in this research: production management and process management. Manufacturing Execution System (MES) is production management software to enable high level control over production facilities in a broad sense. SCADA is a process management layer to supervise the process control devices. These systems are purely software based control systems to supervise process control devices, acquire and monitor production data. The technological advantages, such as Ethernet, started to blur the lines between industrial and commercial networks. Due to the underlying hardware improvements the life cycle and security issues of entire SCADA systems are important considerations to avoid becoming obsolete.

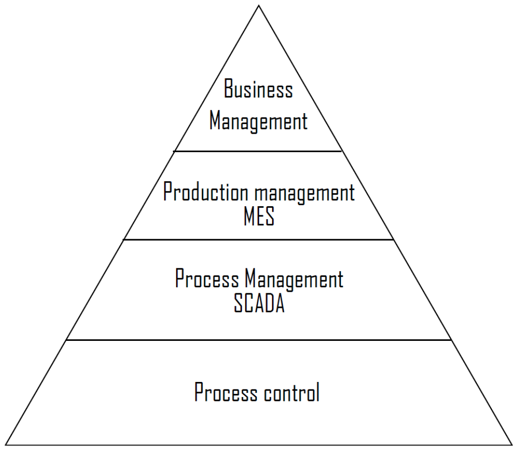


Figure 1.2: Automation Pyramid [13]

1.2 Agile Manufacturing

Agile manufacturing (AM) focuses on the manufacturing enterprise and the business practices needed to adapt to a changing global uncertainty. Agility is defined as “a comprehensive response to the business challenges of profiting from the rapidly changing, continually fragmenting, global markets for high-quality, high-performance, customer configured goods and services” [2].

An agile manufacturing environment creates processes, tools, and a knowledge base that enables the organization to respond quickly to the customer needs and market changes whilst still controlling costs and quality [14]. Flexibility is the key in agile manufacturing. The approach can be achieved through several aspects, e.g., scalability, modularity, and a number of classifications. These manufacturing approaches are applicable when new products are produced that have a short time to market and flexible volumes.

1.2.1 Holonic Manufacturing System

A Holonic Manufacturing System (HMS) is a paradigm describing entities of an organization in living organisms and social organizations. The concept can be used to achieve agile manufacturing systems [15]. The word holon is a combination of the Greek word holos, which means whole, and the suffix on, which means particle. A holon can represent a physical or logical activity. Examples of holons are a robot, a machine, a product order, a flexible manufacturing system, or a human operator. A holon can be composed by several lower-level holons, in contrast to an agent. A holarchy is defined as a system of holons that is organized in a hierarchical structure. The holons cooperate to achieve the systems goals by combining their individual skills and knowledge [7]. Holonic manufacturing concepts can be realized using agent technology. Implementations of holonic concepts in manufacturing are motivated by: 1) improving the evolution of products within an existing production facility and 2) maintaining a satisfactory performance outside of normal operating conditions [15].

1.2.2 Agent Technology

Agent Technology (AT) is a software paradigm bringing concepts from artificial intelligence into distributed systems. A commonly accepted definition by Wooldridge and Jennings [16]: "An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives". Agent-Oriented Programming (AOP) essentially models an application as a collection of components called agents. These agents are characterized by, among other things, autonomy, pro-activity and an ability to communicate [17]. The Multi-Agent Systems (MAS) paradigm is characterized by decentralization and parallel execution of activities based on autonomous entities, called agents. Agents in a multi-agent based system have a high level of autonomy, organize themselves in a heterarchical structure and intelligently cooperate to a common goal. Agent architectures range from reactive to deliberative agents. A BDI architecture is a deliberative architecture. It is designed to implement cognitive agents depending on manipulation of Beliefs, Desires, and Intentions of the agents. This makes the agents more robust against disturbances than reactive agents due to their capability to only react and not anticipate the future [18].

Chapter 2: Problem Description

The goal of this thesis is to provide validation and improve upon the developed production grid that is based on the Grid manufacturing paradigm [5]. A production grid consists of modular and reconfigurable manufacturing platforms, called equiplets. The control system is based upon a distributed and hybrid architecture, and uses agent technology. Both equipment and product are represented by intelligent agents, i.e. a product agent and equiplet agent. The equiplet agent is a virtual representation of the physical equiplet and aims to utilize its capabilities by providing them to the grid. The product agent is responsible during its product manufacturing lifespan in the grid. The manufacturing of a product is divided into product steps that can be performed by an equiplet. The manufacturing system aims to achieve flexibility and scalability through the modularity, reconfigurability and autonomy of the equiplets and product agents.

Based on the previous chapter the following observations can be made:

- Modern manufacturing is characterized by a paradigm shift towards more agile and reconfigurable architectures.
- Changing market demand provides uncertainties and variance in the production process. This is added to existing uncertainties in manufacturing environments [19].
- Industry requires mature and proven technology that are more than just prototypes.

The first observation identifies the challenge: how can reconfiguration be adopted by the production grid. From the second observation follows two problems: what will be the impact of uncertainties and how can the production grid deal with it efficiently. Before addressing these issues, there is the question of how to evaluate functionality and validate the system. This leads to four research questions in the following section.

2.1 Research Questions

1. *Is it possible to emulate and validate the MAS software of a grid with a simulation?*

The literature of agent-based manufacturing is mostly limited to proof of concepts. The reviewed literature confirms the conclusions drawn by Leitão [7] and Trentesaux [6] that to adopt the use of intelligent software in industry, more implementations are needed that go beyond proof of concepts.

The current developed grid manufacturing software is proven with practical approaches. Proof of concepts were developed to validate the system from software to hardware. These concepts are far from real production situations where each part of the system should perform deterministic and in a proven manner [5]. Emulating external events enable the software to be validated by a simulation.

Simulation technology holds tremendous promise for cost reduction, quality improvement, and shortening of the time-to-market for manufactured goods [20]. It plays a significant role in evaluating the design and operational performance of fields including manufacturing systems. Simulations have proven its effectiveness in practice. Discrete simulation is one of the most commonly used techniques for analyzing and understanding the dynamics of manufacturing systems. A brief overview is given of the use of simulation in manufacturing with respect to analysis, performance, verification, and validation [23]. It follows that the following performance evaluations should be examined: product throughput, utilization of equipment, and the accuracy of the schedules.

2. *What impact does reconfigurability of the production platforms have on the production efficiency of the grid?*

A key aspect of agile manufacturing is the ability to adapt to a changing environment. The system should be capable to handle variable and changing product demand regardless the capabilities of the production grid. In this RMS, all modular equipment can be reconfigured during runtime to change the capability of the equiplet. An equiplet agent should be able to handle a reconfiguration without interference to manufacturing process of other equiplets. The behavior and impact of reconfiguring an equiplet in a full size grid should be investigated.

3. *How do disturbances impact the manufacturing system?*

Manufacturing environments are subject to many uncertainties and variance in the production process. Uncertainties about machine breakdowns, maintenance, and the randomness of processing times have major impact on realization of a schedule. The cheap and modular equiplets will likely be more prone to variance in the manufacturing process than more expensive and dedicated equipment. Another factor of variance is the changes in product demand. Both the time between requests to manufacture a product and the type of products, i.e. the combination of product steps, are subject to variation. The actual values of these random factors are unknown beforehand, but become known at the time the variance occurs. So for example the processing time of a product step is known after completion. This affects scheduling of the products and the coordination of the manufacturing process between the product and equiplet agent. Strategies should be developed to deal with these changing factors.

An important factor in the performance of the grid is the planning of products. Van Moergestel has researched scheduling mechanisms and introduced a solution for this scheduling problem [24]. The schedule algorithm is tested in a deterministic setting. This might prove to be insufficient in a more dynamic and stochastic environment.

4. *What strategies or optimizations can counter the effect of disturbances in the manufacturing process?*

What is the best action when the schedule can't be executed as planned. The scheduling algorithm describes three sequential steps to identify the best route of equiplets in the grid. It should be investigated whether these sequential steps could be performed and would yield the same results. A schedule algorithm that is more focused on robustness can be a solution. A possibility would be to follow the planning less strictly and use this as a guideline.

2.2 Research Methodology

The research is conducted at an applied university which implies there is a more practical approach. Functionalities are proven with proofs of concepts lacking a complete integration of all developed concepts.

The agent-based software needs to be developed. It should combine the functionalities of the proof of concepts. The system should work with the already developed software for initiating the production of a product and the software modules for controlling the equipment. The ability for equiplets to reconfigure should be integrated in the MAS. Furthermore, the strategies should be implemented for improving the performance of the production grid.

For emulation of the manufacturing grid, the agents need to be able to receive external events from a simulation. A simulation layer needs to be developed extending the software. Interfaces of the agents should intercept outgoing messages in order to produce new external events. This enables that agents can be tested on a large variety of cases. The cases range from executing a set of product steps on a real equiplet to manufacturing many products in a large manufacturing grid within a dynamic environment.

Below, a list of functionalities/constraints is given for the development of the software. However, not directly applicable in the current research these are identified as potential problems during up-scaling of the production [7]. The system should considered to be placed in a larger context of industrial flexible manufacturing. In future stadia of the project possibilities will become available for testing functionalities or constraints.

- The MAS should aim not to overflow the system with communication messages which can put too much stress on the system. A product agent inquires whether an equiplet agent is capable to perform certain services. The equiplet agents should also return their planning and the agent that are scheduled in a certain time period. By giving the product agent how much free time they still have, the agent can choose the equiplet agent with the smallest workload. This should result in a more robust product schedule and a more distributed workload divided between the equiplets.

- Agents in the grid should be pro-active and cooperate with each other. When agents receive requests they should reply with a constructive answer. When a product agent asks an equiplet agent whether it could perform a production step a certain time, the agent should not merely reply 'no', but also give an alternative time for the production step.

- Product agents should be able to be initialized by multiple entities. The production of a product can be optimized by dividing the product into half-products. These sub-products will consist of parts that are separately manufactured to enable parallel production. For example, when parts required to be produced by different equiplets it does not matter in which order. However, this optimization is not yet applicable and omitted in the current research. Within the design of the product agent it should be considered that the product agent could report to a parent agent or a human interface.

- The product agent will have to allocate resources for the production of the product. This would be either before or as a part of the initialization process. It is possible that certain resources remain after the production process. Product agents should be able to delegate the return of the otherwise wasted resources in order to be used by other product agents.

- Test-Driven Development (TDD) is a software developing paradigm where the developer writes automated test cases for functionality or part of code which would benefit the overall quality of the software. This will contribute to a more proven system. The simulation and MAS is a distributed system where there is communication between many entities. Testing the system to its maximum capacity needs to be possible while maintaining an overview of the correct working of functionality. Debugging while increasing the entities becomes more difficult as the communication between these entities increases. If possible, dedicated test code should be written for part of the software to verify the functionality.

Chapter 3: Architecture

The REXOS architecture is based on a hybrid system using MAS for the deliberative aspects and Robot Operating System (ROS) [27] for the reactive aspects. The term REXOS comes from Reconfigurable EQuipletS Operating System where the abbreviation of eqs is evolved to ex. The architecture consists of three layers, MAS, HAL, and ROS. Basically, autonomous entities in the MAS have cognitive abilities to make the necessary decisions using entities in ROS for direct hardware interfacing. The Hardware Abstraction Layer (HAL) layer is responsible for the translation of product steps to instructions for the equiplets' hardware.

This chapter first describes grid manufacturing which is used in REXOS. Thereafter, an overview is given of the implemented system with the ROS and HAL layer.

3.1 Grid Manufacturing

Grid Manufacturing (GM) is a new production paradigm [26]. It is based upon the use of standardized and modular Reconfigurable Manufacturing Systems. A production grid, or in short a grid, is a group of low-cost manufacturing platforms in a dynamic logistic set-up that can individually be reconfigured, so product and parts can be transported between all systems. The modular and reconfigurable manufacturing platforms are called equiplets. The concept has been introduced by Puik [1].

Definition 1 (Equiplet)**.** An equiplet is a reconfigurable manufacturing device that consists of a standard base system upon which one or more front-ends with certain capabilities can be attached.

Advantages of grid manufacturing are more flexibility and risk reduction during scaling of the production. Manufacturing tasks are distributed over a large number of production systems to achieve the flexibility and risk reduction. Adding equiplets to the grid could lead to increased uptime as multiple equiplets can work parallel. This way bottlenecks in the production process are reduced. During runtime individual equiplets are able to be reconfigured without affecting other equiplets. Consequently, the production grid changes without downtime of the manufacturing process.

Agent technology provides opportunities for the use in grid manufacturing. Grid manufacturing increases the flexibility by providing a heterachical software architecture where product and equiplet agents negotiate directly to create new products.

3.1.1 GEM Architecture

The implementation of the GEM architecture consists out of three layers, a Grid, Equiplet, and Module layer shown in figure 3.1. In the GEM architecture the equiplets in the grid are represented by an agent and nodes. The equiplet agent is a virtual representation of a physical equiplet. A node is a process that performs computations for controlling equiplet modules and is implemented in ROS. A module is a component of an equiplet with the controlling software. The modules directly control the hardware modules. The equiplet is capable to perform manufacturing steps with a combination of modules. The agent is able to communicate through a blackboard with his underlying hardware.

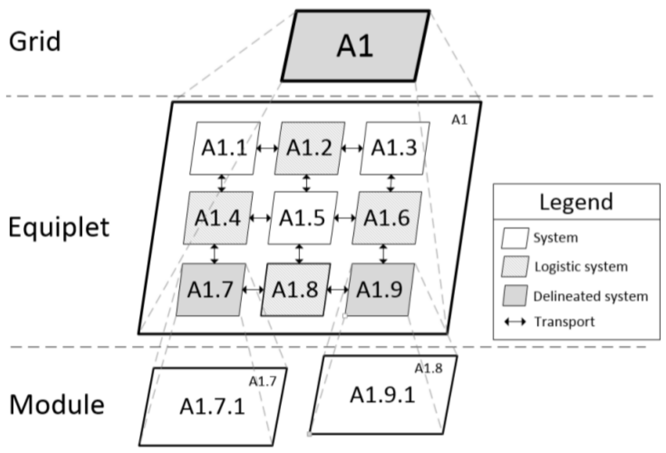


Figure 3.1: Grid system overview - GEM architecture - Grid, Equiplet, Module [26]

An equiplet is represented as an equiplet agent. The agent will publish its capabilities in a global accessible place therefore providing a service to the grid. A product is represented by a product agent during its manufacturing lifetime in the grid. The agent chooses the equiplet agents to perform product steps on basis of the published production steps. When a product agent is instantiated, it will try to plan a path among the equiplets. If the production path is feasible and planned, the corresponding time of the equiplets will be claimed by the product agent. If the deadline is not feasible, the product agent will negotiate with other product agents to see if they are willing to adjust their scheduling.

3.1.2 Decomposition of Products

The goal of a product agent is to complete a product. Products need to be decomposed into steps lower level control software can understand in order to create the product, i.e. the product agent needs to split-up the production of the product into steps which equiplet agents can understand. The products steps can be classified into four classes: altering the shape of the product (e.g. drilling, heating), adding components to the product (e.g. gluing, welding), inspecting the product for quality control, and testing the product.

Definition 2 (Production Step)**.** A production step is an action or group of coordinated or coherent actions on a product, to bring the product a step further to its final realization. The states of the product before and after the step are stable, meaning that the time it takes to do the next step is irrelevant and that the product can be transported or temporally stored between two steps.

Before executing of these steps, more specific instructions need to be formulated. These steps do not prescribe the necessary hardware, but only the action that needs to be performed. Equiplets translate these steps into instructions the underlying modules can execute.

3.1.3 Hardware Representation

In grid manufacturing an equiplet agent is a virtual representation of a manufacturing machine. This equiplet contains multiple modules that control hardware, i.e. sensors and actuators. Modules are specific systems that can individually receive instructions. Each module has its unique state that is the direct representation of the hardware [26]. An error that can be detected at the (lower) device level will influence the modules' state.

3.2 Overview

Before explaining the architecture in more detail, an overview is given of the exchanged information between entities is shown in figure 3.2. To achieve modularity in the system, a clear division is made between the layers. The product agent receives product steps containing a required services and more detailed information about the execution of service. The equiplets register the provided services by the Directory Facilitator. The capabilities depend on the installed modules which are provided by a controller within the HAL layer. The software for module control is given by a user and stored in the HAL layer. The product agent will confirm the capabilities of the equiplet agent before the execution of product steps. Product steps are translated into instruction in the HAL layer and executed in the ROS layer.

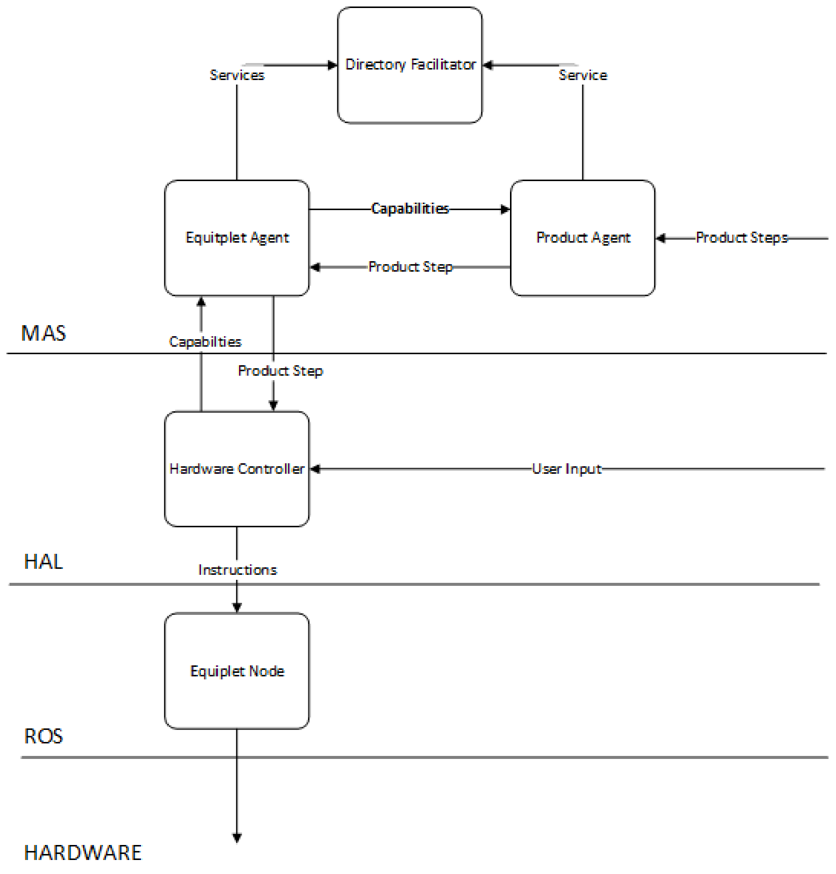


Figure 3.2: REXOS dependencies

Compared to previous implementations [5] the hardware, service, and equiplet agents are merged to equiplet agent and a HAL. As information for manufacturing a product steps is translated by the HAL and passed through the correct software module in ROS. Information of completed instructions, failures, or other complications is passed from ROS to the equiplet agent in order to take the necessary actions.

In figure 3.3 an overview is given of the architecture. The loosely coupled and highly autonomous entities in the distributed manufacturing system resemble a heterarchical control architecture. The grid is a combination of heterarchical and hierarchical approaches [28]. The equiplet agents have a hierarchical structure while being inside a heterarchical MAS architecture. The entities in MAS are able to run on different computers provided that there is an Ethernet connection between the computers. The grid has one central server from which the system is started and therefore contains the Directory Facilitator, Grid Data Acquisition, Logistic Manager, and other central agents not shown i.e. the Agent Management System as this comes with the used framework. Furthermore, the grid is capable of containing multiple products, equiplets, and transport agents. The equiplet agent itself is designed to work on the same computer as the software for controlling the hardware.

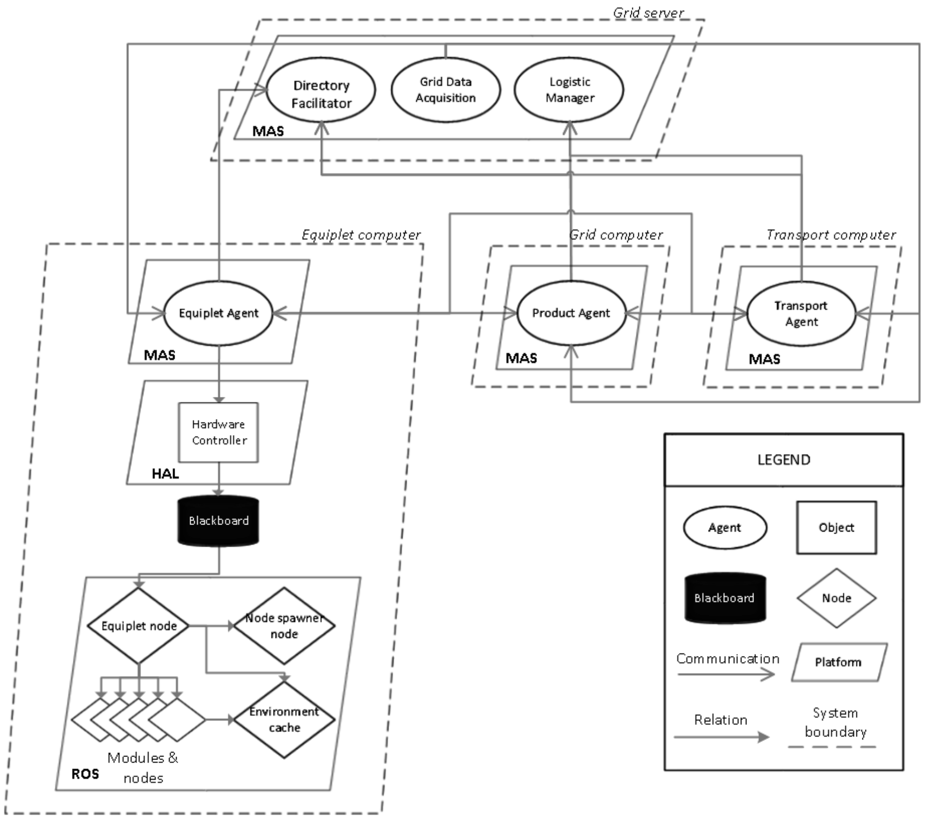


Figure 3.3: Architecture

3.2.1 ROS

ROS is a software framework that provides a middle-ware system and libraries for hardware abstraction [27]. The ROS platform is used to directly control all hardware modules. The middle-ware system uses simple autonomous objects called nodes which communicate through a publish and subscribe service. ROS has an extensive library for robot sensors and actuators. Every module has a ROS node that is used as an interface for a module. Since modules can be adapted when an equiplet is reconfigured, a spawner node is used to start new modules when required. The environment node is used together with a computer vision module to find the location of products within the working environment of the equiplet.

3.2.2 HAL

The Hardware Abstraction Layer enables the equiplet agent to translate product steps to specific hardware instructions and therefore to execute product steps. Each device in the production environment has certain properties and behaviors which can be classified as functional capabilities [29]. Capabilities have parameters, which represent the technical properties and constraints of resources, such as speed, torque, payload, and so on. For example, the capability with concept name 'moving', has parameters 'velocity' and 'acceleration'. The capability parameters enable to determine which resource has the capability that best fits the need of the given product or production requirement.

Combinations of modules define the equiplets capabilities. The HAL layer contains the software for controlling the underlying modules which is stored in a knowledge database. For an equiplet it is common that multiple modules are dependent on each other. From these module dependencies a tree structure emerges where a sub-tree comprises an equiplets capability. For example, a delta robot module with a gripper module will enable the equiplet to pick and place or a delta robot with a pen module will enable it to make drawing.

With a tool or interface the required module software can be loaded to the equiplet. An equiplet can be set in a 'safe' mode where it uses a camera system to detect 2D bar-codes that identifies a module. These QR codes are used to identify a unique identifier and type of the connected modules. This way the corresponding nodes that belong to the modules can be removed or added from the equiplet by scanning the correct QR codes. Before loading software of new modules specific parameters that are required for correct use should be configuration.

Chapter 4: MAS

In this chapter the developed MAS is described. First an explanation of the agents in MAS is given. The equiplet and product agents are described in more detail. These agents have the most influence on the manufacturing process. The product agent has a scheduling procedure for planning its product steps. It consists of finding equiplets for its product steps and planning these steps at the equiplets. Manufacturing environments are subject to uncertainties and variance in the production process. The strategies to improve the production are rescheduling, queue jumping, and reconfiguration.

The Multi-Agent System is a major part of the system. It is responsible for the deliberative aspect in the grid. The MAS consists of a collection of agents in the JADE framework [30]. The MAS architecture is organized according to the following characteristics:

- Autonomy: agents make independent decisions and are responsible for the execution of the decisions toward successful completion.
- Cooperation: agents work together as a group to respond to events and toward completion of their goals.
- Communication: agents share a common language for communication in order to cooperate.
- Reactive: agents should be able to react on changes in their environment and have the capability to detect failures and to isolate failures based on the environment.

Agents in MAS are capable to interact using a common language. The language used in the grid is compliant with the standard of the Foundation for Intelligent Physical Agents (FIPA). FIPA is an organization founded to standardize agent systems [31]. The structure of a message is defined by the Agent Communication Language (ACL) which prescribes messages to contain: sender, receiver, content, performative, conversation-id, in-reply-to, language, and ontology. The sender and receiver are the agents who communicate the content. Prescribed by the custom defined *grid-ontology*, the content that is communicated between agents is formatted in JSON. The performative denotes the type of the communicative act of the ACL message, e.g. *inform*, *request*, *propose*, or other performatives. The conversation describes the topic of the communication, e.g. *can-execute*, *product-arrived*. The agent will reply with the appropriate information or an acknowledgement to incoming messages. The agent will match on the performatives and conversation to correctly de-serialize content and respond accordingly.

For transporting parts between equiplets an Automated Guided Vehicle (AGV) is used. The AGV is represented by a transport agent. A product schedules a transport agent during the lifespan of a product assembly. Although, transport agents could be scheduled for designated trips between equiplets. The AGV needs to be capable of transporting all the required parts for the production from begin of the assembly until a product is finished. Thereafter the AGV could be reused for the next product assembly.

On grid level a logistic manager agent is responsible for the coordination of the transportation units. The logistic manager knows the routes between the equiplets in the grid and therefore has useful travel information for product agent during scheduling. Depending on the capabilities of the transportation units, the logistic manager is able to assign routes to transport units to avoid collisions. This could be a solution when the units cannot drive completely autonomous when there is no sufficient collision detection available or it is not desirable as paths are too narrow to avoid each other. Alternatively, transport units could coordinate through communication and whenever there is a disagreement resolve the allocation of a route with an auction. The best solution would heavily depend on the available transport units, grid set-up, and manufacturing facility.

Before discussing the equiplet and product agent, the overall properties of MAS are described.

4.1 MAS Properties

A grid G consists of a set of product agents P, a set of equiplet agents E, a set of transport agents T, and the management agents M. The management agents are a monitoring agent, Logistic Manager Agent, Agent Management System (AMS), and DF agent. From these agents there is usually one instance in the grid.

$$G = \langle P, E, T, M \rangle$$

A grid is capable of manufacture a product P_i if all the product steps are capable of be manufactured by the grid.

A product to be built is divided into product steps. A product step σ consists of a service and criteria: $\sigma = \langle s, c \rangle$, where s is the service that can be provided by a grid and c are the criteria of the product step to be performed. For example, a service can be *pick and place* with the criteria where the object needs to be picked up and coordinates where to place the object. Furthermore, the criteria of a product step provide information about the dimension and material of the objects needed for the execution of the product step.

Whether a grid is capable to manufacture a product step depends on the capabilities of the equiplets.

Definition 3 (Capability of Equiplet)**.** A capability C_e of an equiplet e is defined as the services that the equiplet provides to the grid combined with the limitations of this service. $C_e = \langle s, l \rangle$, where s is a service and l is the limitations of the service s. The limitations contain among others the boundaries of the service. For example, a service *pick and place* only works with object not too large, heavy, and within the equiplets reach.

An equiplet is capable of manufacturing a product step, if the equiplet provides the service and the criteria are within the limitations of the equiplet. The product agent can search for a path along the equiplets if the grid is capable to manufacture its product steps.

Definition 4 (Production Path)**.** The product is characterized by a sequence of production steps. Consider a product p to be built with three production steps, this product has production path: $\langle \sigma_5, \sigma_2, \sigma_4 \rangle$

The order in a production path is important as most assembly steps have priorities. Normally, each product step requires the completion of the step before it. However, there are situations where first two half-products are made and combined. When the order of two or more product steps is irrelevant these can be viewed as half or sub products. The product agent can choose to spawn child agents to manufacture half products in parallel. When the child is finished it will transfer the production information to the parent agent that will finish the production. If there is a deadline, the parent will coordinate the scheduling of the children with its own scheduling. There are cases where it is desirable to have half products in stock. When a product requires one of these half products it includes the product steps if the stock is empty. In a car factory certain half products can be identified such as the engine and chassis.

4.2 The Equiplet Agent

The equiplet agent is a virtual representation of a physical equiplet and is aware of the capabilities of the hardware. The goal of the agent is to utilize his capabilities by providing them to the production grid. The agent will initialize the HAL through which it will receive the knowledge about its capabilities. Furthermore, the equiplet agent has knowledge about his position in the grid, the product steps that need to be executed and have been executed. On the basis of the executed task the equiplet could provide a more specific time indication for product steps which is needed for product scheduling. This first time indication should be provided along with the capabilities.

The equiplet agent has one behavior that listens to communication from for example the product agent. Incoming message conversations are *product-arrived*, *product-release*, *product-delayed*, *schedule*, *can-execute*, and *information-request*. A description of these conversations is:

- *product-arrived*: A product informs the it has arrived at the equiplet and its product step is ready for execution. The equiplet will begin with the execution of the product step if it is favorable according to his schedule.
- *product-release*: A product agent releases its reserved time slots at the equiplet. The equiplet removes the time slots of its schedule.
- *product-delayed*: The product indicates that it will be delayed. The equiplet will tolerate this if is it does not affect other product agents.
- *schedule*: The product agent requests to reserve one or more timeslots at the equiplet.
- *can-execute*: The product agent seeks confirmation whether the equiplet can execute product steps. The equiplet agent will check whether the criteria of the product step are within the limitation of his capabilities.
- *information-request*: A monitoring agent is able to request information of the equiplet. The equiplet agent can provide information about the schedule, current state, history and/or more.

4.3 The Product Agent

The products that are manufactured by the grid are represented by a product agent. The product agent is responsible for the production of a product from the moment there is a product order until the created product leaves the grid. The agent will be initialized with the required parts ready for manufacturing or claim the parts by an appropriate resource manager in the grid. The goal is to ensure that the product order is completed and to supervise the progress. The agent will inform the creator of faults and other state changes during manufacturing.

A product agent is able to have 6 different states: *scheduling*, *traveling*, *waiting*, *processing*, *error* or *finished*. A state transition diagram is shown in figure 4.1. After the scheduling is successfully completed the agent the state transitions from *scheduling* to *traveling*. The product will travel to an equiplet where it will wait until the equiplet agent informs it starts processing its product step. The state transitions from *traveling* to *waiting*, and when the equiplet begins with manufacturing the product step, to *processing*. If the equiplet agent informs the product that its step is finished, the product is either done with production or will travel to the next equiplet in its production path. So the agent will be in the state *finished* or the same transitions happen as after successful scheduling. If it is the same equiplet, the product agent immediately notifies it has arrived. If the scheduling was not successful the product agent will be in the *error* state. The agent informs the creator of the product as either product cannot be made within the deadline or the required equiplets are not available in the grid.

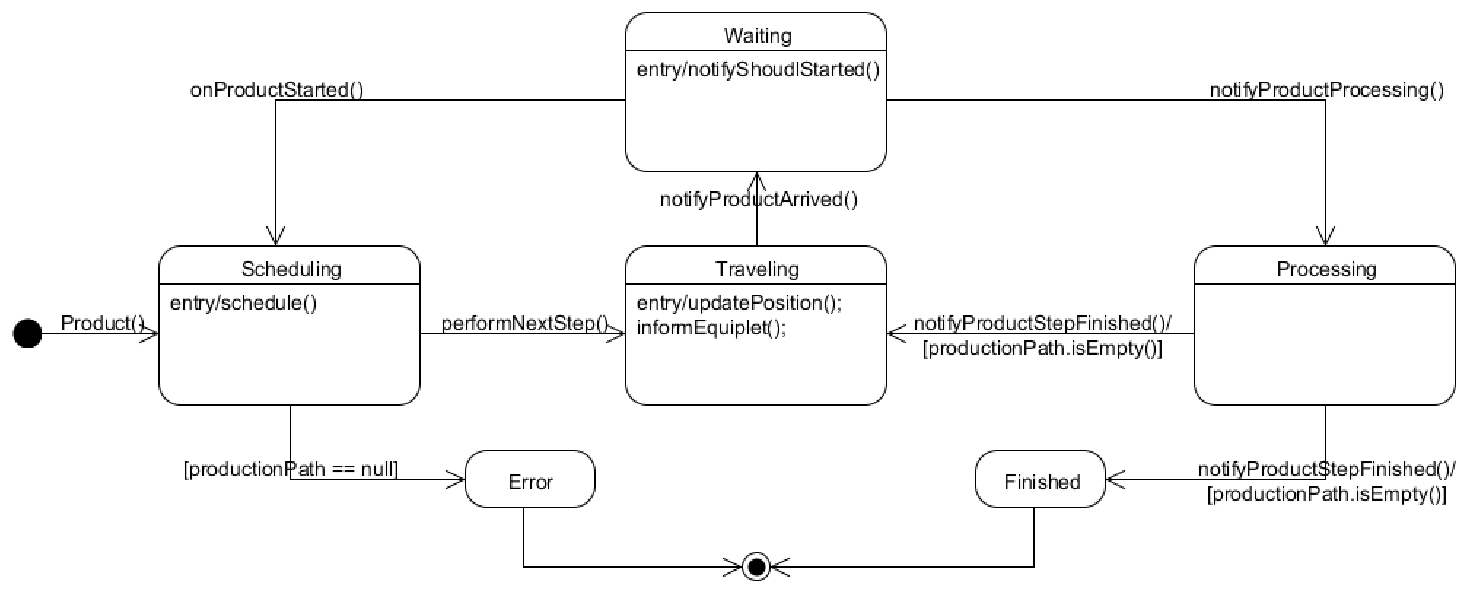


Figure 4.1: Product STD

A product agent has two behaviors: a listener and a scheduling behavior. The listener behavior is similar to that of the equiplet agent. It is active during the lifetime of the product agent. The incoming messages the agent can expect are:

- *product-processing*: The equiplet agent informs the product agent it starts with the processing of his product step.
- *product-delayed*: The equiplet agent informs the planned product step will be delayed such that the product agent can take precaution to arrive in time at other equiplets in the production path.
- *product-finished*: The equiplet agent informs the product agent it is finished with the execution of the product step. The product agent will be able to commission the transportation to the next equiplet or the product is finished.
- *information-request*: A monitoring agent is able to request information of the product. The product agent provides information about his current state, location and progress.

The schedule behavior will be instantiated when the agent is in the *scheduling* state. It will schedule time slots with the equiplet agents according to the product scheduling goals. A product agent can prefer to minimize its makespan or balance the load between equiplets depending on the goal of the agent.

4.4 Scheduling

In manufacturing, allocation of resources is of major importance to the system. The goal is to optimize production and therefore maximize the utilization of the production machines. The problem is similar to that of job shop scheduling where jobs are assigned to resources at certain times. The products steps can be seen as jobs and the equiplet as resources. There are different approaches in methods for solving this scheduling problem. The problem is well known to be NP-hard [19]. Many heuristic algorithms have been developed including dispatching priority rules, local search, and shifting bottleneck procedure [32].

There are fundamental differences in scheduling of equiplets compared to classical job shop scheduling. Usually, the scheduling algorithms assume jobs are known beforehand and take some computing time to calculate a schedule. As products can arrive at any time, a lot of processing power is necessary to calculate a new schedule each time a product arrives. This would lead to redundant use of processing time as the newly arrived product makes the current schedule infeasible when it is not scheduled. So scheduling can begin from scratch with the new situation. Further, scheduling is usually done by a central scheduling entity. This entity needs to have knowledge of all the machines and products in the system. Assigning tasks in a more distributed way gives the agents a higher degree of autonomy [33].

Van Moergestel [24] describes a combination of a heterarchical and hierarchical approach where product agents schedule their product steps on a central plan blackboard. If the product agent fails to schedule its product steps within a deadline, the agent asks other agents if they could release or exchange time slots to make its schedule feasible. The schedule procedure looks first for equiplets providing the services to match product steps where after it schedules the product steps with the equiplets.

4.4.1 Product Step Matching

When product agents are created in the grid, they need to find equiplets for executing their product steps. In JADE there is a DF (Directory Facilitator) that provides a yellow pages service by means of which an agent can find other agents providing the services it requires in order to achieve his goals [17]. The "yellow pages" service allows agents to publish one or more services they provide so other agents can find each other. An agent wishing to publish one or more services must provide the DF with a description. A service description includes the service type, the service name, the languages, and ontologies required to use the service.

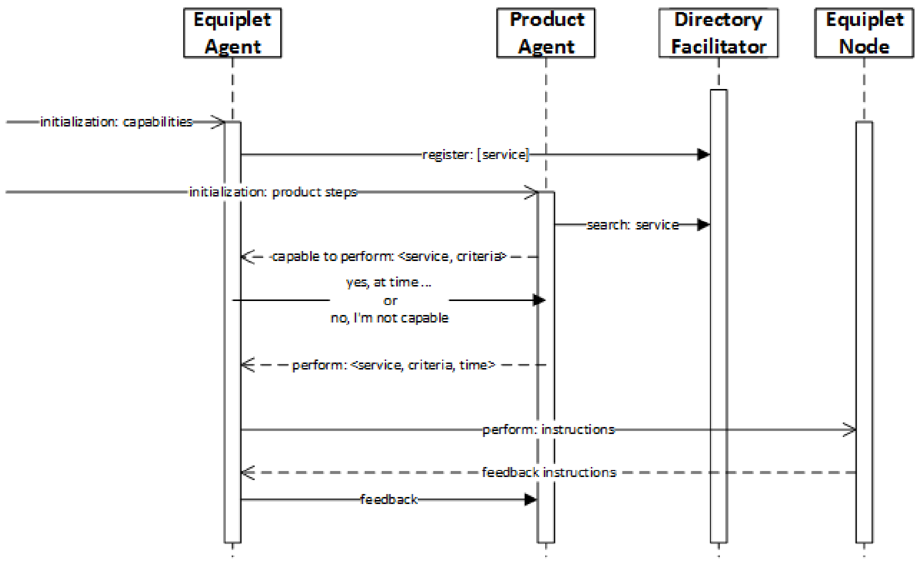


Figure 4.2: Communication from initialization until performing a product step

Figure 4.2 shows a equiplet agent registers their services with the DF. The product agent asks the DF with a description containing the service for the equiplets providing the service, which is represented by *searchEquiplets* in algorithm 1. After finding the equiplets, the product agent will ask whether the equiplet is capable to execute the step and information about its schedule. This is to check if certain criteria of the product step are set. The equiplet tests if the step can be translated into instructions for its modules. The time the product will take to travel to each equiplet will be provided by the logistic manager. With this information the product calculates a feasible production path which can be scheduled by the equiplets.

Algorithm 1 Product Step Matching

```
for all $\sigma \in P_i$ do
    $(s, c) \leftarrow \sigma$
    $E \leftarrow \text{SEARCHEQUIPLETS}(s)$
    $E_{capable} := \{\}$
    for all $e \in E$ do
        if $C_e(\sigma)$ then
            $E_{capable} := E_{capable} \cup \{e\}$
        end if
    end for
end for
$travelTimes := \text{RETREIVETRAVELTIMES}(E_{capable})$
$productionPath := \text{SCHEDULE}(E_{capable}, travelTimes)$
```

4.4.2 Schedule Algorithm

After the necessary information is gathered, a schedule algorithm will match the product steps with equiplets. The schedule algorithm is implemented by performing a sequence of steps [24]: A matrix is created, where each row is a product step σ and each column an equiplet e. If the step σ_i is supported by the equiplet e_j then, the cell $\alpha_{ij} = 1$, otherwise $\alpha_{ij} = 0$. The next step is to minimize the transport between equiplet by increasing the cells of equiplets that are capable to manufacture multiple consecutive product steps. The algorithm will search for a sequence of values $\alpha_{ij} \geq 1$ and changes these values to the length of the sequence. This is an example of a matrix constructed by a product agent with the products steps $< \sigma_1, \sigma_2, \sigma_3 >$ and 4 equiplets in the grid:

	e_1	e_2	e_3	e_4
σ_1	2	1	1	0
σ_2	2	0	0	2
σ_3	0	1	1	2

Table 4.1: Scheduling matrix

The next step is to take workload into account to balance the work among the equiplets. A high load of an equiplet decreases the associated values in the matrix. The load is a value between 0 and 1. The load μ_e of an equiplet e is as followed defined:

$$\mu_e = 1 - \frac{S_r}{S_t}$$

where S_r is time scheduled and S_t the timespan. The timespan is given by the product agent. This will be the product deadline as this period is of interest to the product agent.

The product agent chooses a production path from the matrix with the highest score. The agent will request the first possible timeslots at the equiplet. The process of planning, optimizing, and scheduling is an atomic action of the product agent. One product agent can schedule its product steps at the same time. It is shown that a high load of the equiplets in the grid can be achieved if the total production time, i.e. the total time taken by all production steps, is shorter than the time between the release time and the deadline of the product agent.

To summarize, the algorithm identifies three values that affect the score of an equiplet: whether the equiplet can perform the product step, if the equiplet can manufacture multiple consecutive product steps, and the load of the equiplet.

The schedule algorithm has a few drawbacks. It does not take processing or travel times into account. Further, the algorithm takes gaps in the schedules not into account. This can lead to inefficient situations:

Example 1. Suppose there is a situation where a product agent must choose between two equiplets with the same capability to plan its product step. Figure 4.3 shows the planning of the two equiplets in this situation. The product agent chooses to plan at the equiplet e_1 as this has a lower load. However, the product step can be processed sooner at equiplet e_2 with a higher load.

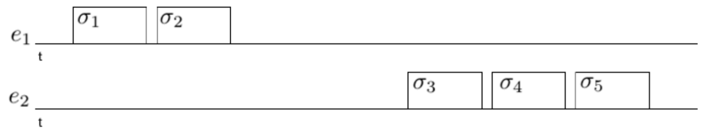


Figure 4.3: Equiplet schedules

A shortest path algorithm can be used for finding the equiplets with most favorable load to execute the product steps. The pair (G, c) consists of a directed graph G and a cost function $c : A \to \mathbb{Z}$. A graph is $G = (N, A)$, where N are the nodes and A the arcs between the nodes. The set of nodes consists of the set product steps and an equiplet that is capable to execute the product step. There is an arc between these nodes if the equiplet is capable to execute the next product step. A source node is added with an arc to each node associated with the first product step. Similar, a sink node is added with an arc from each node associated with the last product step. The shortest path from the source node to the sink node gives a production path. Figure 4.4 shows a graph where a product has three product steps to schedule. For the first and third product step there are 3 options and 2 options for the second product step.

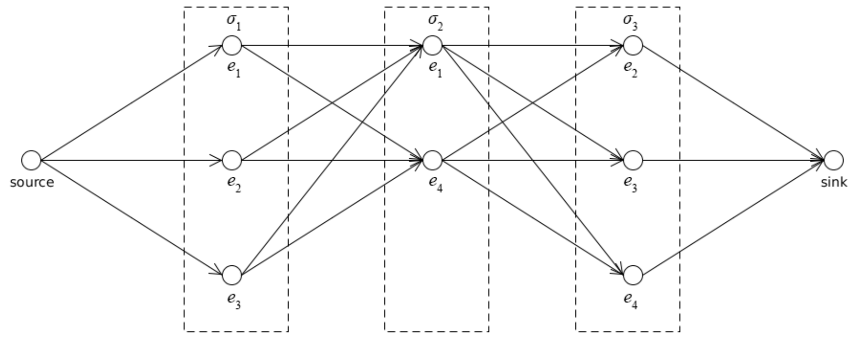


Figure 4.4: Scheduling Graph

This graph can be solved with single-source shortest path algorithm such as Dijkstra [34]. Depending on the cost function different strategies can be chosen. Similar to the previous described scheduling algorithm the emphasis could be on balancing the workload of equiplets. This would give the following cost function:

$$c(u, v) = \mu_v$$

Compared to the matrix approach, the production path constructed by the algorithm is less favorable to equiplets capable to execute multiple consecutive product steps. Equiplets with multiples capabilities will be preferred over equiplet with less capabilities. Even when there is little room in a equiplets' schedule, i.e. the equiplet has a high load. If an equiplet is preferred for the first product step in the sequence it is preferred for the second product step.

Alternatively, the cost function could depend on the first time a equiplet is available. This would give production paths similar to an earliest due date (EDD) algorithm.

$$c(u, v) = \frac{d - a_v}{d - t}$$

where d is the deadline of the product, a_v is the first available time for the product step, and t is the current time. The first available time of node v depends on the finish time of the previous product step, the travel time between the equiplets associated with node u and v, and the available time in the equiplets' schedule.

As the cost to a node depends on the equiplets associated with the previous nodes in the path leads this to a problem. Suppose a product agent needs to schedule the product steps σ_1 at equiplets e_1 or e_2 and σ_2 at equiplet e_3. In figure 4.5 the schedules of the three equiplets can be seen with the possibilities for the two product steps. The product step σ_1 will be executed earlier at e_1 than e_2, however, the production path with e_2 is faster. This is because the travel time from e_1 to e_3 takes so long that the product step will be scheduled behind the already scheduled product step.

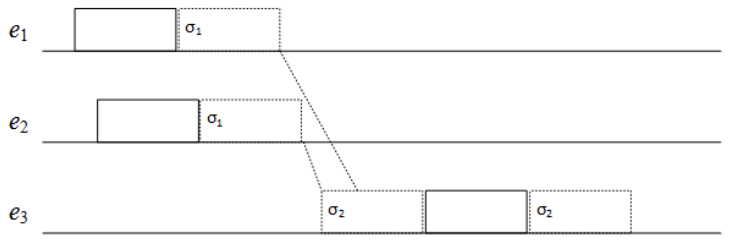


Figure 4.5: Scheduling Problem

To calculate the path with the lowest cost an algorithm will add a node to a path that has the lowest cost until the sink node is reached. By adding each iteration a node the cost of the path increases, so if the sink node is reached the best production path is found. The algorithm shown below works as follows: a worklist *paths* is initialized; for each neighbor of the last node in the best the path, a new path is added to the worklist. The algorithm uses a list of paths sorted on score, such that if the first path in the worklist contains the destination than there is no better solution.

Algorithm 2 Scheduling Product Steps

```
$paths := \{\}$    ▷ a sorted (on score) set of processed subpaths

for all $(e, load, duration, available) \in options(\sigma_1)$ do
    $arrival := this_{created} + travelTime(this_{pos}, e_{pos})$    ▷ arrival by equiplet
    $time := max(arrival, available)$    ▷ first possibility an equiplet can be scheduled
    $score :=$ SCORE(time, load)    ▷ score of path
    $path := [Node(e, time, duration)]$
    $paths.append(path, score)$    ▷ add path to processing paths
end for
while $paths$ is not empty do
    $path := paths.first();$    ▷ get and remove the path with the best score
    $node := path.last();$

    if $node = destination$ then    ▷ best path reaches the destination
        return $path$
    end if
                                    ▷ add all neighbors to new subpath
    for all $neighbor \in neighbors(node)$ do
        $travel := travelTime(node_{equiplt}, neighbor_{equiplet})$
        $time := firstPossibleTime(node_{finished} + travel, neighbor_{available})$
        $score := path_{score} \times$SCORE(time, $neighbor_{load}$)    ▷ score of new path
        $path.append(Node(e, time, options[neighbor]_{duration}))$
        $paths.append(path, score)$
    end for
end while

function SCORE$(t, l)$
    return $\frac{this_{deadline} - t}{this_{deadline} - current\ time}$
end function
```

The product agent has also the option to balance the workload while trying to minimize its due date. This is a combination of the two cost functions:

$$c(u, v) = \mu_v \frac{d - a_v}{d - t}$$

4.5 Rescheduling

It would be ideal to predict the exact processing and travel times such that no conflicts arise in the schedules. In real situations there are always variances in the manufacturing process. Little is yet known about the fluctuation of executing a job on an equiplet. When the equiplet finishes a job earlier or on time nothing will go wrong. The equiplet can wait for the next job or start with the next job when it is ready. The latter is preferable as jobs are then more likely to finish on time. When a job is tardy such that the next job would be delayed, the product agent of the delayed job needs to be informed. There are a number of possibilities what could happen next:

- An approach would be to keep the order of jobs in the schedule, but ignore the start times. When a job is delayed, nothing would change in the schedule. This could decrease the gaps in the schedule, but conflicts could arise when delayed products have many equiplets left in their production paths. This results in a snowball effect as the products in the affecting equiplets schedules will also be delayed and so on.
- The equiplet shifts all the jobs until everything fits while remaining the order of jobs. This would compress the schedule such that there is an opportunity for better utilization, but also a higher risk of affecting more jobs when there is another tardy job. The equiplet should inform the product agents of the jobs that are shifted. When there is a tight schedule this would mean informing a lot of product agents. The product agents should in turn inform the equiplet agents. This approach would lead to a snowball of communications.
- If the product agent discovers that a product step did not start on the agreed time, it can take his loss and reschedule his remaining product steps. This approach would not affect the scheduled jobs of other agents. When the product agent reschedules his product steps, it naturally takes as much as possible released time slots back. Although, there is a good chance that there is only room at the end of the equiplets schedule. This could lead to products not meeting their deadlines.

Both the first and the third options are implemented. The product agent schedules a trigger to ensure its job has started before affecting other product steps in its production path. If the triggers fire before the job has started the agent reschedules its remaining product steps. The procedure of rescheduling is as follows: the product agent will release the scheduled time slots and initiates a new schedule behavior to schedule its remaining product steps.

4.6 Queue Jumping

Many times there are products waiting at equiplets ready to be processed while the equiplet is idle. This happens when the first product in the schedule has not yet arrived. The equiplet will be idle until the arrival of that product. The cause of delays range from processing times that are longer than expected to breakdowns and unexpected maintenance for equiplets. This causes an unwanted effect in all the scheduled products behind the delayed product.

Queue Jumping is a procedure where products that are available (i.e. have arrived) at the equiplet can start before they are scheduled in case the next scheduled product has not yet arrived. The product can jump q places ahead in the schedule of the equiplet. It gives products that have already arrived priority over products that are scheduled to start next, but have not arrived in time to start. The product that is too late is placed behind the current product that starts. This might result in some products not completing within their expected completion time.

4.7 Reconfiguration

In the grid manufacturing there are no assumptions made about the products that will be manufactured in the future. Although, the configuration of equiplets could be optimized for the products to be made as there is a relation between the available resources and the product demand. The product demand is subject to variance in an agile environment. The production needs to adapt quickly to the market demand. The equiplets are easy to reconfigure to reduce downtime of the equipment. The reconfiguration procedure of an equiplet agent consists of de-registering its services with the DF and finishing the product steps in its schedule. After reconfiguration, when the equiplet is functional, the agent registers the changed services with the DF and is able accept work from product agents.

In the grid an entity is made to suggest that equiplets change their capability. The entity will suggest capability changes based on results of a simple algorithm. After a certain time has elapsed the simulation enables this algorithm to change an equiplet if necessary. The algorithm identifies the equiplet with the highest utilization and for each capability two equiplets with the lowest utilization. The equiplet with the lowest utilization will be reconfigured to the capability of equiplet with the highest utilization. The two lowest utilizations combined need to be lower than a threshold to prevent equiplets changing their capability when it is not beneficial to the grid. The two lowest equiplet are taken into account as an equiplet can have an unfortunate schedule such that the load drops below the threshold. If the utilizations together were less than 100% the work could have been done by one equiplet. To prevent equiplets changing capabilities too much the threshold is set on 110%. The grid will always keep one of each capability to prevent that certain products no longer could be manufactured after reconfiguration.

Chapter 5: Simulation

A discrete-event simulation model is developed for the purpose of investigating the presented model. After each event the state of the system changes to a new situation. The simulation is structured in the following way: initiation of the simulation, while the simulation is not ended: get the next event from the event stack, advance the simulation time, handle the event, add new events to the event stack, and update statistics.

In the simulation a discrete time is used. These time steps are relative to the start of the simulation. The simulation time start at zero and each iteration after an event the simulation jumps to the next discrete time step.

5.1 Simulation Model

Eight different events are generated: *product* creation, product *arrived* at equiplet, product *started*, equiplet *finished* with a job, equiplet *breakdown*, equiplet is *repaired*, *reconfigured*, and simulation is *done*.

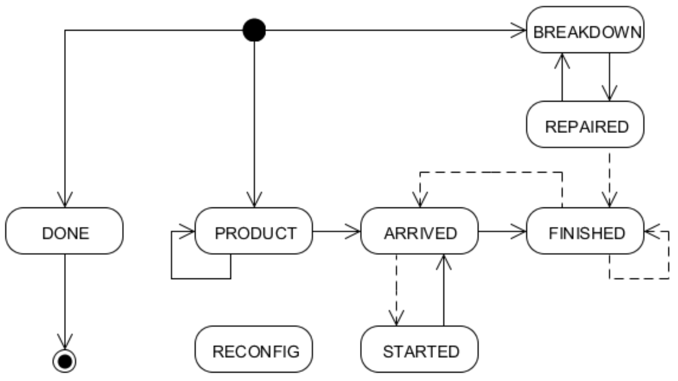


Figure 5.1: Simulation event graph

In figure 5.1 the events can be seen in an event graph. The graph shows an edge from the black dot to the events that are on the event stack at the start of the simulation and an edge to the circled dot for the event which causes the simulation to end. There is an edge between events if it causes the scheduling of an event. The dashed edge indicates that the scheduling of the event is optional, i.e. depends on certain conditions.

- A *product* creation event triggers the creation of a product agent with a certain set of product steps and a deadline. When the agent manages to schedule his product steps, a product arrived event is added to the event stack.
- A product *arrived* event will trigger when a product arrives at an equiplet where after the product agent informs the equiplet agent that it has arrived.
- A *started* event is triggered at the latest time that a product should have been started. If this is not the case the product knows that it has to reschedule the remaining product steps.
- An equiplet is *finished* with a product step. He informs the corresponding product agent which results in, if needed, the simulation schedules a product arrived event.
- The simulation gets a *breakdown* event when an equiplet breakdown or is in need of maintenance.
- The simulation gets a *repaired* event when the equiplet can continue his activities. After an equiplet is repaired a new breakdown event is scheduled.
- A *reconfig* event is scheduled when the equiplet can be shut down and reconfigured into a new configuration. The event will trigger when the capabilities of the equiplet are changed and can register these services with the DF agent.
- The *done* event will end the simulation. The required procedures, such as saving statistics and terminating the living agents, are executed. This event can be added at the beginning of the simulation to ensure a certain run time or can be added after a number of products are created.

The equiplet can be in the states idle, busy or error. The equiplet has, next to these states, three additional simulation states:

- *error finished*: when the equiplet should have finished with the job, but has broken down in the meantime. When the simulation receives the *finished* event, the equiplet still needs to take the time that it was broken to finish the job.
- *error repaired*: when the equiplet is repaired but the simulation does not know when the job would have been finished. When the simulation receives the *finished* event it can be reset.

- *error ready*: when the equiplet would have started with executing a job of the product that arrives, but is broken down.

The simulation requires the distinction between these states to identify the correct handling of an event. It is for the simulation not possible to look into the future, i.e. look at or edit the event stack. So certain events need to be disregarded or handled differently. Equiplets need to make a distinction between the state busy, when they are normally busy with a job or when they continue with a job after been broken. The same goes for the distinction in the error state, when the equiplet would have been finished or when a product arrives while being broken. In the figure 5.2 the relation between the 6 states are shown. The three additional states are specializations of the equiplet states.

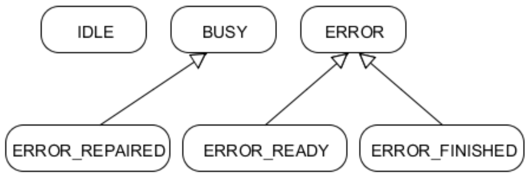
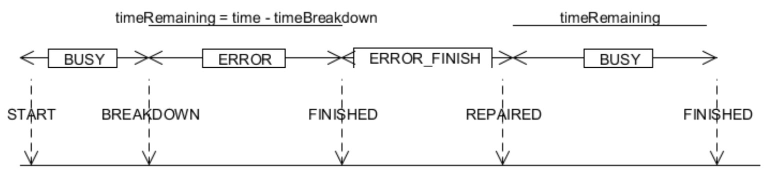
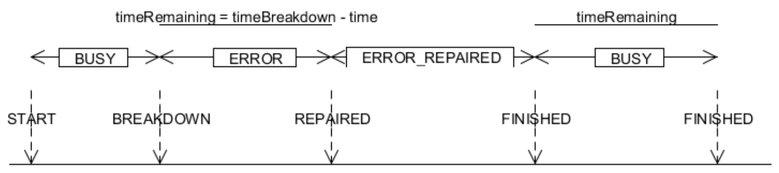


Figure 5.2: Equiplet states

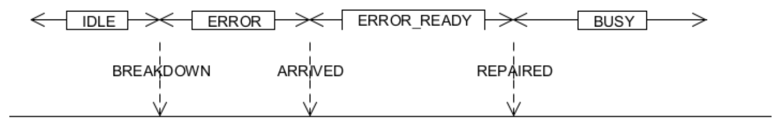
To get more insight into the state transitions the following graph is made, shown in figure 5.3. The graph focusses on the transitions when equiplets breakdown. The first two cases are when the equiplet is executing a job and breaks down. The finished event is generated by the simulation that presumes the equiplet will not breakdown. The distinction between the two first cases is the time the equiplet is broken, so whether the equiplet is repaired before or after the finished event fires. The finished event is postponed with the time the equiplet was in the error state. The third case shows that the simulation can add a finished event to the event stack as the equiplet directly continues after being repaired. The last cases are when the equiplet is idle, breaks down and remains idle.



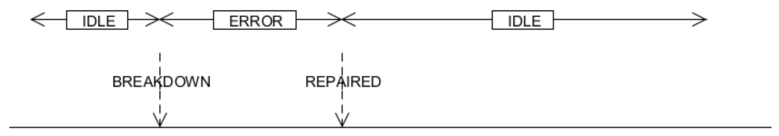
(a) equiplet should have finished during a breakdown



(b) equiplet should have finished after a breakdown



(c) product arrived during breakdown

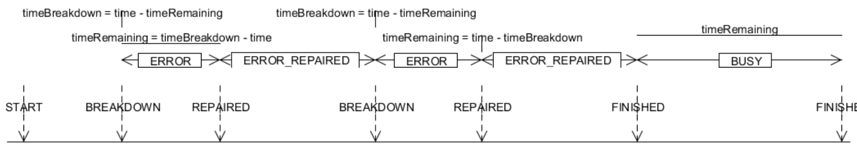


(d) equiplet breaksdown while idle

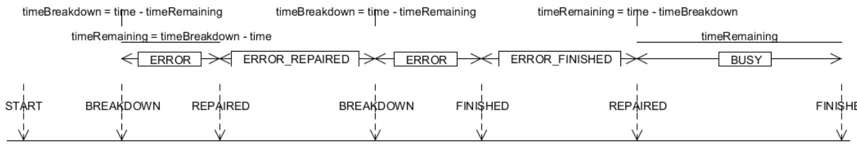
Figure 5.3: Equiplet timeline

It is possible that the equiplet breaks multiple times before its finishes a job, although this is not likely in real situations. While an equiplet is busy with the remainder of a job, after being repaired, the state of the equiplet is error repaired.

When the duration of a job is longer than the uptime between breakdowns, the equiplets state transitions from error repaired to error which changes the behavior when the repair is finished. An example of this would be a 3D printer that requires multiple refills, i.e. maintenance, before completion. After being repaired the time remaining will be the current time subtracted with the breakdown time of the previous breakdown. The time remaining becomes the sum of the equiplets downtime. The two cases in figure 5.4 show the method that can work for two or more breakdowns.


timeBreakdown = time - timeRemaining
timeBreakdown = time - timeRemaining
timeRemaining = timeBreakdown - time
timeRemaining = time - timeBreakdown
timeRemaining
ERROR
ERROR_REPAIRED
ERROR
ERROR_REPAIRED
BUSY
START
BREAKDOWN
REPAIRED
BREAKDOWN
REPAIRED
FINISHED
FINISHED

(a) equiplet should have finished during a second breakdown


timeBreakdown = time - timeRemaining
timeBreakdown = time - timeRemaining
timeRemaining = time - timeBreakdown
timeRemaining = timeBreakdown - time
timeRemaining
ERROR
ERROR_REPAIRED
ERROR
ERROR_FINISHED
BUSY
START
BREAKDOWN
REPAIRED
BREAKDOWN
FINISHED
REPAIRED
FINISHED

(b) equiplet should have finished after a second breakdown

Figure 5.4: Equiplet timeline

For the simulation, interfaces were made to simulate external and/or internal events incoming at agents. The simulation takes over tasks from for example the transport agent, seen in figure 3.3. A simplified class diagram can be seen in figure 5.5. The simulation will create the product agent and takes over the task from a transport agent to inform the product agent it has arrived at the equiplet. Similarly, the simulation fakes the notifications that a job is executed and malfunctions generated by the equiplets hardware.

The agents are built within the JADE framework [30]. As the communication between agents is asynchronous, a lock mechanism guarantees events are completely handled before the next event. This ensures the discrete property of the simulation. When a product agent is created and finished with his scheduling, the simulation gets a callback from the agent. Similarly, the product agent informs the simulation thread when it is travelling, starts with processing, and finishes processing.

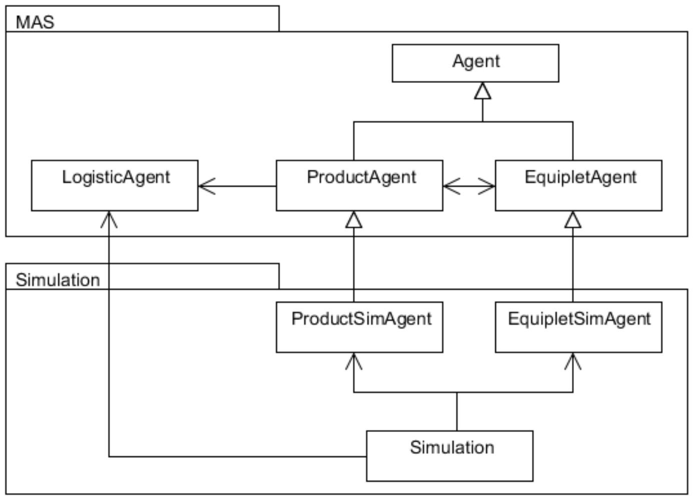


Figure 5.5: Simulation UML

The interfaces provide three methods to the simulation for simulating events: *notifyJobFinished*, *notifyBreakdown*, and *notifyRepaired*. For the equiplet the state transitions caused by these are shown in figure 5.6. The communication between the product and equiplet agents are used to inform the start of manufacturing a product step and a product has arrived at the equiplet. These events are handled by *notifyProductArrived* and *informProductProcessing*.

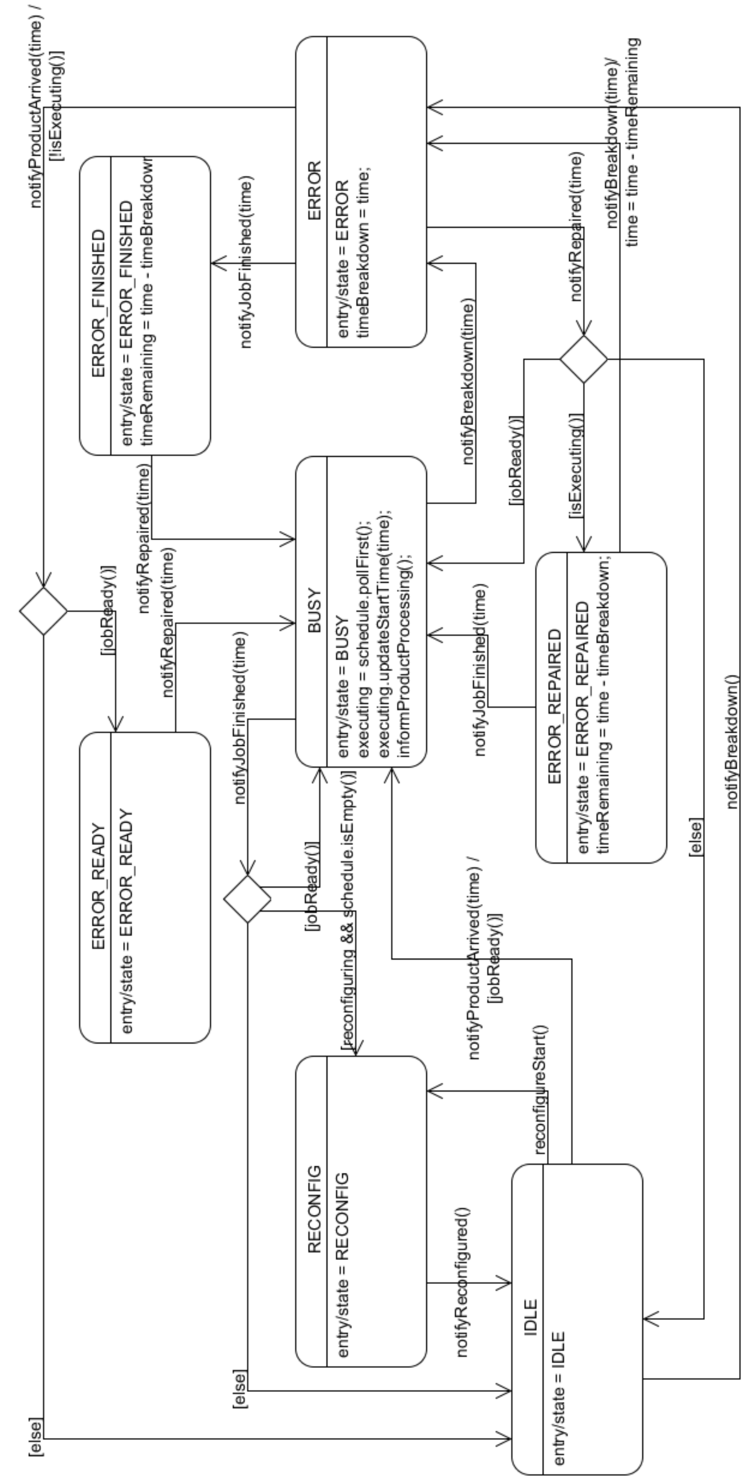


Figure 5.6: Equiplet STD

5.1.1 Product Generation

The time between product arrivals, i.e. product events, is chosen to have an exponential distribution. The exponential distribution is common and powerful modeling tool because of its lack of memory [35]. Also, the lack of memory allows a steady-state modeling approach to be used. Normally, distributions are derived from empirical data which is in this project still unknown. The inter arrival times of products follow a Poisson distribution with a mean depending on the target utilization of the grid:

$$\rho = \frac{E(S)}{E(A)}$$

where the utilization ρ is the expected service time $E(S)$ divided by the expected inter arrival time $E(A)$.

A product is spawned by the simulation with a certain amount of product steps. Depending on the simulation configuration, the number of product steps can be either deterministic or with an average and minimum number of product steps. The utilization is the given target load for each simulation run. The number of products that are produced is dependent on the average load of the equiplets.

$$E(S) = \frac{1}{|P_i| \times |E|} \sum_{e \in E} \frac{1}{|S_e|} \sum_{s \in S_e} E(s)$$

The formula above calculates the expected average service time for a grid, where $|P_i|$ is the number of product steps, $|E|$ is the number of Equiplets in the grid and $E(s)$ is the given expected processing time of a service s.

5.1.2 Breakdowns

It would be ideal to have production equipment that always works perfectly and never need maintenance or breaks down. In the simulation no distinction is made between an equiplet breaking down or the equiplet needing maintenance. The reason makes no difference for whether the equiplet can continue production. The equiplet receives from the simulation the information that it is broken. Normally, a failure would be detected by an internal monitoring behavior or a signal from a hardware controller. After breakdown of the equiplet the simulation adds a repair event to the event stack. For each equiplets an average breakdown time and repair time can be set. These two variables will, similar to product generation, have an exponential distribution.

5.1.3 Steady State

Each simulation will run for 1000000 time steps. This period will be sufficient for each equiplet to breakdown often enough and produce a certain amount of products to simulate a long period of manufacturing. Little's law states variations such as product arrival rate, processing time, and other variables will converge to the average as time increases. The runtime chosen is similar to previous simulations in relation of the largest variable time compared to the runtime i.e. processing time versus run time. A certain time is required until the production grid runs at a reasonable capacity when the simulation starts. When there is little difference between the utilization and the capacity to the grid it takes a long time until a steady state emerges. To speed up the process a time period is taken where products are spawned at a faster rate to bring the grid up and running. Statistics are only collected from this point and onwards.

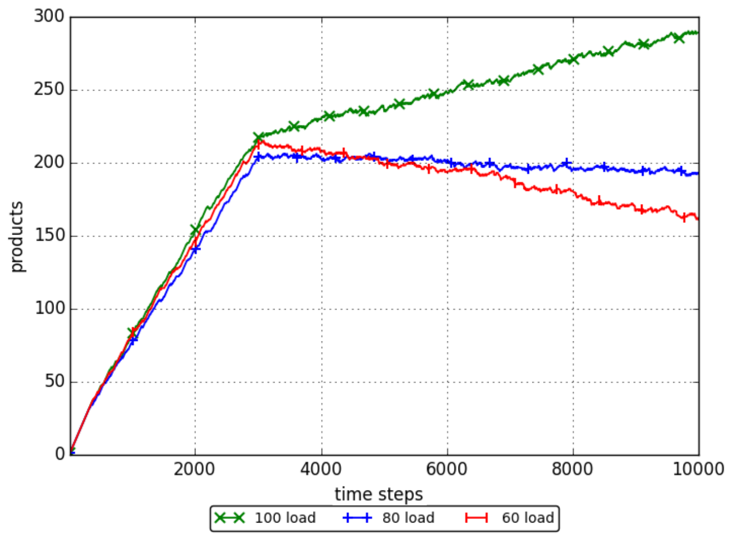


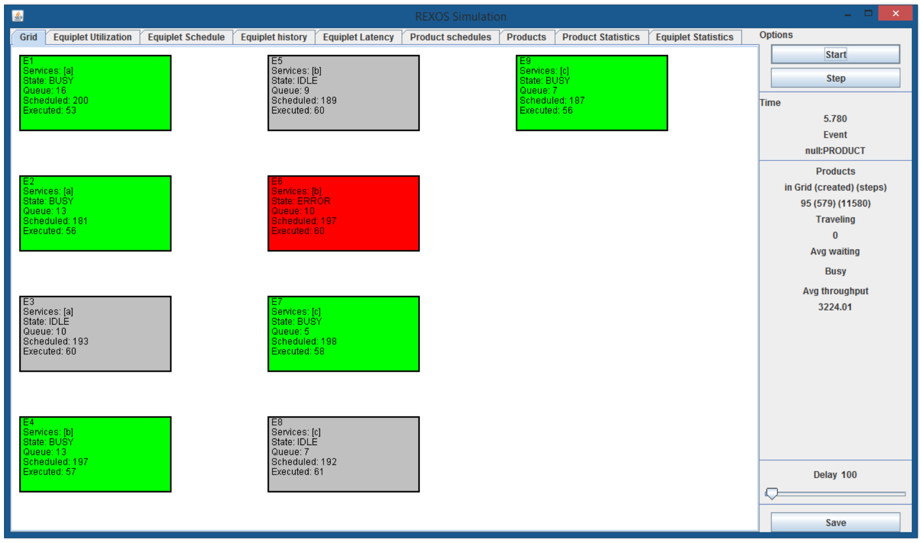
Figure 5.7: Simulation warm-up period

In the warm-up period a certain number of products will be spawned such that thereafter a plausible number of products in the system exist. A test case is used to validate the choice of a warm-up period of 3000. Three runs are done with a target utilization of 100%, 80%, and 60%. The data in figure 5.7 shows after the warm-up period around 200 products were created. Afterwards this number will decrease with an utilization of 60% and increase with an utilization of 100%. With an utilization of 80% the products in the system will remain roughly stable. The number of products will eventually increase to above 300 products.

5.1.4 Verification and Validation

The verification and validation comes down to the “process of determining whether a simulation model is an accurate representation of the system for the particular objectives of the study.” To ensure the correct programming and implementation of a model the following steps are used: 1) Debugging the program; 2) Checking the internal logic of the model; 3) Comparing the model output with the information obtained from previous simulation results.

In figure 5.8 a graphical interface for the simulation is shown. This enables to see the current state of the grid during the simulation. In the interface the equiplets can be seen with different colors corresponding to their equiplet state. Each box represents an equiplet containing the supported services, the queue, i.e. the products arrived by the equiplet, product steps scheduled and product steps executed. The simulation can be paused and run step by step. Furthermore, a delay between events can be set to view the effects and progress more clearly. Statistics can be plot during the simulation. For example, a Gantt chart of the schedule and history can be seen. In the right panel there are state and statistical information like time, the created products with the number of product steps, and products traveling.



```
E1
Services: [a]
State: BUSY
Queue: 16
Scheduled: 200
Executed: 53

E5
Services: [b]
State: IDLE
Queue: 9
Scheduled: 189
Executed: 60

E9
Services: [c]
State: BUSY
Queue: 7
Scheduled: 187
Executed: 56

E2
Services: [a]
State: BUSY
Queue: 13
Scheduled: 181
Executed: 56

E6
Services: [b]
State: ERROR
Queue: 10
Scheduled: 197
Executed: 60

E3
Services: [a]
State: IDLE
Queue: 10
Scheduled: 193
Executed: 60

E7
Services: [c]
State: BUSY
Queue: 5
Scheduled: 198
Executed: 58

E4
Services: [b]
State: BUSY
Queue: 13
Scheduled: 197
Executed: 57

E8
Services: [c]
State: IDLE
Queue: 7
Scheduled: 192
Executed: 61
```

```
Time
5.780
Event
null:PRODUCT
Products
in Grid (created) (steps)
95 (579) (11580)
Traveling
0
Avg waiting
Busy
Avg throughput
3224.01
Delay 100
```

Figure 5.8: Simulation GUI

The settings has a verbosity level which adjusts the level of output in order to view 0) nothing during the simulation, 1) only debug information, 2) only the gui, 3) the gui with debug and state information, and 4) the gui with more debug such as scheduling information. However, the simulation will run slower by showing the gui and debug information.

For different parts of the equiplet agent unit tests were written. Functionalities like availability and load give correct results at first sight while in later stages of the simulation the production of products can come to a standstill by inaccurate implementation of these functionalities. These lead to incorrect information for the product agent and eventually to miscommunication or infeasible equiplet schedules.

When the grid is running live, some situations can occur that are impossible in the discreet event simulation. Due to the discreet nature of the simulation there cannot be two events handled simultaneously. Vice versa, the simulation interfaces of the agents cannot receive some events when they are in certain states. For example, an idle equiplet cannot receive an event that the product step is finished. To find these kind of problems there will be an exception thrown to stop the simulation, which is not desirable in a real situation. Usually, these are only symptoms of the problem but through tracing debug information the source can be discovered.

For verification of the simulation, runs can be done with settings where the results are known beforehand. For example, a simulation run without breakdowns or stochastic processing times would result in product agents that are always at the equiplets on time and equiplets finish the product steps on time, so there is no latency and products finish always within their deadline.

5.1.5 Assumptions

Still much is unknown about the implementation and the use of a production grid. Additionally, the grid should perform as well as possible independent of the available hardware. For this research certain simulation settings are chosen or omitted as they are still unknown, have no effect on the result, or are beyond the scope of this research. The following assumptions are made about the grid:

- It is assumed that the distributions of the processing times are unknown by the system. The actual processing time of job becomes only known upon completion. An equiplet agent only knows the average time it takes to perform a job and not the exact time a priori.

- The sequence of product steps needing to be performed to make a product is important i.e. a product must perform σ_i before σ_j if $i < j$.

- No distinction is made between the reasons of downtime of an equiplet. The reason could be the equiplet requires maintenance, a failure in one of its modules, or other reasons the equiplet cannot complete a job. Also, no distinction is made between maintenance or repair time. The averages of both breakdown and repair time are given for each equiplet in the simulation.

- The chance the equiplet breaks down is not affected by the state of the equiplet i.e. whether an equiplet breaks down does not depend on the time it is idle or busy. This can be different in real situations as, for example, a 3D printer needs only maintenance after performing a certain number of product steps. Contrary, there is equipment that does not depend on whether they manufacturing a product step or is idle.

- There is enough room for every product at an equiplet. The maximum size of the queue depends on the room available at the equiplets and the size of a waiting product. Many factors play a role here; size of the products and parts, size equiplets, production facility, transportation method, and other choices. All is related to the implementation of the production grid.

- Travelling between equiplets always succeeds and takes a certain time per distance to be covered. Whether paths to equiplets are free, transport units have enough room to avoid each other, or other transport issues are not taken into account.

5.2 Simulation Experiments

First we look at the scheduling results compared with the previous experiments. The difference between the scheduling methods and objectives are investigated. Further, the impact of disturbances and strategies to counter these effects are tested. Finally, the impact of reconfiguration during the manufacturing process is examined.

All results of each experiment are based on the average of 10 runs. Each product has a deadline of 10000 in which it is required to plan its completion. When the product is not able to schedule its product steps it is marked as *failed* and does not enter the grid, i.e. does not start with processing. Products that start with processing but are delayed by variable processing times or other manufacturing disturbances will still try to complete. These products are marked as *overdue*. The *production time* of a product is the time it has taken to manufacture the product.

To verify the simulation and the schedule algorithm the aim is to achieve a load i.e. $\rho = 0.8$ without products failing to schedule. This 80% was discovered by van Moergestel [24] to be the limit in which all product production is feasible. In this case there is a grid with 9 equiplet having each a unique capability. The results in figure 5.9 show the grid has an average load of almost 80%. The difference can be explained by the randomness of the product steps of a product. As each product consists of different product steps some equiplets will be used more, up to a point that products fail to schedule, despite that the average load lower is than the feasible target load. Changing the grid to have 3 capabilities gives the product less variety of product steps and for each step two alternative equiplets. This will result in a load that converges to the 80%.

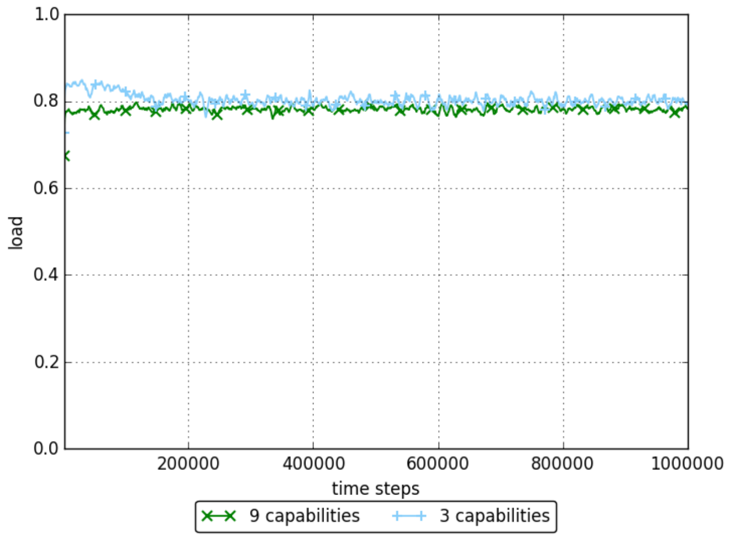


Figure 5.9: Results of load 80%

A slightly different approach, compared to previous simulations, is taken when products fail to schedule. For the investigation of the grid utilization, when the product fails to schedule it will not enter the grid and will be marked as failed to schedule. As it is unknown whether the utilization is feasible, the products will not coordinate with each other to search for a complete feasible schedule. When the target utilization is below a certain threshold, coordination is not necessary as all products are able to schedule. When the target utilization is too high, coordination to search for a feasible schedule will keep occurring each time a product enters the grid. This results in a snowball effect of communication and eventually not leads to improvement.

	Finished	Failed	Production Time	Load
Matrix	36055.8	0	1342.06	0.8034

Table 5.1: Matrix scheduling results with $\rho = 0.8$

Table 5.2 shows the average result of 10 runs with a range of $0.7 \leq \rho \leq 0.9$. The results confirm the threshold of 80% load as all products succeed to schedule with $\rho = 0.8$, but there 0.8% of the products fail to schedule with $\rho = 0.85$.

It is interesting to the see the average production time increases rapidly between $\rho = 0.8$ and $\rho = 0.85$. The grid is at its maximum production between these utilizations. The average production time will near the deadline of the products, which is 10000 time steps. As the grid becomes overcrowded the products can only just schedule within their deadline. The moment when the average production time comes close to the deadline products will fail to

ρ	Finished	Failed	Production Time	Load
0.70	31592	0	696,43	0,704
0.75	33830	0	859,79	0,754
0.80	36056	0	1342,06	0,803
0.85	37663	313	8967,07	0,840
0.90	37733	2481	9624,30	0,841

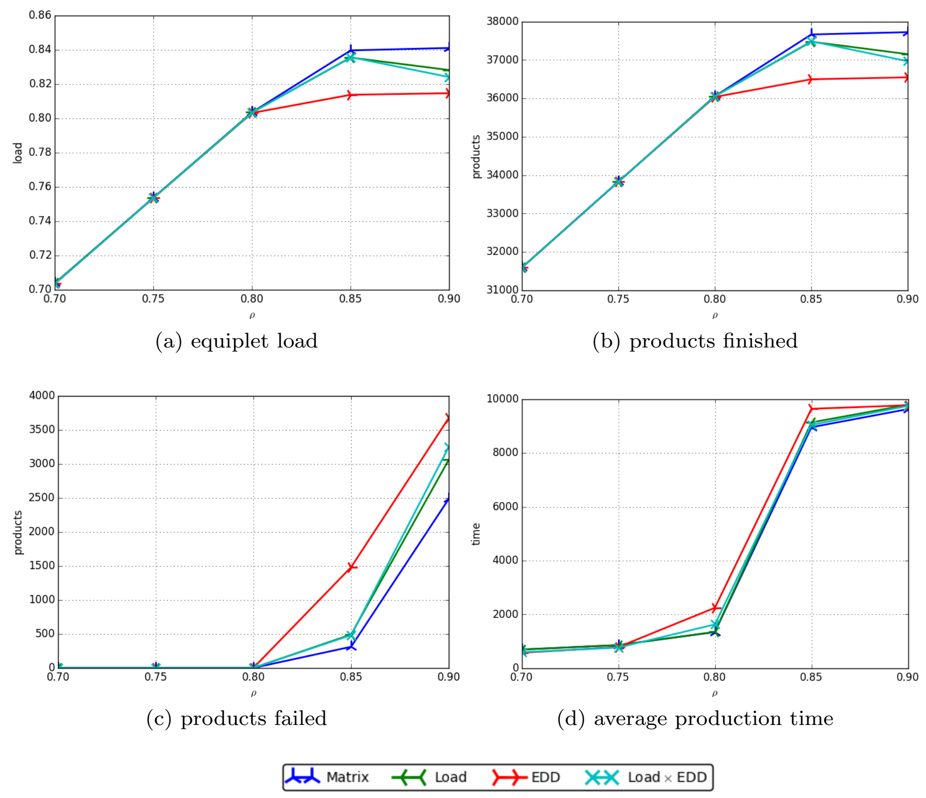
Table 5.2: Matrix scheduling result with $0.7 \leq \rho \leq 0.9$

schedule. Increasing the utilization above $\rho = 0.85$, would only yield a small increase of finished products but more failed products.

5.2.1 Scheduling

To see whether different scheduling techniques would affect the performance of the grid each scheduling algorithm is tested between a range of $0.7 \leq \rho \leq 0.9$. The following scheduling techniques are investigated: Matrix, Load, EDD, and Load $\times$ EDD. The simulation is initialized with a grid of 9 equiplets with 3 unique capabilities taken each 20 time steps to complete. A product consists of 10, randomly chosen between these capabilities, product steps and a deadline of 10000 time steps after the creation.

To see differences between the scheduling algorithms the results are plot in figure 5.10. For the four metrics: equiplet load, products finished, products failed, and average production time of a product a graph is plot with the results of each algorithm. Looking at the result, when $\rho \leq 0.8$ all scheduling techniques perform similar. All products are able to schedule and complete in reasonable similar time. The first algorithm that has difficulty is EDD. There is a slightly longer production time at $\rho = 0.8$ and from that point the maximum production is reached such that products will fail and the load does not increase linear. Matrix scheduling performs overall the best. Products taking a more selfish approach in scheduling their products steps with EDD scheduling, i.e. try to finish as quickly as possible, does not benefit the overall production of the grid. Balancing the load between equiplets results in an overall better performance.

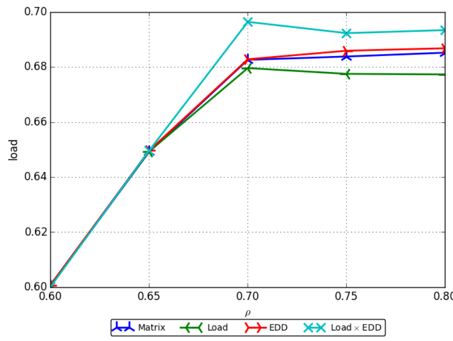


(a) equiplet load
(b) products finished
(c) products failed
(d) average production time

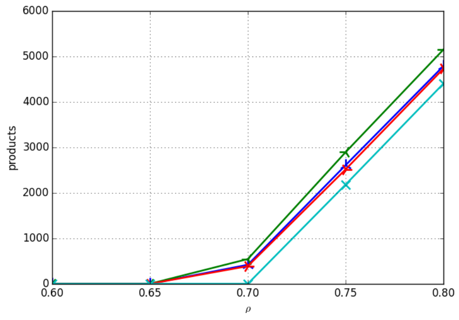
Figure 5.10: Deterministic scheduling for $0.7 \leq \rho \leq 0.9$

From the scheduling results different patterns can be seen. The four metrics are heavily correlated with each other. The equiplet load shows the same results as the products finished. The utilization matches the average equiplet load in a simulation run, if it is not for the products that are still in production. Further, products start to fail when the capacity of the grid is reached. From that point on the average production time increases rapidly until the system's capacity is reached and the average production time comes close to the products' deadline.

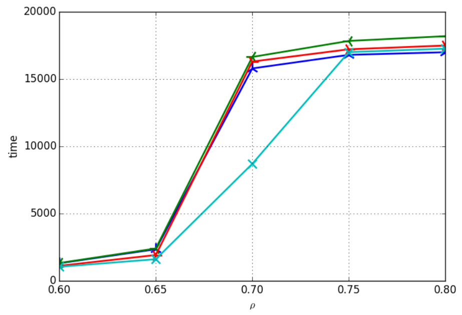
To show a more realistic case, the effects of stochastic processing times are taken into account. The same set-up is used, but products are able to reschedule remaining product step. The simulations are run between a range of $0.6 \leq \rho \leq 0.8$. The results in figure 5.11 show an overall reduction of the production. This is due to the products steps taking longer than expected up to the point that products reschedule their remaining products steps as the production path becomes infeasible.



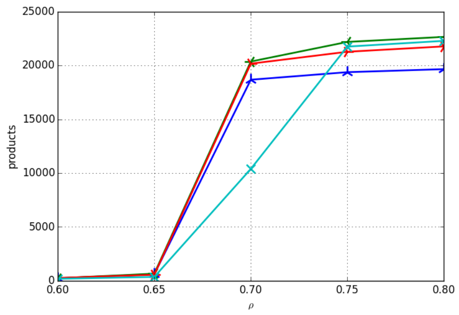
(a) equiplet load



(b) products failed



(c) average production time



(d) products overdue

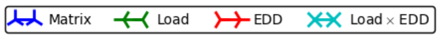


Figure 5.11: Stochastic scheduling for $0.6 \leq \rho \leq 0.8$

With variable processing times there is a reduction of 15% products produced at $\rho = 0.8$. The average production time even increases above the deadline. Due to products able to reschedule with an increased deadline also more products are overdue. The results show the products' average production time follows a similar pattern as the overdue products.

With stochastic processing times the Load × EDD algorithm yield the most manufactured products. The matrix algorithm gives the least overdue products.

Looking at both the deterministic and stochastic scheduling results, it becomes clear that the schedule algorithms have an optimum production. From that point on trying to make more products does not yield more finished products. Moreover, attempting to produce slightly more than the optimum would decrease production. With deterministic processing times the point in time that products start to fail are discrete.

Comparing the computation time of the 3 algorithms with that from the Matrix algorithm, the latter calculates a solution in a faster and less varying time. The algorithm iterates twice over the matrix; first to construct the matrix and then calculating the production path. The other approaches would in the worst case compute all the production paths. While load of the equiplets increases the algorithm takes longer to compute. The product agent has to consider the algorithm to use. Whether a longer computation time outweighs a shorter production time depends on the application of the production grid.

5.2.2 Queue Jumping

Variable processing times reduces the effectiveness of the production grid. The grid will perform worse as products steps will not complete in time and delay all other products that scheduled behind the delayed product step. Queue jumping aims to counter the effects of these disturbances by allowing a product to jump q places in the queue. A product arriving at place less than q in the queue can start immediately with processing. The grid has the same set-up as previous simulations, i.e. of 9 equiplets with 3 capabilities.

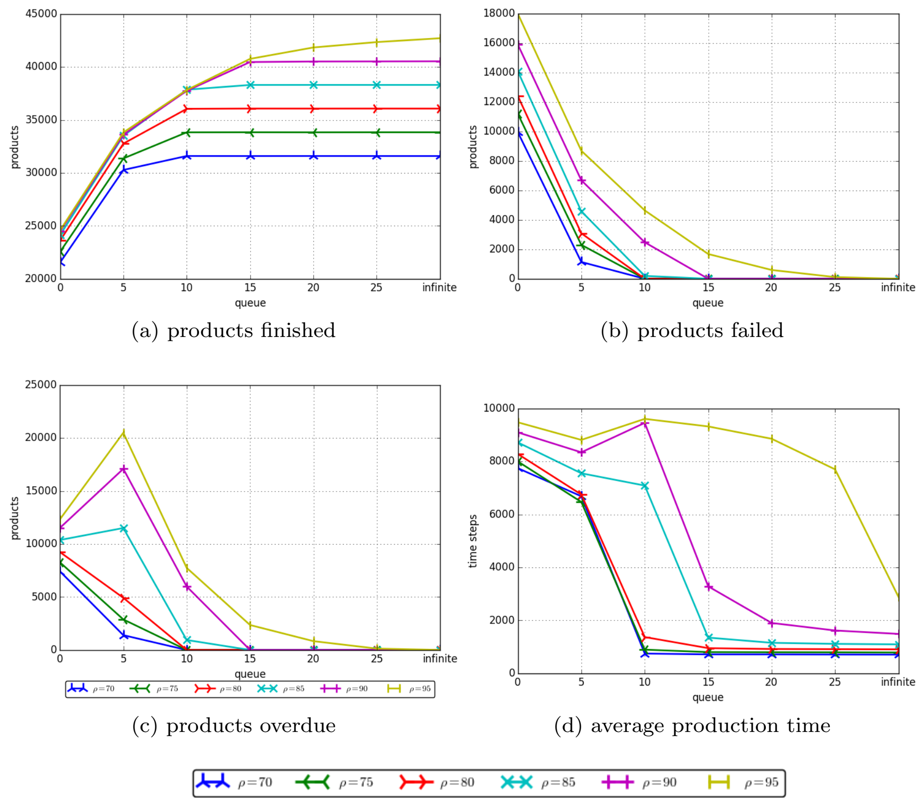


Figure 5.12: Queue jumping results

The results show a correlation between q and the products that fail to schedule or are overdue. More products are able to schedule the greater q is. With

$q = 15$ the load can be increased to 90% and with $q = 25$ almost to 95%. In the latter case, only around 100 products fail to schedule. With an infinite q all products successfully schedule and overdue products are reduced to 0.

Increasing q will have first a negative effect on the overdue products and average production time. With $q = 5$ there is a peak in overdue products when $\rho > 0.8$. For $\rho > 0.9$ the average production time will first decrease and then increase until $q > 10$ where after the time decreases. These two phenomena can be explained by the increase of products in the system and therefore more products are negatively affected by a queue jump. The greater q, the more products that would be affected by a queue jump fall within the threshold q and have therefore no delay. This increases the overall performance of the grid.

q	Finished	Failed	Overdue	Production Time	Load
infinite	43631	1033	1429	9067.33	0.985

Table 5.3: Result of queue jumping with $\rho = 1.00$

The results of $\rho = 0.95$ give wonder to what the result would be when trying to utilize the equiplets 100%. As table 5.3 show the result when $\rho = 1$. The production load could be increased to 0.985. However, the breakdown or need for maintenance prevents the production would be increased to 100%.

5.2.3 Reconfiguration

An aspect of agile manufacturing is the ability to adapt to the product demand. For testing the reconfiguration procedure the grid is initialized with a suboptimal equiplet set-up i.e. not suited production of the arriving products. The reconfiguration procedure reconfigures equiplets for a more optimal production grid set-up. Simulations are done to test the ability of the grid to reconfigure its equiplets. The set-up consists of 24 equiplets with 4 different capabilities. The set-up has an even distribution of capabilities. However, the time it takes to complete a product step is different. In this case the average production time for $\sigma_1 = 15$, $\sigma_2 = 15$, $\sigma_3 = 30$, and $\sigma_4 = 120$. The simulation will have $\rho = 0.8$ such that fewer products will fail to schedule while the grid adapts to the product demand.

	Finished	Failed	Production Time	Load
base	10637	10631	9546	0.353
reconfig	20636	617	5879	0.773

Table 5.4: Results of a non-reconfigurable vs. reconfigurable system

Table 5.4 shows that when producing a variety of products, reconfiguring the manufacturing machines proves to be more efficient. This was expected since that product steps take a different amount of time to complete, as such a bottleneck will likely occur, since systems cannot adapt to the demand. Figure

5.13 shows this result over time, clearly the reconfigurable systems adapt with the demand and saturates to a 77% load. This is a 94% increase in products finished. Drops in load with the reconfigurable systems are due to the changeover time when an equiplet has to be reconfigured to be able to provide another capability to the products.

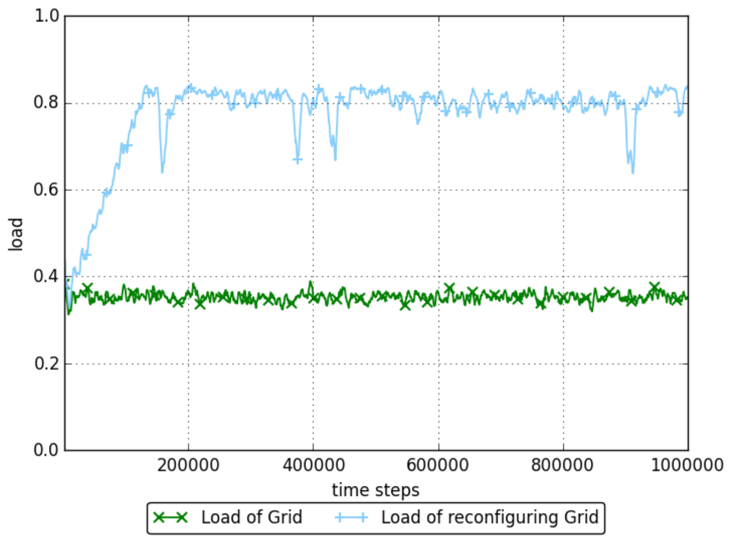


Figure 5.13: Results of one run with none vs. reconfigurable systems over time

Looking closely to figure 5.13, utilization drops occur after the grid has reached an optimum. The grid aims to improve further than the current optimum resulting in a capability change of an equiplet. However, this leads to a sub-optimum grid and changes back shortly after. It occurs when it is believed that changing the capability of an equiplet could result in a higher load of the equiplet. The cause is due to the randomness of the required capabilities for products such that predicting the future is too difficult.

Table 5.5 contains for three cases the result of a base setting, rescheduling turned on, and queue jumping (qj) with a queue of 20. The 3 cases are all with reconfiguration turned on, which is compared to when there are stochastic processing times and to the third when also the equiplets are possible to breakdown. As can be seen both stochastic processing times and breakdowns greatly affect the amount of products that can be finished. Together they halve the amount of products that can be completed in the base case. Both rescheduling and queue jumping counter these effects, almost doubling the amount of products that can be manufactured within the same time frame when these disturbances occur. The impact of stochastics and breakdowns prove to bring the products created under the level of the original base configuration (without reconfiguration). Rescheduling brings the load from 35% to 54% and queue jumping brings this up to 73%.

reconfig

	Finished	Failed	Overdue	Production Time	Load
base	20636	617	0	5878.9	0.773
resched.	20685	578	0	5721.7	0.775
qj	20924	397	101	3283.0	0.791

reconfig and stochastics

	Finished	Failed	Overdue	Production Time	Load
base	9693	11567	5879	11223.7	0.363
resched.	16380	4691	14063	17161.2	0.587
qj	20830	434	138	5507.6	0.763

reconfig, stochastics, and breakdowns

	Finished	Failed	Overdue	Production Time	Load
base	9295	11971	5759	11592.6	0.353
resched.	15089	6024	12608	16940.2	0.541
qj	19654	1540	607	8382.7	0.731

Table 5.5: Reconfiguration results

Looking at the average production time of products when there are disturbances, rescheduling products will increase the average processing time with around 300% compared to 200% without. This result in 85% finished products that are overdue. Without rescheduling the finished products that are overdue are around 60%. Although, the average production time of products that finishes later than planned is substantially less. With the deadline set to 10000 time steps, products that reschedule take around 70% longer than they are supposed to with the possibility of disturbances.

Chapter 6: Discussion

The results of the simulation depend on the assumptions and the chosen parameters for the examined cases. It is difficult to choose objective parameters to achieve unbiased results. The literature simplifies this problem by removing unknown factors. In the dynamic environment the processing times of product steps, downtime of equiplets, number and type of products are subject to variance. If the system is capable to efficiently handle a high level variance in the system the performance would be at least equal or better. Although, it is difficult to optimize the configuration of the manufacturing system to an optimum with the unknown configuration.

A stochastic model is used to validate more realistic cases with variable processing times and breakdowns. However, it is a challenge to provide distributions of the random variables without knowing the actual performance of the hardware. More research is required on the performance of the hardware. Empirical data should confirm the chosen distributions to improve the quality of the results. For the processing time an exponential distribution is used affecting the realization of the schedule. A distribution with less variance would have less impact on the realization of the schedule.

The comparison of the schedule algorithms depends less on the chosen parameters, although certain cases would give different results. The matrix scheduling approach would perform worse for cases where equiplets have multiple capabilities. The product agent would prefer equiplets with multiple capabilities rather than equiplets with one of these capabilities, as explained before. The challenge is to test fair cases without making specific cases for a certain result. For now the cases have equiplets with equal processing time and distributions. This affects the scheduling. Therefore, different cases could give different results. Furthermore, the equiplet set-up, number of capabilities in the grid, travel times, and other factors affecting the results. More cases should be tested to verify the current results.

The assumption that equiplets have room for infinite products is not realistic. The amount would depend on the application of the production grid, i.e. type of products, type of equiplets, manufacturing facility, and so on. It is expected that the utilization of equiplets would drop when the amount of products allowed in the grid is limited. The results show that the average production time of a product will decrease with lower target loads. So, certain optimizations should be applied depending on the objective of the manufacturing facility.

The results of reconfiguration show a clear improvement in performance. This shows the ability of the production grid to adapt to the current demand. The increase in performance depends on the start configuration. The results can vary greatly between different cases. The correlation between the production capabilities and the product demand depends on the available resources and the intensity of product orders, which are both unknown. The actual values of both factors make it difficult to give a prediction about the performance gain of reconfiguration.

Research in this area is more focused on describing an agent-based control system rather than testing the performance of the system. There is more research on scheduling in stochastic models. The literature identifies that agent technology is well suited to be used in manufacturing systems as agents can deal with uncertainties. However, research combining agent-based systems in stochastic environments is limited. Comparing the agent-based systems to conventional systems would imply building both systems and defining metrics for evaluation. The latter would be difficult in itself as these are usually tailored systems depending on the needs of the production company.

Chapter 7: Related Work

This research shows an implementation of an agent-based system for a manufacturing system based on the RMS and Agent Technology paradigms. The system combines a heterarchical architecture of MAS for controlling the manufacturing process and equiplet agents that have a hierarchical architecture. Manufacturing control systems are traditionally implemented using centralized or hierarchical control approaches. Unfortunately, no algorithm can foresee every possible failure of a highly complex system, neither can a strategy be deterministically designed for every situation [36]. This article from Duffie summarizes the advantages of scheduling in a heterarchical versus hierarchical manufacturing systems. Intelligent and distributed manufacturing control systems can be divided into four basic types of control architectures: centralized, hierarchical, modified hierarchical and heterarchical [37]. Leitão [7] and Trentesaux [6] present surveys of the intelligent and distributed manufacturing control systems using the emerging paradigms. Overall trends in various manufacturing sectors are the shift from hierarchical management structures to more levelled structures that reduces middle management, i.e. moving towards more modularity, autonomy, and self-sufficiency at the lowest possible levels [2].

Jarvenpaa targets more planning and adaptation than operational control of the system [38], [29]. Jarvenpaa describes a capability-based adaptation methodology to supports adaptation of production systems. During the study, the initial assumption that humans cannot be removed from the adaptation planning process was confirmed. A capability-based methodology is applied for combining modules to define the equiplets' capability. Equiplet agents can utilize these capabilities through the HAL and therefore making it easy to reconfigure an equiplet by adding or removing modules.

Simulation in manufacturing literature over the last decade is reviewed, analyzed and categorized by Negahban et al. [39]. The review identifies three general classes of manufacturing system design, manufacturing system operation, and simulation language/package development which can be subdivided based on application area. While research in this area becomes more available, these contributions are often informal and fragmented. However, simulation use, applications, and software have been addressed in several books [21] [22]. Komma et al [40] focuses on modeling different agents in a manufacturing domain in JADE, where machines and Automatic Guided Vehicles (AGV) are modeled as agents. The application leads to an agent-based simulation with a reactive agent architecture. Barbosa and Leitão [25] state that simulation is crucial in analyzing behavior during the design phase and present a simulation designed for studying agent-based control systems for deployment into a real operation. A holonic control system architecture for design and development of agile shop floor control systems is presented by Langer and Alting [41]. The concept combines the best features of hierarchical and heterarchical organization as it preserves the stability of a hierarchy while providing the dynamic flexibility of heterarchy. A multi-agent system for modelling and controlling the manufacturing process in a job shop system is developed and presented by Florea and Cristea [42].

By the lack of proven technologies, the industry is hesitated to adopt agent technology. By developing a simulation that extends the agent-based system, the research contributes with a practical implementation while enabling to test the performance of the system. Probably the first full-scale industrial agent-based manufacturing system brought into operation was Production 2000+ [43][44]. An overview of an agent-based solution developed by the Rockwell Automation company for the purpose of industrial control is given by Marik et al. [45]. Maturana et al. [46] presents an agent-based control system that has been implemented for a chilled-water system and a heating, ventilation, and air-conditioned system.

An important factor influencing the utilization of the grid is the performance of the agents scheduling behavior. Scheduling and optimization is subject of many research [19]. The scheduling in this research is closely related to dynamic job shop scheduling. Dynamic job shop scheduling differs from static shop scheduling in the arrival of jobs which are arriving continuously over time in a random manner. Many approaches can be taken to tackle different aspects of this NP-hard problem. Scheduling algorithms diverge from integer linear programming and branch and bound techniques to bottle neck heuristics, constraint satisfaction, and local search methods [47–51]. An extensive literature review of job shop scheduling is given by Jain [32] and Cheng [52]. Brun and Portioli [53] argue that distributed systems have an edge over centralized systems and propose a multi-agent system for simulation of shop floor scheduling. However, there is not many research on agent-based systems in stochastic environments while agent-based system should then have the advantage. With the simulation it is shown that in a stochastic environment the performance drops significant. However, the agents can improve the performance with the investigated strategies.

A good overview of multi-agent scheduling is given by Weerdt and Clement [33]. A distributed scheduling method for heterarchical systems is described by Duffie and Prabhu [36]. The article summarizes the advantages of scheduling in heterarchical versus hierarchical manufacturing systems. A comparison between these types of manufacturing systems, related to this research, shows that the hierarchical approach has an advantage when problems arise at equiplets [54].

Chapter 8: Conclusion

This thesis presents an architecture and simulation for production in an agile manufacturing grid of equiplets. The implemented MAS is based on the Grid manufacturing paradigm. The agents in MAS have been provided with strategies to improve the performance of the production grid. The product agent has gained the ability to reschedule its product steps and the scheduling objectives: Load, EDD, and Load $\times$ EDD. Furthermore, queue jumping and reconfiguration of equiplet agents is implemented in the production grid. The simulation provides a tool for verification and analyzing of the production grid. This gives the possibility for evaluation of the production grid and gives insight into the efficiency of developed strategies.

Based on the results conclusions can be drawn. This answers the research questions:

1. *Is it possible to emulate and validate the MAS software of a grid with a simulation?*

MAS can be validated by imitating the behavior of the underlying equiplet software. This enables to verify the quality of the software. During the development and testing of the simulation software bugs have been found which otherwise would not be exposed. By developing a simulation that works with operational software a more proven result of the technology is given. Furthermore, the simulation enables to evaluate the performance of the production grid.

2. *What impact does reconfigurability of the production platforms have on the production efficiency of the grid?*

The reconfigurable production platforms have impact on the architecture and the performance of the production grid. The MAS is structured in a heterarchical architecture while equiplet agents itself are structured in a hierarchical architecture. Defining the equiplets' capability by combinations of modules allow these capabilities to be provided as a service to the grid. The architecture enables the underlying hardware of the equiplets to be easy reconfigured during runtime of the system. In a dynamic production environment this allows for a more extendable and maintainable system when equipment can be added or adapted separately. By adopting the RMS paradigm the architecture overcomes shortcomings of more conventional manufacturing systems.

Furthermore, the reconfiguration results show that the production grid is able to advice and handle reconfiguration of equiplets. This prevents shutting down the production during reconfiguration. Reconfiguring equiplets to fit the product demand improves the performance depending on the capabilities of the grid, i.e. the set-up of the equiplets. The simulation can be used to advice a set-up for the manufacturing of certain products. This gives insight in the equiplets that should be reconfigured to another capability before or during production.

3. *How do disturbances impact the manufacturing system?*

To answer the third research question: disturbances in the manufacturing process have great impact on the performance of the production grid. The grid will perform worse as products steps will not complete in time and delay the other products that scheduled behind the delayed product step. An utilization of 80% cannot be achieved. With the simulation the effectiveness of strategies for countering the effects of disturbances were investigated.

4. *What strategies or optimizations can counter the effect of disturbances in the manufacturing process?*

In a deterministic model, balancing the load between equiplets with the Matrix scheduling approach results in an overall better performance. Products taking a more selfish approach in scheduling their products steps with EDD scheduling, i.e. try to finish as quickly as possible, does not benefit the overall production of the grid. In a stochastic model the algorithm Load $\times$ EDD that combines the two objectives give the best result when not all products are able to be manufactured.

Furthermore, rescheduling the product steps is better than sticking to the schedule after an infeasible product schedule emerges. This results in a production grid that is able to manufacture more products. Although not all products are able to complete within their deadline.

In addition, queue jumping aims to counter the effects of disturbances. It allows a product to jump certain places in the queue at the equiplet. The procedure will increase the production, as products will have to wait shorter on other products.

Chapter 9: Future Work

The simulation together with the implemented architecture provides the possibility to investigate more schedule techniques and objectives. Objectives which are not yet explored are to minimize product work span, shortest queue first or other objectives [19].

The result of simulations with queue jumping gives an indication scheduling becomes redundant if equiplets do not stick to their schedules. It would be interesting to investigate whether products need to plan their production path before entering the grid and just take ad-hoc decisions where to go before production of a product step. An advantage could be a reduction of products in the grid with the same performance compared to when products schedule their product steps. The difficulty would be to decide whether products can enter the grid without overcrowding but maximizing the production.

A real-time simulation is needed to investigate the need for atomic scheduling. In a previous scheduling investigation [13] one planning board was used for all the schedules which one product agent could access. In the current implementation the schedules are distributed among the equiplet agents such that products can plan simultaneously. The equiplet agents are responsible for their own schedules. The performance could increase without atomic scheduling as multiple products can plan simultaneously. This would be the case when agents plan a few product steps such that the schedule procedure is fast enough to complete before the next product enters the gird or no conflicting time slots are scheduled. However when two agents try to reserve the same time slot, one of the agents needs to restart his scheduling procedure. A rule should prescribe which agent has priority: a given priority of the product, closest deadline, smallest of completion time of the preferred production path, or another rule depending on the objective of the gird. The rule could apply to the whole production path or only to the conflicting time slots. Although, when one conflicting time slot is not available the time slots thereafter become probably infeasible and should be released.

AGV systems play an important role in flexible manufacturing systems. The implementation of techniques to satisfy minimizing congestion, vehicle utilization, and throughput adds more complexity to the grid. Including the use of AGV's within the simulation adds more uncertainties as the reliability of the AGV's in service and travel times will affect the tardiness of the executing of product steps.

The simulation could be used to evaluate the effectiveness and efficiency of the grid layout to reduce manufacturing cost. At this stage of the research there is little known about the preferences of future implementations and therefore equiplet allocation in a facility layout design. To include these preferences in the simulation more has to be known about the possible equiplets available and products to be manufactured. Furthermore, research should show whether the simulation is sufficient to evaluate different layout alternatives or an extension has to be developed to optimize the layout.

Similar to industrial manufacturing systems, SCADA needs to be incorporated into the system to monitor and control the system [12]. The correct dependencies and communications need to be established to ensure safety of equipment, information, and execution of commands. At this stage a monitoring agent takes partially the responsibility of data acquisition i.e. to monitor the state and performance of the equiplets. This information is passed on to a human interface as a website. A possibility would be that an agent in the grid would be responsible to subscribe or poll equiplets for information and relay this to a human interface. Another approach would be that equiplets directly communicate with a human interface. As a control or optimization agent must have the same or similar information, the latter option would increase the communication in the grid. Such an optimization agent would be responsible to suggest reconfiguration of an equiplet. Either way, the implementation of providing information would be handled within a listener behavior of the equiplet agent. An alternative would be a web-service hosted by the equiplet to present information. The latter would be preferable when there are, next to an overview interface, local interfaces of the equiplet distributed over the manufacturing facility. The kind of information that would be displayed is still unknown. Which terminal or interface should display information as equiplet states, machine states, module states, communication logs, equiplet schedules, equiplet histories, and/or equiplet performances. From this information the relevant data needs to be chosen for each interface, control, and optimization entity in the grid.

The presented model is applied on a manufacturing production line. The allocation of resources is a natural application of the presented simulation model. Although due to the generalization of capabilities of the resources and products to be produced, the simulation model could be applicable in other field than manufacturing. An application that can be considered is the allocation of resources in the workplace. The products are the natural counterparts of the product or service a company develops or offers to customers. The equiplets would be the counterparts of the employees working on these products. The capabilities of the equiplet agents would be defined by abilities and skills of the employee. The agents could have, next to the time a service takes also, a performance index for solving or working on certain services. These indexes would be updated after finishing product such that the allocation of resources for next products would be more accurate. The agents learn from the given feedback on the quality and time worked on products. The reconfigurability of the grid would be achieved by sending employees to training courses. The challenging part of this would be the classification of the employees' abilities and skills which is more of a psychology nature.

To improve the productivity of employees in the company more performance measurements and metrics could be taking into account when planning the employee on certain tasks. The variation in work could lead to higher productivity.

Bibliography

[1] E. Puik and L. van Moergestel, "Agile multi-parallel micro manufacturing using a grid of equiplets," in *Precision Assembly Technologies and Systems* (S. Ratchev, ed.), vol. 315 of *IFIP Advances in Information and Communication Technology*, pp. 271–282, Springer Berlin Heidelberg, 2010.

[2] M. Mehrabi, A. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *Journal of Intelligent Manufacturing*, vol. 11, no. 4, pp. 403–419, 2000.

[3] E. Puik, L. van Moergestel, and D. Telgen, "Cost modelling for micro manufacturing logistics when using a grid of equiplets," in *Assembly and Manufacturing (ISAM), 2011 IEEE International Symposium on*, pp. 1–4, May 2011.

[4] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. V. Brussel, "Reconfigurable manufacturing systems," {*CIRP*} *Annals - Manufacturing Technology*, vol. 48, no. 2, pp. 527 – 540, 1999.

[5] D. Telgen, L. van Moergestel, E. Puik, P. Muller, and J.-J. Meyer, "Requirements and matching software technologies for sustainable and agile manufacturing systems," in *INTELLI 2013, The Second International Conference on Intelligent Systems and Applications*, pp. 30–35, 2013.

[6] D. Trentesaux, "Distributed control of production systems," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 971 – 978, 2009. Distributed Control of Production Systems.

[7] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Eng. Appl. Artif. Intell.*, vol. 22, pp. 979–991, Oct. 2009.

[8] S. Hu, "Paradigms of manufacturinga panel discussion," in *3rd Conference on reconfigurable manufacturing, Ann Arbor, Michigan, USA*, 2005.

[9] Y. Koren and M. Shpitalni, "Design of reconfigurable manufacturing systems," *Journal of Manufacturing Systems*, vol. 29, no. 4, pp. 130 – 141, 2010.

[10] H. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, 2005.

[11] H. ElMaraghy, T. AlGeddawy, A. Azab, and W. ElMaraghy, "Change in manufacturing research and industrial challenges," in *Enabling Manufacturing Competitiveness and Economic Sustainability* (H. A. ElMaraghy, ed.), pp. 2–9, Springer Berlin Heidelberg, 2012.

[12] B. Galloway and G. Hancke, "Introduction to industrial control networks," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 2, pp. 860–880, 2013.

[13] L. van Moergestel, E. Puik, D. Telgen, and J.-J. Meyer, "Decentralized autonomous-agent-based infrastructure for agile multiparallel manufacturing," in *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*, pp. 281–288, March 2011.

[14] S. L. Koh and L. Wang, "Overview of enterprise networks and logistics for agile manufacturing," in *Enterprise Networks and Logistics for Agile Manufacturing* (L. Wang and S. L. Koh, eds.), pp. 1–10, Springer London, 2010.

[15] A. Giret and V. Botti, "Engineering holonic manufacturing systems," *Computers in Industry*, vol. 60, no. 6, pp. 428 – 440, 2009. Collaborative Engineering: from Concurrent Engineering to Enterprise Collaboration.

[16] M. Wooldridge, *An Introduction to MultiAgent Systems.* Wiley, 2009.

[17] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology).* John Wiley & Sons, 2007.

[18] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a bdi-architecture," *KR*, vol. 91, pp. 473–484, 1991.

[19] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems.* Springer, 2008.

[20] C. McLean and S. Leong, "The expanding role of simulation in future manufacturing," in *Simulation Conference, 2001. Proceedings of the Winter*, vol. 2, pp. 1478–1486 vol.2, 2001.

[21] A. M. Law and D. M. Kelton, *Simulation Modeling and Analysis.* McGraw-Hill Higher Education, 3rd ed., 1999.

[22] J. Banks, *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice.* A Wiley-Interscience publication, Wiley, 1998.

[23] A. Law and M. McComas, "Simulation of manufacturing systems," in *Simulation Conference Proceedings, 1998. Winter*, vol. 1, pp. 49–52 vol.1, Dec 1998.

[24] L. van Moergestel, E. Puik, D. Telgen, and J.-J. Meyer, "Production scheduling in an agile agent-based production grid," in *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on*, vol. 2, pp. 293–298, 2012.

[25] J. Barbosa and P. Leitao, "Simulation of multi-agent manufacturing systems using agent-based modelling platforms," in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pp. 477–482, 2011.

[26] D. Telgen, E. Puik, L. van Moergestel, and J.-J. Meyer, "Distributed and heterarchical control for grid manufacturing," in *SICE Annual Conference (SICE), 2013 Proceedings of*, pp. 526–531, Sept 2013.

[27] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.

[28] D. Telgen, L. van den Brink, L. van Moergestel, E. Puik, and J.-J. Meyer, "Adding reconfiguration to an agile agent based production grid," in *Proceedings of the 25th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2015)*, 2015.

[29] E. Järvenpää and S. Torvinen, "Capability-based approach for evaluating the impact of product requirement changes on the production system," in *Advances in Sustainable and Competitive Manufacturing Systems* (A. Azevedo, ed.), Lecture Notes in Mechanical Engineering, pp. 173–185, Springer International Publishing, 2013.

[30] "Jade site — java agent development framework." `http://web.archive.org/web/20140722012000/http://jade.tilab.com/`. Accessed: 2014-07-22.

[31] "Fipa." `http://web.archive.org/web/20140706054022/http://www.fipa.org/`. Accessed: 2014-07-22.

[32] A. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390 – 434, 1999.

[33] M. de Weerdt and B. Clement, "Introduction to planning in multiagent systems," *Multiagent Grid Syst.*, vol. 5, pp. 345–355, Dec. 2009.

[34] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to algorithms.* MIT press, 2001.

[35] G. Curry and R. Feldman, "Single workstation factory models," in *Manufacturing Systems Modeling and Analysis*, pp. 69–108, Springer Berlin Heidelberg, 2011.

[36] N. A. Duffie and V. V. Prabhu, “Real-time distributed scheduling of heterarchical manufacturing systems,” *Journal of Manufacturing Systems*, vol. 13, no. 2, pp. 94 – 107, 1994.

[37] D. Dilts, N. Boyd, and H. Whorms, “The evolution of control architectures for automated manufacturing systems,” *Journal of Manufacturing Systems*, vol. 10, no. 1, pp. 79 – 93, 1991.

[38] E. Järvenpää, “Capability-based adaptation of production systems in a changing environment,” *Tampereen teknillinen yliopisto. Julkaisu-Tampere University of Technology. Publication; 1082*, 2012.

[39] A. Negahban and J. S. Smith, “Simulation for manufacturing system design and operation: Literature review and analysis,” *Journal of Manufacturing Systems*, vol. 33, no. 2, pp. 241 – 261, 2014.

[40] V. Komma, P. Jain, and N. Mehta, “An approach for agent modeling in manufacturing on jade reactive architecture,” *The International Journal of Advanced Manufacturing Technology*, vol. 52, no. 9-12, pp. 1079–1090, 2011.

[41] G. Langer and L. Alting, “An architecture for agile shop floor control systems,” *Journal of Manufacturing Engineering*, vol. 19, no. 4, pp. 267–281, 2000.

[42] A. M. Florea and P. D. Cristea, “Multi-agent model of manufacturing systems,” in *In Proceedings of WMC 1999, the Second World Manufacturing Congress*, pp. 27–30, 1999.

[43] S. Bussmann, N. R. Jennings, and M. Wooldridge, *Multiagent systems for manufacturing control: a design methodology.* Springer Verlag, 2004.

[44] K. Schild and S. Bussmann, “Self-organization in manufacturing operations,” *Commun. ACM*, vol. 50, pp. 74–79, Dec. 2007.

[45] V. Mařík, P. Vrba, K. H. Hall, and F. P. Maturana, “Rockwell automation agents for manufacturing,” in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS ’05, (New York, NY, USA), pp. 107–113, ACM, 2005.

[46] F. Maturana, R. Staron, K. Hall, P. Tich, P. lechta, and V. Mak, “An intelligent agent validation architecture for distributed manufacturing organizations,” in *Emerging Solutions for Future Manufacturing Systems* (L. Camarinha-Matos, ed.), vol. 159 of *IFIP International Federation for Information Processing*, pp. 81–90, Springer US, 2005.

[47] V. Vinod and R. Sridharan, “Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system,” *International Journal of Production Economics*, vol. 129, no. 1, pp. 127 – 146, 2011.

[48] H. S. Fathabadi, S. Khodayifar, and M. Raayatpanah, “Minimum flow problem on network flows with time-varying bounds,” *Applied Mathematical Modelling*, vol. 36, no. 9, pp. 4414 – 4421, 2012.

[49] J. Kschel, T. Teich, and B. Zacher, “Real-time dynamic shop floor scheduling using evolutionary algorithms,” *International Journal of Production Economics*, vol. 79, no. 2, pp. 113 – 120, 2002. Theoretical Approaches and Decision Support.

[50] H. Wenqi and Y. Aihua, “An improved shifting bottleneck procedure for the job shop scheduling problem,” *Computers & Operations Research*, vol. 31, no. 12, pp. 2093 – 2110, 2004.

[51] T. ElMekkawy and H. ElMaraghy, “Real-time scheduling with deadlock avoidance in flexible manufacturing systems,” *The International Journal of Advanced Manufacturing Technology*, vol. 22, no. 3-4, pp. 259–270, 2003.

[52] R. Cheng, M. Gen, and Y. Tsujimura, “A tutorial survey of job-shop scheduling problems using genetic algorithmsi. representation,” *Computers & Industrial Engineering*, vol. 30, no. 4, pp. 983 – 997, 1996.

[53] A. Brun and A. Portioli, “Agent-based shop-floor scheduling of multi stage systems,” *Computers & Industrial Engineering*, vol. 37, no. 12, pp. 457 – 460, 1999. Proceedings of the 24th international conference on computers and industrial engineering.

[54] D. Telgen, L. van Moergestel, E. Puik, A. Streng, R. Scheefhals, T. Bakker, L. Hustinx, A. van den Brink, and J.-J. Meyer, “Hierarchical management of a heterarchical manufacturing grid,” in *Proceedings of the 24th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2014)*, pp. 825–832, 2014.

Appendix A: Simulation Results

Scheduling

Matrix Scheduling

ρ	Finished	Failed	Production Time	Load
0.70	31592	0	696,43	0,704
0.75	33830	0	859,79	0,754
0.80	36056	0	1342,06	0,803
0.85	37663	313	8967,07	0,840
0.90	37733	2481	9624,30	0,841

Load Scheduling

ρ	Finished	Failed	Production Time	Load
0.70	31595	0	592,83	0,704
0.75	33832	0	766,58	0,754
0.80	36053	0	1627,09	0,803
0.85	37489	481	9044,14	0,836
0.90	36968	3248	9752,46	0,824

EDD Scheduling

ρ	Finished	Failed	Production Time	Load
0.70	31593	0	695,4	0,704
0.75	33829	0	858,9	0,754
0.80	36059	0	1350,85	0,804
0.85	37477	491	9136,35	0,835
0.90	37152	3060	9792,51	0,828

Load $\times$ EDD Scheduling

ρ	Finished	Failed	Production Time	Load
0.70	31593	0	695,4	0,704
0.75	33829	0	858,9	0,754
0.80	36059	0	1350,85	0,804
0.85	37477	491	9136,35	0,835
0.90	37152	3060	9792,51	0,828

Table 1: Deterministic scheduling for $0.7 \leq \rho \leq 0.9$

Matrix Scheduling

ρ	Finished	Failed	Overdue	Production Time	Load
0.60	27154	0	226	1294,90	0,601
0.65	29373	0	607	2337,92	0,649
0.70	30730	418	18677	15781,31	0,683
0.75	30777	2609	19378	16790,53	0,684
0.80	30835	4788	19659	16986,92	0,685

Load Scheduling

ρ	Finished	Failed	Overdue	Production Time	Load
0.60	27162	0	180	1033,45	0,600
0.65	29384	0	352	1597,97	0,650
0.70	31421	0	10387	8689,17	0,696
0.75	31186	2183	21755	16993,05	0,692
0.80	31205	4403	22287	17250,68	0,693

EDD Scheduling

ρ	Finished	Failed	Overdue	Production Time	Load
0.60	27150	0	227	1314,62	0,601
0.65	29371	0	653	2395,84	0,649
0.70	30576	544	20367	16633,90	0,680
0.75	30463	2895	22189	17825,99	0,678
0.80	30434	5150	22658	18178,86	0,677

Load $\times$ EDD Scheduling

ρ	Finished	Failed	Overdue	Production Time	Load
0.60	27159	0	256	1105,46	0,601
0.65	29386	0	552	1909,2	0,650
0.70	30745	388	20169	16291,54	0,683
0.75	30848	2516	21276	17205,56	0,686
0.80	30867	4730	21773	17484,36	0,687

Table 2: Stochastic scheduling for $0.6 \leq \rho \leq 0.8$

Queue Jumping

$\rho = 0.70$

q	Finished	Failed	Overdue	Production Time	Load
0	21551	9942	7420	7745.60	0.490
5	30278	1138	1398	6685.67	0.686
10	31596	0	0	746.30	0.713
15	31592	0	0	715.90	0.713

20	31594	0	0	714.52	0.714
25	31593	0	0	710.08	0.713
infinite	31592	0	0	706.55	0.713

$\rho = 0.75$

q	Finished	Failed	Overdue	Production Time	Load
0	22563	11183	8270	7999.01	0.514
5	31369	2301	2883	6470.12	0.710
10	33829	0	0	892.87	0.764
15	33835	0	0	801.25	0.763
20	33830	0	0	795.58	0.764
25	33835	0	0	792.59	0.764
infinite	33834	0	0	785.91	0.764

$\rho = 0.80$

q	Finished	Failed	Overdue	Production Time	Load
0	23620	12396	9239	8281.30	0.538
5	32770	3087	4922	6752.27	0.742
10	36048	0	0	1369.52	0.813
15	36072	0	0	950.10	0.814
20	36070	0	0	918.18	0.814
25	36072	0	0	911.75	0.813
infinite	36069	0	0	904.82	0.814

$\rho = 0.85$

q	Finished	Failed	Overdue	Production Time	Load
0	24145	14074	10378	8717.91	0.550
5	33518	4581	11506	7553.73	0.759
10	37854	194	943	7089.48	0.854
15	38302	0	0	1348.60	0.864
20	38306	0	0	1152.40	0.864
25	38304	0	0	1111.90	0.864
infinite	38306	0	0	1092.41	0.864

$\rho = 0.90$

q	Finished	Failed	Overdue	Production Time	Load
0	24485	15892	11519	9097.00	0.557
5	33621	6683	17090	8345.34	0.762
10	37749	2475	5989	9456.90	0.853
15	40465	0	0	3284.14	0.912
20	40517	0	0	1901.08	0.914
25	40524	0	0	1616.84	0.913
infinite	40535	0	0	1488.03	0.915

$\rho = 0.95$

q	Finished	Failed	Overdue	Production Time	Load
0	24584	17981	12329	9478.14	0.560
5	33821	8687	20495	8813.64	0.765
10	37825	4638	7748	9608.30	0.853
15	40763	1679	2355	9322.15	0.920
20	41839	596	834	8856.75	0.944
25	42348	112	131	7701.46	0.956
infinite	42711	0	0	2884.60	0.963

Table 3: Queue jumping results

Reconfiguration

base

	Finished	Failed	Overdue	Production Time	Load
base	10637.3	10630.5	0	9546.25	0.3531
resched.	10639.9	10624.7	0	9546.27	0.3532
qj	12083.0	9173.50	1174.60	9323.91	0.4108

reconfig

	Finished	Failed	Overdue	Production Time	Load
base	20636	617	0	5878.9	0.773
resched.	20685	578	0	5721.7	0.775
qj	20924	397	101	3283.0	0.791

stochastics

	Finished	Failed	Overdue	Production Time	Load
base	7103	14181	5463.7	11617.5	0.2376
resched.	9042.5	12188.6	5653.1	15063.9	0.2939
qj	11547.3	9722.10	1991.70	8901.64	0.3921

reconfig and stochastics

	Finished	Failed	Overdue	Production Time	Load
base	9693	11567	5879	11223.7	0.363
resched.	16380	4691	14063	17161.2	0.587
qj	20830	434	138	5507.6	0.763

stochastics and breakdowns

	Finished	Failed	Overdue	Production Time	Load
base	6731.50	14563.5	5117.10	11955.30	0.2309
resched.	8912.30	12317.70	5817.40	15433.79	0.2946

qj	11308.4	9961.70	1926.80	8840.24	0.3916

reconfig, stochastics, and breakdowns

	Finished	Failed	Overdue	Production Time	Load
base	9295	11971	5759	11592.6	0.353
resched.	15089	6024	12608	16940.2	0.541
qj	19654	1540	607	8382.7	0.731

Table 4: Reconfiguration results