Agent Technology in Agile Multiparallel Manufacturing and Product Support

Leo van Moergestel

SIKS Dissertation Series No. 2014-31
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Agent Technology in Agile Multiparallel Manufacturing and Product Support

Agenttechnologie voor Flexibele en Herconfigureerbare
Parallele Productie en Productondersteuning
(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op
gezag van rector magnificus, prof.dr. G.J. van der Zwaan, ingevolge van het
besluit van het college voor promoties in het openbaar te verdedigen op
woensdag 10 september 2014 des middags te 12.45 uur door

Leonardus Joseph Maria van Moergestel

geboren op 24 maart 1955, te Dussen

Promotor: Prof.dr. J-J.Ch. Meyer

The work presented in this thesis was supported by the University of Applied Sciences Utrecht (HU)

Contents

Dankwoord

In januari 2010 startte officieel mijn promotietraject. In januari 2014 lag een eerste versie van mijn proefschrift bij mijn promotor. Ik ben de Hogeschool Utrecht zeer erkentelijk voor het bieden van de mogelijkheid om te promoveren. Het proefschrift draagt op de omslag naast de titel mijn naam. Dit zou de indruk kunnen wekken dat ik in mijn eentje de klus heb geklaard. Niets is minder waar! Veel mensen hebben mij gesteund en zijn direct of indirect van onschatbare waarde geweest voor mijn promotieonderzoek.

Veel studenten hebben meegewerkt in het onderzoekssemester. In groepjes hebben ze gedurende een half jaar full-time meegewerkt aan een onderdeel van het onderzoek, vaak met het co-auteurschap van een wetenschappelijke paper als eindresultaat. Op het gevaar af iemand over het hoofd te zien, ga ik toch de namen in alfabetische volgorde noemen: Alexander Streng, Bas Alblas, Dennis van der Werf, Duncan Jenkins, Franc Pape, Frank van Rooden, Garik Hakopian, Glenn Meerstra, Hendrik Folmer, Hielke Veringa, Jaap Koelewijn, Johan van Schieveen, Lars Stolwijk, Maarten Aalbers, Martijn Beek, Martijn van Aalst, Martin Rijntjes, Mart van de Moosdijk, Mathijs Kuil, Matthijs Grünbauer, Niels van Nieuwenburg, Pascal Muller, Ramòn Hagenaars, Raymond Siudak, Robbert Proost, Roel Blaauwgeers, Roy de Kok, Roy Scheefhals, Stefan Marchal, Tommas Bakker, Wim Wakelkamp en Wouter Langerak. Bedankt jongens (meisjes zijn zeldzaam bij technische informatica) voor jullie enthousiasme en jullie bijdrage aan het onderzoek.

Collega's en medewerkers van het intitute for ICT van de HU wil ik ook graag bedanken voor hun belangstelling en voor de soms serieuze en vaak informele gesprekken over mijn promotiewerk. Ik heb met leidinggevenden binnen en buiten het institute for ICT regelmatig gesproken over de voortgang van mijn onderzoekswerk. Bij iedere mijlpaal kon ik rekenen op een reeks van positieve reacties. Medewerkers van het kenniscentrum wil ik ook bedanken en met name Janny Bakker de Leeuw voor het snel en kordaat regelen van allerlei zaken, zoals reserveringen, boekingen en financiële overzichten bijhouden van projecten.

De leden van de leescommissie hebben hun werk zorgvuldig gedaan. Met prof. dr. Cees Witteveen en prof. dr. Jaap van den Herik heb ik een mondeling onderhoud gehad. Prof. dr. Frances Brazier, prof. dr. Andrew Tanenbaum en prof. dr. Jörg Müller hebben via e-mail hun aanwijzigingen en tips ter verbetering doorgegeven. Hartelijk bedankt, thank you very much, herzlichen Dank!

In de nu volgende drie stukjes tekst zal ik mij richten tot drie personen die een bijzondere rol hebben gespeeld tijdens mijn onderzoek. De eerste in het rijtje is de lector Erik Puik. Vijf jaar geleden gaf ik te kennen dat ik bij jouw lectoraat onderzoek wilde doen. Jij hebt toen het voorstel gedaan waarmee de basis voor dit onderzoek is gelegd. Ik ben zeer onder de indruk van je kennis en inzicht in een heel breed vakgebied. Je hebt het onderzoek aangestuurd en mij ook weer ruim baan gegeven voor de verdere uitvoering. Dat is op zich al heel knap, want aansturen en ruim baan geven lijken elkaar in de weg te staan, maar niet op de wijze die jij hanteerde. Ik heb veel van je geleerd en ben blij dat ik bij je lectoraat verder kan werken aan ons onderzoek.

De tweede persoon die ik persoonlijk lof wil toewuiven is Daniël Telgen. Het onderzoek waar Erik en ik mee bezig waren, was ambitieus en veelomvattend. Een derde onderzoeker erbij was geen luxe en gelukkig hebben we die in jou gevonden. Jouw enthousiasme, gedrevenheid, ambitie en streven naar kwaliteit hebben het onderzoek een enorme boost gegeven en dat geldt ook voor de opleiding technische informatica, die wij beiden een warm hart toedragen.

En last but not least gaat mijn dank uit naar mijn promotor John-Jules Meyer. We zijn leeftijdsgenoten en we leerden elkaar in de tachtiger jaren van de vorige eeuw (tjonge, wat klinkt dat lang geleden) kennen. Toen ik als wetenschappelijk systeemprogrammeur op de VU werkzaam was in de groep van Andy Tanenbaum, was jij promovendus bij Jaco de Bakker. Daarna zijn we ieder onze weg gegaan totdat ik op zoek ging naar een hoogleraar, bij voorkeur in Utrecht, die agent technology als specialisatie had en die mogelijk promotor voor mijn onderzoek kon zijn. Het was een blij weerzien en al snel besloot je dat ik bij jou als promovendus terecht kon. Bedankt voor het in mij gestelde vertrouwen. Toen ik later in een mail naar Andy Tanenbaum schreef dat ik het promotiepad was opgegaan en jij mijn promotor was, mailde hij als reactie hierop: *John-Jules is a great guy, you're lucky to have him as your advisor.* Dit krachtige statement geeft precies de situatie weer. Ik zou bijna zeggen dat het nog zwak uitgedrukt is. Elk bezoek aan jou heb ik als buitengewoon prettig ervaren. Je hebt mij op een vriendelijke manier op mogelijke fouten gewezen en me tegelijk ook geprezen om de zaken die ik goed deed. Zoiets geeft je vertrouwen en zin om enthousiast door te gaan. Je dacht mee over de problemen en gaf me nuttige tips en adviezen. Ik ben heel blij dat we hebben afgesproken dat onze samenwerking niet stopt na mijn promotie.

In dit dankwoord past het ook om personen te noemen die niet direct aan het onderzoek hebben meegewerkt, maar indirect wel een belangrijke rol hebben gespeeld. Ik wil mijn echtgenote en kinderen bedanken voor hun belangstelling, de support en het feit dat ze accepteerden dat ik nogal vaak bezig was met mijn werk (en dan druk ik me nog zwak uit). Mijn gezin heeft zich weten te redden tijdens mijn afwezigheid op de vele conferenties die ik heb bezocht. Ook overige familieleden en vrienden die mij hebben gesteund wil ik bedanken. Natuurlijk wil ik mijn ouders niet vergeten, Adriaan van Moergestel (1897-1979) en Adriana van Moergestel Nieuwesteeg (1916-2011), die mij de kans hebben gegeven om te studeren en die best wel trots zouden zijn geweest met dit resultaat. Wie weet, misschien zijn ze dat wel.

Zaandam, 2014

Chapter 1

Agent-based manufacturing

This thesis deals with agent technology and its use in agile multiparallel manufacturing and product support. We will develop an agent-based software system to control specifically developed production hardware. The use of agent technology is extended to the whole life cycle of the products. This chapter presents an introduction to agile agent-based manufacturing. It describes the concept of the agent-based software infrastructure for agile industrial production. This is the basis of the thesis. Several definitions and descriptions are given for both manufacturing as well as agent technology. The hardware and the software are described and compared to standard production technologies. In this introductory chapter the focus is on a basic global view of the concept and the system as well as its subparts that have been developed. In agent-based manufacturing as it will be introduced, the production is done on special devices called equiplets. A grid of these equiplets connected by a fast computer network and a possibility to transport products, should be capable of producing a variety of different products in parallel.

The content of this chapter is as follows. First an introduction about manufacturing concepts will be presented. Next a widely accepted standard production approach is reviewed and summarized. This will reveal why a new approach might be useful. Then, the new approach discussed in this thesis will be introduced together with the technologies it is based upon. This will lead to topics of research and related research questions. Related work will also be discussed and compared to our work. The chapter ends with an overview of the chapters that go into more detail about specific parts of the implementation.

1.1 Introduction

The requirements of modern production systems are influenced by new demands such as time to market and customer-specific small quantity production. In other words, the transition time from product development to production should be minimal and small quantity production must be cheap. To fulfil these requirements we need to develop new production methods. Such a new approach means new production hardware as well as co-designed software. At the Utrecht University of Applied Science we have developed special production platforms that are cheap, agile and easy configurable (Puik and Moergestel, 2010). These platforms can operate in parallel. We call these platforms equiplets and a collection of these equiplets is called a production grid. The idea behind the concept is that we need a production system that is capable of producing a lot of different products in parallel. This is what we call multiparallel manufacturing. The software infrastructure for such a production grid is highly responsible for the agile and diverse way of production. In this chapter we present some possible realizations of the control software and we will propose a model that seems interesting for further investigation. This model is based on agent technology. Though we based our model on our own designed production hardware, the agent-based approach can be useful in other production environments as well as will be discussed in chapter 6.

The concept of using a collection of cheap machines is comparable to the research done around 1980 where the focus was on using cheap microprocessor-based computer systems to cooperate in a multiprocessor or multicomputer environment (Tanenbaum et al., 1991), but because the focus is now on manufacturing real physical products, there are many differences and also many specific problems to be solved to make the concept work.

1.2 Background and domain

This section is dedicated to manufacturing technologies. The main reason to have this section is to give a background and reference models for the concepts introduced in the later part of this chapter. The section contains the following subsections.

1.2.1 Production concepts.

1.2.2 Push-driven versus pull-driven manufacturing.

1.2.3 Lean manufacturing.

1.2.4 Agile manufacturing.

1.2.1 Production concepts

In this thesis the words production, manufacturing and assembling are used intermixed. There are some subtle differences in the meaning of these words.

- Production is used as the most generic term. It can be used for material products as well as things like software, ideas, theories etc.
- The word manufacturing has its origins in Latin: *manus* meaning hand and *facere* meaning to make. Though it literally means make by hands it is now also applied to situations where material things are made using machines.
- Assembling is used for the type of manufacturing where components or sub-parts are put together using several possible techniques to make the final product.

When we take a closer look at making things, we may distinguish three approaches for production.

1. Making a single product. This is normally the case in situations where a specific one time product is needed. Examples are ships, some buildings, special items where a single unit is needed.
2. Continuous production is a type of production, where there is a continuous stream of output. This type of production is encountered in the chemical industry.
3. Batch production. This type of production is mostly used to make a number of similar products. Normally the number is rather big as we shall see shortly.

When we look at the production infrastructure, especially the infrastructure for batch processing, again three different approaches can be seen.

1. Dedicated production line. This is a concept that is widely used and fits the need for cheap mass production. The machinery that is used is dedicated to do one specific task in the production.
2. Flexible manufacturing system (FMS). A flexible manufacturing system has, as its name suggests, some flexibility to react to changes. The most well known change is changing to new product types. This change can be incorporated in the production machines. This concept was introduced in the time that cost-effective Computer Numerical Control (CNC) machines became available. Another type of flexibility is routing

flexibility. This concept is based on the fact that there might be several machines to perform the same operation on a product. This introduces choices for the product in following the production line.

3. Reconfigurable Manufacturing System (RMS). A reconfigurable manufacturing system is a manufacturing system that is designed for fast changes, both in hardware as well as software components, in order to quickly adjust production capacity and functionality in response to sudden changes in market or in changes in requirements. The main goal of RMS is to achieve cost-effective responsiveness. An RMS can adapt easier and faster to changes than an FMS. The drawback is that RMS is more complicated. To achieve a high level of recongurability, the system should meet the following characteristics in its design (Koren and Ulsoy, 2002):

   - Modularity. The basic components are modules that can easily be exchanged or replaced.
   - Scalability. To adapt to changing demands, scalability is an important characteristic.
   - Integrability. Modules can be easily integrated in the system.
   - Convertibility. Changes to the production system are easy to achieve.
   - Customization. Adaptation of the system to specific needs are possible.
   - Being diagnosable. To prevent that searching for failing modules takes along time, modules and the system itself should be diagnosable.

   The cost-effectiveness of RMS is achieved by designing a system with an adjustable structure, and around a part family. An adjustable structure enables system scalability in response to market demands and system/machine adaptability to new products. Structure may be adjusted at system level by adding new machines, and at machine level by adding/removing machine software (Koren et al., 1999).

1.2.2 Push-driven versus pull-driven manufacturing

Standard mass production is mostly push-driven. This means that the expected purchase of a product is anticipated for and products are pushed into the market. Pull-driven waits for demands for a product and at the moment the production is started, it is sure that the market will accept it.

1.2.3 Lean manufacturing

Lean manufacturing has its origins in Japan. The production at Toyota has been the model for lean manufacturing (Shingo, 1989). The concept is based on five steps:

1. What is the value of the product from the customers perspective?
2. Discover where in the production process this value is added.
3. Determine the waste in the process, remove it and shorten the duration of the lead time.
4. Apply pull-driven production instead of push-driven production.
5. Keep the waste away and try to optimise the process.

The challenge is to discover what really adds value to the product. For example, keeping a lot of products or half products in stock is considered waste. From the client perspective it does not matter how big the stock is. The time the client has to wait for his product is a very important issue. So the production stream should be optimal with a minimum of internal delay. The aforementioned steps result in a set of best practices.

The lean concept is not limited to manufacturing. Production of software can also use this concept. Poppendieck (Poppendieck and Cusumano, 2012) lists 10 basic practices which make Lean Manufacturing so successful and their application to software development. It is interesting to see how the lean concept can be applied to such a thing as software development.

1. Eliminate waste eliminate or optimize consumables such as diagrams and models that do not add value to the final deliverable.
2. Minimize inventory minimize intermediate artifacts such as requirements and design documents.
3. Maximize flow use iterative development to reduce development time.
4. Pull from demand support flexible requirements.
5. Empower workers generalize intermediate documents, tell developers what needs to be done, not how to do it.
6. Meet customer requirements work closely with the customer, allowing them to change their minds.
7. Do it right the first time test early and refactor when necessary.

8. Abolish local optimization flexibly manage scope.
9. Partner with suppliers avoid adversarial relationships, work towards developing the best software.
10. Create a culture of continuous improvement allow the process to improve, learn from mistakes and successes.

1.2.4 Agile manufacturing

In response to the customers demand, manufacturing companies have to focus on low cost, high quality and rapid responsiveness (Koren and Ulsoy, 2002). A new paradigm called Agile Manufacturing was invented. It focuses on agility, i.e. the quick and accurate response to changes in the market and technology while controlling production cost. A slightly adapted definition from (Goldman et al., 1995) is:

Definition 1 (Agile manufacturing)**.** *An agile manufacturing system is a system that is capable of operating profitably in a competitive environment of continually and unpredictably changing customer requirements.*

In the next section where standard production automation is discussed, we will discover that this type of production is not agile by itself.

1.3 Standard production automation

Standard production software is mostly designed for batch production or continuous production. Continuous production can be considered as an endless batch. These production approaches are characterized by the fact that it is bulk processing. Many items or a large quantity of the same products are produced. We will discuss standard automation software, the way it is used, its properties and its shortcomings.

1.3.1 Standard automation software

The software for standard production systems is based on a layered model. This model is mostly referenced as the automation pyramid (Vogel-Heuser et al., 2009) (Figure 1.1).
Below we will give a short explanation of the layers in this pyramid:

- At the top, we find the business management software. This is the software level where the orders for production come in and where the

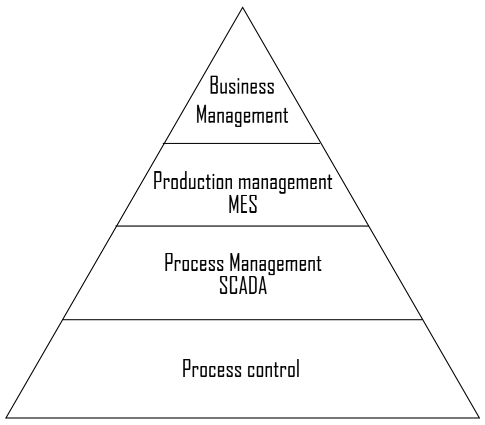


Figure 1.1: Automation pyramid

connection with clients, suppliers and other important things in the outside world is handled.

- The production management layer software is mostly covered by software systems called MES. MES is an abbreviation of Manufacturing Execution System. This software enables high level control over production facilities in a broad sense. We will describe the functionality of the MES layer in more detail later.

- Process management layer is the software that supervises the process control devices. It will also collect production data. The software in this layer is mostly referred to as SCADA, which is an abbreviation of Supervisory Control And Data Acquisition.

- The process control layer is responsible for the actual production process itself. In an automated environment the software controls motors, heating devices, robot arms, etc.

To understand a production system based on this software model, we will first go into more detail, especially the MES and the SCADA layer will be examined.

MES layer

The functionality of the MES layer is formulated by the Manufacturing Enterprise Solutions Association (MESA). They defined eleven functions for the MES layer (Kletti, 2007).

1. Resource allocation and status. The status of the resources is monitored and controlled. Manufacturing systems as well as tools and materials and also human workers are resources controlled by the MES.

2. Operations scheduling. Planning of the usage of the resources and controlling the performance of the resources.

3. Dispatching production units. Execute orders and allocate resources for the execution.

4. Document control. Management of information and making information available to all involved parties. The information is about products, processes, orders, recipes, authorisation, instructions for workers, system documentation and batch reports.

5. Data collection/acquisition. Collect and manage data about the use of resources and other important data about the production.

6. Labour management. Management of workers, work planning schemas, qualification and authorisation of workers.

7. Quality management. Logging, tracking and analysing the product quality and process characteristics.

8. Process management. Control and support the work to be done and keep operators informed.

9. Maintenance management. Planning of maintenance to keep the production systems at the required validation level.

10. Product tracking and genealogy. Create a product history by tracking the processing of materials and also the source of the materials used in the production.

11. Performance analysis. Analyse the process execution and compare this with the planned process execution to check if there are bottlenecks in the process execution. Find ways to circumvent problems and keep this information available for future use.

It depends on the type of production what MES functions should be available. For example in the food and drug production, product tracking and genealogy is obligatory while in other industries this might be optional. Most MES implementations are based on a realtime database.

SCADA layer

The SCADA-layer gets its information or production commands from the MES layer. Most SCADA implementations cover the following functions:

1. Translation of production command from the MES-layer or operator commands to a set of actions to be done by the SCADA layer.

2. Execute these actions. These actions are more specific than the global order commands from the MES-layer.

3. Collect and log data from the production layer. This is the data-acquisition task of SCADA.

4. Send appropriate data to the MES layer. The SCADA system will act here as a filter, sending only relevant data to the MES layer.

5. Setup and maintain an operator interface. Most SCADA systems offer operators a graphical user interface to overview and control the production process.

6. Handle alarm situations in the production process. When a SCADA system is configured for a certain environment, alarm conditions are recorded in the system, so they can be recognized at an early state.

Most SCADA systems are capable to perform more functions, but for our understanding of the automation process, the aforementioned functions are the most important.

1.3.2 Layers at work

We will use Figure 1.2 to describe in short what will happen when an order for a certain quantity of products is received. The top layer will issue a production request to the MES layer. This layer makes an inventory of the the resources needed to make the product and checks for availability (planning resources). At the time the resources are available, a product batch command is issued to the SCADA layer. This layer will control the production process by issuing commands to the production equipment. All layers send their feedback to the layer they receive commands from. So the MES layer will inform the top layer (sales) about the status of the production and also when the product batches are ready for shipment. This description is in some aspects a simplification, but gives a good idea about the production process as a whole. A more detailed layer of the automation piramid is given in Figure 1.3. Here all subsystems are visible and the type of networking

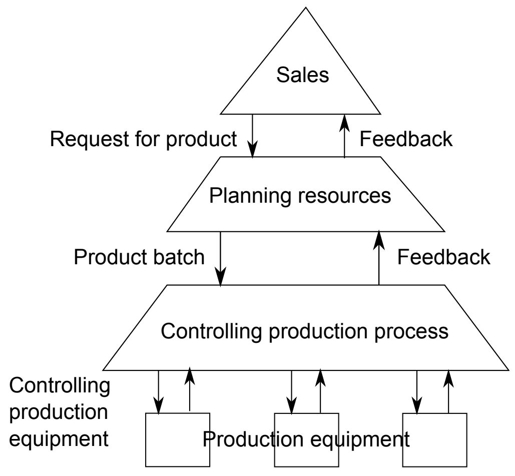


Figure 1.2: Layers in the automation piramid at work

the layers are connected with. The hard realtime systems and networks are at the bottom, while the not so time-critical systems are positioned more towards the top. We do not explain figure 1.3 in detail at this moment. The main reason is to show the actual position of MES and SCADA in the general manufacturing software model.

1.3.3 Properties of standard automation

As stated earlier, this production automation model is designed for producing large quantities of the same product. Normally these products are produced in batches.

Definition 2 (Batch process (Shaw, 1982))**.** *A process is considered to be a batch process if the process consists of a sequence of one or more steps or phases that must be performed in a defined order. The completion of this sequence of steps creates a finite quantity of finished products. If more products are to be created, the sequence must be repeated*

To make a product (that is a real hardware thing, not software or a service) one needs raw material and actions to work on this raw material. In many cases the raw material are actually components that are used as parts of the final product. Normally these components are also produced by a production process. As stated earlier, the production automation model is designed for producing large quantities of the same product. Two approaches of batch production exist. To understand what the two approaches are, consider a product that is made by a sequence of actions performed by machinery or craftsmen. These actions are also called production steps. In Chapter 2

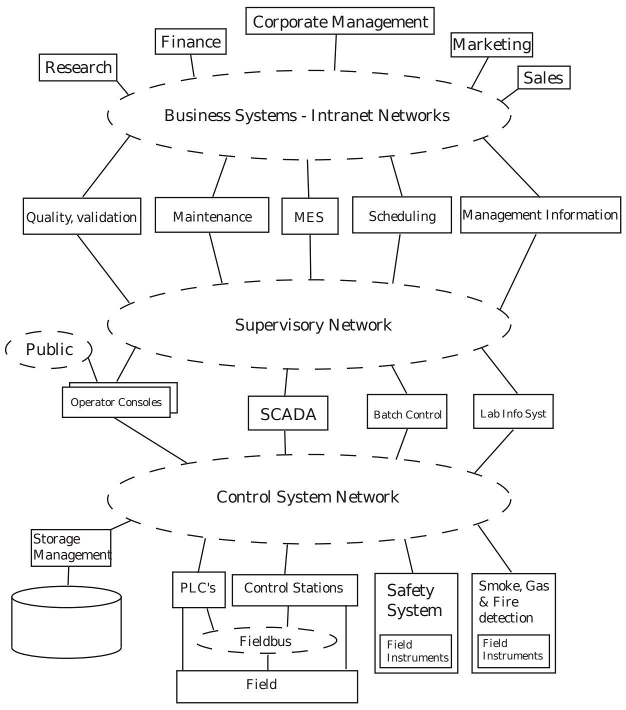


Figure 1.3: Layers in the automation piramid in more detail

a definition of a product step in our context will be given. Here we mention two different approaches.

1. Stepwise approach: the production starts and the first action of the sequence is performed. This leaves a set of incomplete products. Then the production environment is changed. This could be an adjustment of the machinery to perform a new type of action or the use of other tools by craftsmen to perform an action. This is repeated for all necessary steps and in the last step the final product is created. The approach is also appropriate for production of one single product. Because we need storage for the intermediate sets of incomplete products after every step, the approach is used for small scale production. A second property of this type of production is that there is a long delay between start of production and final completion of the products. The investment on machinery however is lower than in the next approach, because we adapt (as far as possible) the machines and/or craftsmen after each step to the next step.

2. Pipeline approach: in this approach all machinery (or craftsmen with specific tools) is available and the product to be made is handed over

to the machine that is capable to perform the next action for the production.

For both cases there is an optimum for the size of a batch. Though this size depends on several factors, the easiest approach is to consider the cost of the overhead of batch switching and the cost of storage and inventory. The optimum batch size is given by the point where the storage and inventory cost plus the batch cost for a given size are at a minimum (see figure 1.4). In practice this is more complicated as the price of raw material may fluctuate and other parameters such as market demand influence the optimum (Sarker and Khan, 2013). Between these two types, there is also a hybrid approach

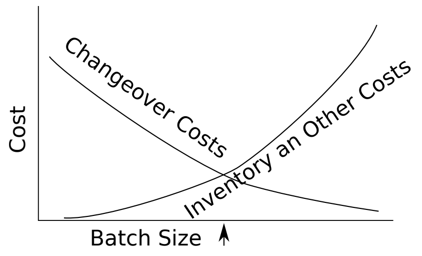


Figure 1.4: Optimum batch size

where the pipeline approach is used to make half-products that are collected to form a set and then handed over to another pipeline that will build the actual product. This situation occurs when production platforms produce components that are used as 'raw material' by other production platforms (that could be owned by a different company). A batch is a quantity of products produced without interruption. The size of a batch should be big enough to be cost-effective but should be limited because of maintenance and because of production supply fill-up. For every different batch a software configuration of normally the lower two layers of the automation pyramid should be available. The transition from one batch to the next one is called a batch switch. There are two types of batch switches.

1. A switch between batches of the same product.
2. A switch to a batch of a different product.

The overhead of a switch of the first type is not so large, but still some time is required. This time is normally used for preventive maintenance of the production equipment, for filling up component trays, etc. What also should be done is allocate resources for the next batch and adjust the software configuration of the lowest two layers to these newly allocated resources. In

Figure 1.5 we use t_s to denote the time for this switch. As can be seen in the figure, a larger batch introduces relative smaller overhead because t_s is mostly independent of the batch size.

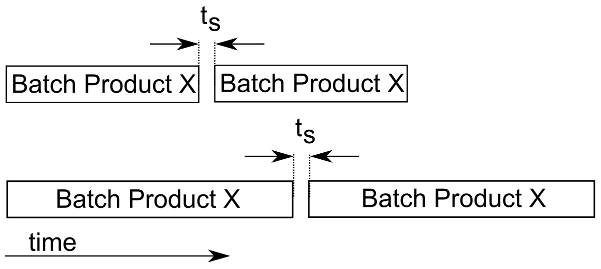


Figure 1.5: Batch switch overhead when switching to a new batch of the same product

A switch of type two takes a longer time because in addition to t_s we also need time t_p to reconfigure the software on the lower two layers and perhaps the hardware of the lowest layer (see Figure 1.6). The production equipment will get different software to operate and also the SCADA system needs to be reconfigured to match the new situation. Sometimes this means that a production platform or parts of it are unavailable for some time because of the reconfiguration that should also be tested of course. As a consequence batch switching of this type introduces a lot of overhead and production (and the final product) is cheaper if we produce batches of the same product and not a sequence of batches for different products. There are situations where switching between different batches is incorporated in the production of a manufacturing plant. Consider for example the food industry. To produce a batch of peanut butter, a production line could be set up. There are however variants to standard peanut butter (with honey, pieces of peanuts, chocolate etc.). Normally the market for these variants is not the same, meaning that the amount of production of these variants could be lower. It is in that case not a good idea to set up separate production lines for these variants. It is more cost-effective to produce a batch of variant A, next a batch of variant B, and several batches of the most popular standard product. The same production line should be used in that situation, so switching between different batches becomes compulsory. All batches are also big by themselves to be cost-effective, because even switching to a new but similar batch introduces overhead.

The introduction of a new product means development and testing of new configurations for the three lower software layers. Sometimes we need new software for our production system or even new hardware. The transition from product development to producing can be time-consuming, because in

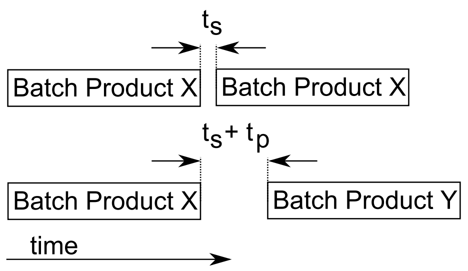


Figure 1.6: Batch switch overhead when switching to a batch of a different product

the development stage we did not use the production equipment that is used during the real production. So after development of a new product an extra step is needed to switch to production on the production equipment. To test this transition we also need to use the final equipment so it will not be available for normal production for some time.

Our investigation in properties of standard batch production automation can be summarized in four properties.

1. Huge batches for cost-effective production.
2. Small overhead introduced by batch switching of the same product.
3. Large overhead introduced by switching to another product.
4. Hard transition from product development to product production.

1.3.4 Shortcomings of standard manufacturing

After identifying the four properties of standard batch production, we mention six weak points of standard production.

1. Standard manufacturing is suitable for mass production, but small quantities or even single unique products according to user requirements are not advisable.
2. Standard manufacturing uses costly dedicated production machinery, that should be used at a high load to make it cost-effective.
3. Pipelined batch production is vulnerable for failures of manufacturing equipment.
4. Most standard manufacturing is push-driven. This can result in overproduction and waste of money, materials, resources and time.

5. The transitions from concept to product to mass production takes several steps and might take too long to be competitive in the market.

6. Most SCADA and MES implementations are not well suited for decentralization and could introduce a single point of failure

In the next section equiplet-based manufacturing will be introduced. The goal of this type of manufacturing is not to be a replacement of standard production. The goal is to offer a cost-effective solution for the situations where standard production is inadequate.

1.4 Equiplet-based production

The basic production platform for the new agile production system is the equiplet. The concept of an equiplet has been introduced by Puik, see (Puik and Moergestel, 2010).

Definition 3 (Equiplet)**.** *An equiplet is a reconfigurable manufacturing device that consists of a standard base system upon which one or more frontends with certain production capabilities can be attached.*

The frontends give the equiplet the possibility of production. It means that at the moment the frontend is attached to the equiplet, certain production steps can be accomplished. Every frontend has its specific set of production step capabilities. A picture of an equiplet with a delta-robot frontend is shown in Figure 1.7. A delta-robot is a special type of robot, that can perform fast pick and place actions. With this frontend the equiplet is capable of pick and place actions. A computer vision system is part of the frontend. Using this vision system the equiplet can localise parts and check the final position they are put in. The first field of application of the concept was building microdevices with a three dimensional structure (in contrast with the two dimensional approach used in placing electronic components on printed circuits). In this case one could think of steps to pick up a component and place it at a certain position (pick-and-place). Applying adhesive could be an option for this pick-and-place step. A local computer system is available on the equiplet for running control software depending on the applied frontends. Software configuration and management of an equiplet should be simple and easy.

As already mentioned, we call a collection of equiplets a production grid or in short a grid. The equiplets in a grid do not necessarily have the same frontend. Some frontends are unique, other frontends are available on several equiplets. The production process as a whole will dictate which frontends

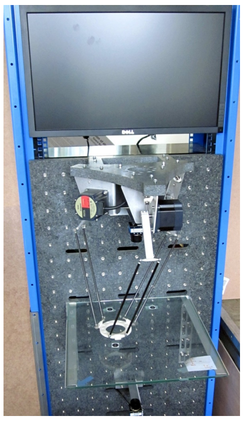


Figure 1.7: An equiplet with a delta-robot frontend

are needed. In the subsection 1.4.1 the properties of this agile production grid will be discussed, and in 1.4.2 enablers for this type of production will be mentioned.

1.4.1 Properties of equiplet-based production

Small-scale production To produce small-scale batches or even unique single products, standard batch production automation is inadequate because of the shortcomings mentioned and summarized in section 1.3.4. To make a single product, we should guide a product along the set of equiplets that offer the required steps for the product to be made. At the same time other products, requiring different sets of steps can also be made, assuming that access to the equiplets is adequately scheduled. When we use the concept of equiplets one should think of multiple production systems capable of producing many different products in parallel. We call this multiparallel production. At any moment we can start the production of a new or different product. This type of production does not introduce the overhead of batch switching and is capable of starting the production of a different product during the time another product is produced.

Time to market As mentioned in section 1.3.4 the transitions from concept to mass production might take too long. It means that the *time-to-market* might be too long. The time-to-market is the time that it will take for a newly developed product to go into mass production. From an economic point of view, this time should be minimal. Normally new products are developed at the Research and Development department. Then the production automation team will search for ways to make the transition to mass production. This phase is sometimes referred to as upscaling. To test this upscaling we also need to use the production floor equipment for test batches. The aforementioned steps are visualized in Figure 1.8.

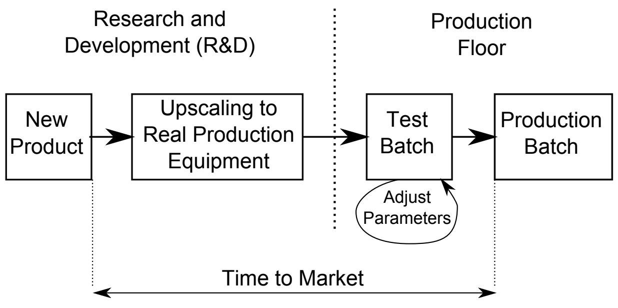


Figure 1.8: Steps involved for a new product

To make the transition from product development at the Research and Development department much easier, equiplets are used in the product development as well as in the final production process. So development, production automation and testing will be combined. The extra step of upscaling or adapting to the real production system is absent. The production is done by the same equipment and software as in the product development phase. This alleviates the aforementioned time-to-market problem (Puik et al., 2011). In Figure 1.9 this approach is visualized.

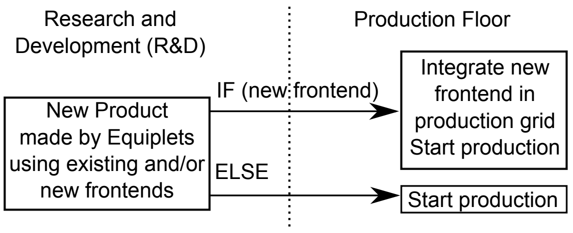

Research and Development (R&D)
Production Floor
New Product made by Equiplets using existing and/or new frontends
IF (new frontend)
ELSE
Integrate new frontend in production grid Start production
Start production

Figure 1.9: Developing a new product using equiplets

Reliability The grid production system is less vulnerable for failing production machinery, because equiplets can offer redundancy and the software architecture that will be described shortly is decentralised. System faults will not block other still operating parts of the grid.

1.4.2 Enablers for the equiplet-based production

Developments of the last few decades support the realization of this equiplet-based production. We mention five of them.

1. Internet and fast computer networks: our model is a distributed system where communication between the components should be fast and reliable.
2. Interactive web technology: this technology helps to involve the user of the product in the design and requirements phase.
3. Powerful micro-systems: all systems should have computing power to support a multitasking environment. This is not a problem any more in modern processor designs.
4. 3D printing: this technology enables making possibly unique parts for a product at low cost and low quantity.
5. Availability of cheap mobile robots: in the past attempts have been made to implement agile manufacturing, but the flexible transport infrastructure turned out to be a big problem. Nowadays, cheap mobile robot platforms are available that can be used to implement flexible transport.

1.5 Hardware and software infrastructure

This section invcestigates possiblities for hardware and sofwtare realisation of the agile manufacturing system. It contains four subsections.

1.5.1 Hardware. A global hardware overview.

1.5.2 Equiplets. The software for the equiplets.

1.5.3 Software infrastructure requirements.

1.5.4 Possible software architectures.

1.5.1 Hardware

Figure 1.10 shows the hardware setup of our production system. Only three equiplets are shown, but one could think of a grid of 64 or even more equiplets. The equiplets are connected by a standard (fast) network infrastructure based on switches or routers, thus offering a high bandwidth communication infrastructure. To monitor the grid we have a system that could be a standard industrial personal computer. This system will display information to a grid operator about the production processes in the grid. The storage of production information, necessary software components and the control of the several parallel production processes is done by supporting systems.

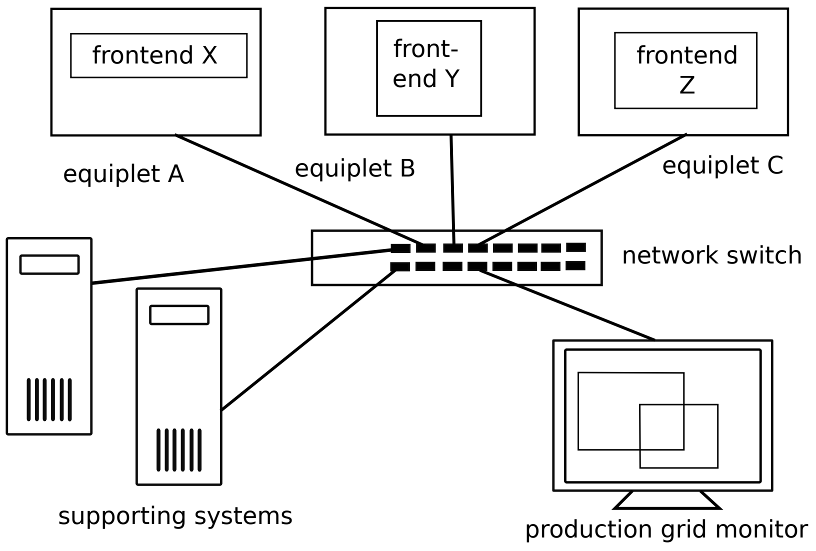


Figure 1.10: Hardware infrastructure

1.5.2 Equiplets

The equiplets have an onboard computer system that is running an operating system capable of network access. They also have the possibility to download extra software from the supporting systems. The basic software model for an equiplet is shown in Figure 1.11. In this figure we have the following software components.

1. Local Realtime OS. This is the operating system running on the equiplet hardware.

2. A network driver is needed to make a connection to the outside world.

3. The network can be used to load additional software parts from the supporting systems.

4. Every equiplet has some basic control software. This software is needed to enable the on-board communication bus that is used to connect to the equiplet frontend devices. For the communication with frontend devices a standard fieldbus, like CAN (Tindell et al., 1994) or Modbus (Rane, 2010) can be used. USB (Axelson, 2001) is also an option for connecting the frontend hardware, especially for devices like cameras. When a frontend is attached to the equiplet, this communication channel or bus can be used to discover the type of frontend and its devices and this gives information about extra drivers that are needed to control this frontend hardware.

All equiplets share this basic model. As can be seen in Figure 1.11 it is a layered software architecture, where the lowest layers are closely tied to the hardware. At the highest layer are the applications that run on top of an operating system.

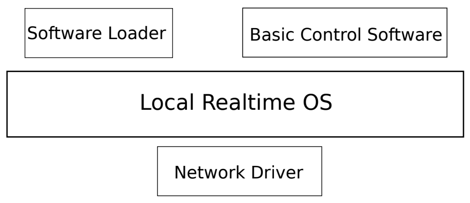


Figure 1.11: Basic software model for an equiplet

When we want to use an equiplet we have to take into account the software requirements for one or more frontends. So our basic model is not sufficiently equiped for this task. We need extra on-board software that is stored on the supporting systems and requested by a certain equiplet, depending on the needs of the local attached frontends. The on-board basic control software will discover what kind of extra software is needed. From the supporting systems two software components are requested.

- Equiplet Frontend Drivers. They are needed to actually make the connection to the equiplet frontend hardware. The need for these drivers is discovered by the on every equiplet available basic control software.

- Frontend specific software. It acts as the interface to higher software layers to control the frontend hardware using the aforementioned drivers.

The result is the software model of Figure 1.12. The model is shared for all our software architectures, but some architecture solutions proposed in subsection 1.5.4 will expand this software model for equiplets.

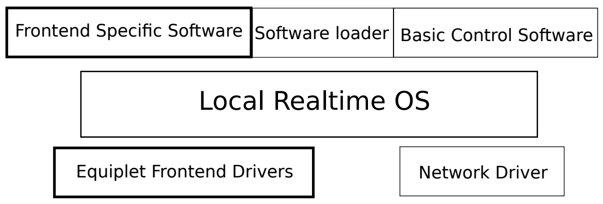


Figure 1.12: Expanded software model for an equiplet

1.5.3 Software infrastructure requirements

For adequate operating software infrastructure, we have imposed the following three requirements:

1. Efficient use of the equiplets. Equiplets are not expensive, but we need some kind of load balancing over all the available equiplets because this will lead to more parallelism and also prevents us from overusing a small subset of equiplets.

2. Small-scale production in parallel with each other should be possible. This is one of the goals for our grid production infrastructure.

3. Time to market of a newly developed product should be minimal. The transition from the development department to the production floor should be easy and simple.

1.5.4 Possible software architectures

To give an idea how the production of a single product will look like, we have plotted a path that the product will follow along certain equiplets (Figure 1.13). Such a path will be called a production path. On every equiplet in this production path, one or more production steps are done. A production step is an action performed on the product by a single equiplet. Some equiplets can perform a set of production steps. Note that equiplet A is visited twice in the production path depicted in Figure 1.13.

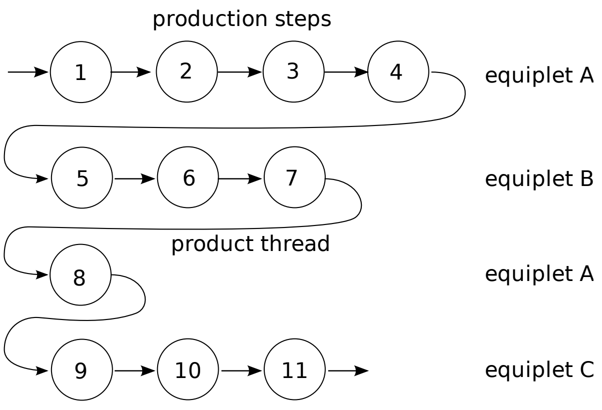


Figure 1.13: Selecting set of production steps

In Figure 1.14 we plotted the paths of three products and this results in a fabric of production paths along the available equiplets. The first production step is shown as circles with fat borderlines.

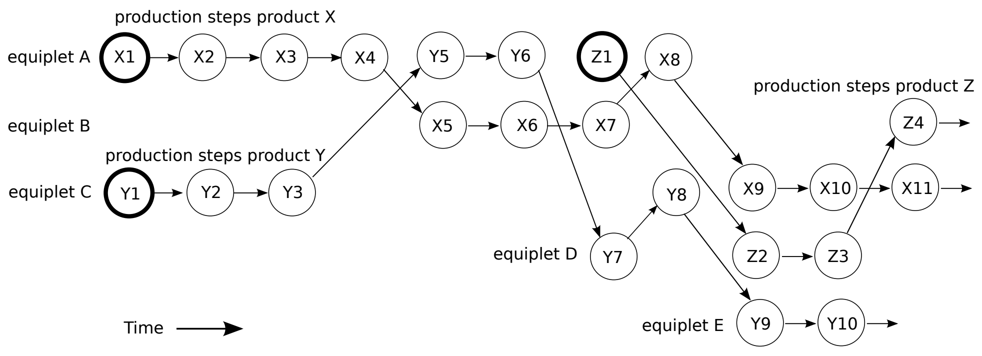


Figure 1.14: Product path fabric

To make a contrast with a possible batch process for producing a huge quantity of similar products as is the normal way of production, we also include Figure 1.15. The figure shows the approach that is used in batch environments for huge quantity production. We have a pipeline of production steps along the production platforms. In this concept all steps should take the same amount of time or all steps will adjust to the longest step time in the pipeline. This is not necessary in the equiplet approach, though Figure 1.14 might suggest that. In Figure 1.16 the situation for steps of different duration is shown. This approach fits in the equiplet-based solution, giving it an extra advantage over standard batch processing.

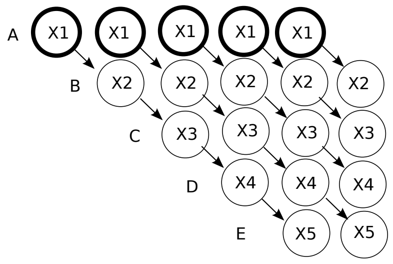


Figure 1.15: Batch production using a pipeline

To realize this we can apply two different models or architectures.

1 A centralized system, controlling the equiplets. The equiplets have the minimal software configuration as shown in Figure 1.12. Figure 1.17 gives an impression of this architecture. Compare this with a situation where we have a huge printer server where we connect a multiple of heterogeneous printers. Every printer has its own commands and capabilities.

2 A distributed system where the on-board software of the equiplet is

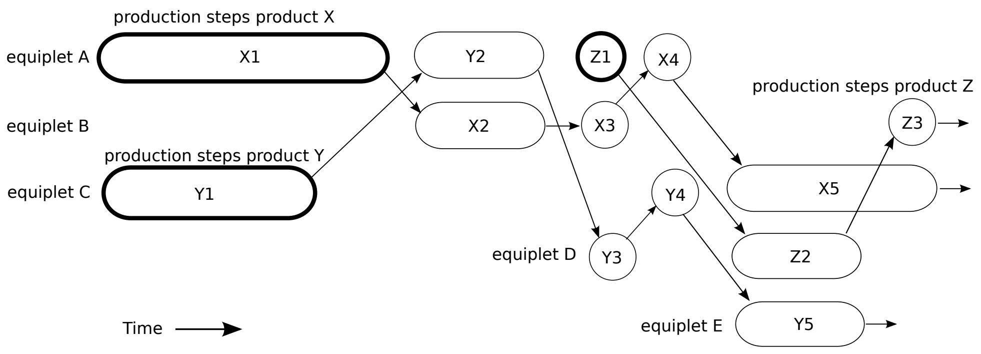


Figure 1.16: Product path fabric with varying step durations

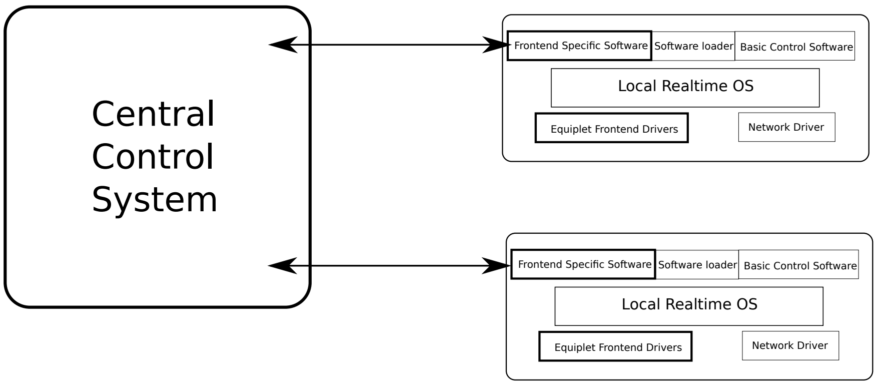


Figure 1.17: Centralized production control

enhanced with extra software so these equiplets can act as active nodes in this distributed environment. Instead of sending low level commands to the equiplets, we can now send a command to perform a certain production step. The on-board software is capable to perform such a production step.

Model 1 has the advantage that the equiplet software is kept to a minimum, but it results in a complex software system running on a central control system. The complexity is due to the requirement to run a part of the control system of all the equiplets. This software running on the central control system will be a single point of failure so we should have a failover system (see 1.7.2) for a complex piece of software which increases the complexity.

Model 2 has the advantage that the local software on the equiplets could continue the production process in case of temporary failure in the supporting systems, the network or in one of the other equiplets. This model also uses the available processing power in the equiplets and this will scale better when more equiplets are added. When we concentrate on this solution we can again choose from two possibilities:

2A A coherent distributed system where we have a central planning system that will send different software tasks to perform certain production steps to the equiplets. This situation is depicted in Figure 1.18.

2B A system composed by autonomous operating parts, working together on the production tasks as depicted in Figure 1.19.

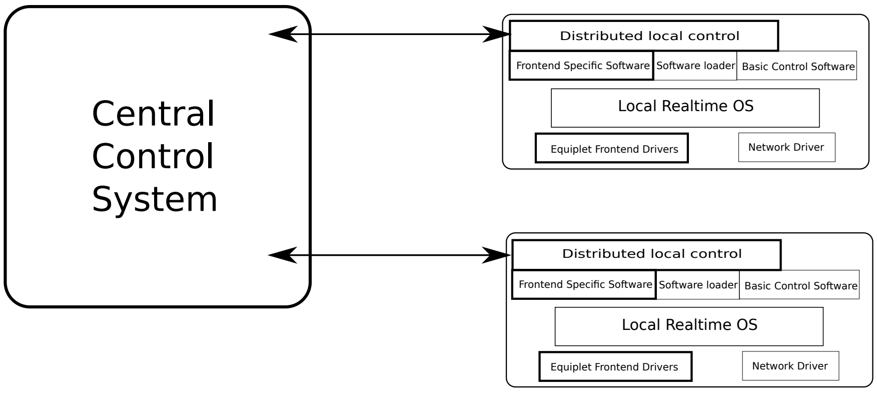


Figure 1.18: Distributed production control

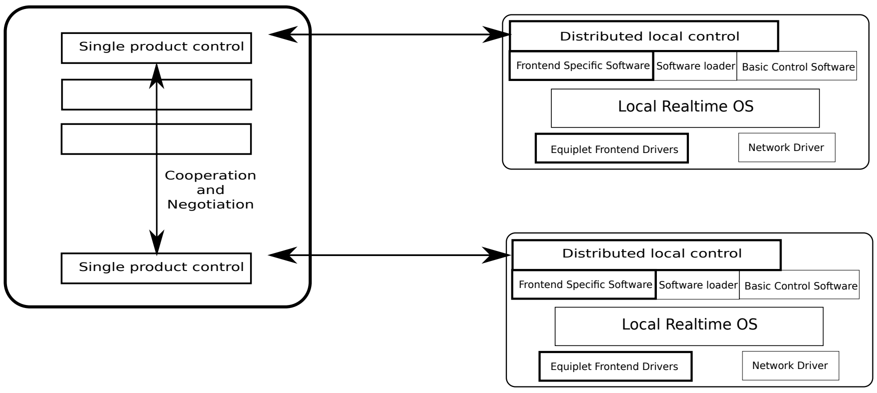


Figure 1.19: Distributed production control by autonomous subsystems

In case of a coherent distributed software system, we still need a complex software system in the central control to monitor all the equiplets and to decide what to do next when an equiplet completes its task. This central control system software system has at least three drawbacks:

1. Hard to maintain.

2. Single point of failure for the whole production process.

3. Not easy to adapt to new situations, in other words not agile.

When we look at our production system it is easy to separate the production of product X from product Y. This will result in a breakdown of complexity. An autonomous software component or subsystem should be able to make a single product. These product-making software entities only share the resources or environment, but each having its own path along the equiplets. Of course communication between these autonomous part is crucial. They have to negotiate about the use of the resources (i.e., equiplets) in the environment. So the system composed of autonomous parts seems to fit the requirements for the agile production system. Model 2B also fits the best in the grid concept, because autonomous software entities with communication capabilities can be easy implemented on a grid infrastructure. By using these autonomous entities we get a flexible, scalable and reliable production system (Paolucci and Sacile, 2005) as we will show in the next sections. The autonomous software entities must satisfy at least the following five requirements:

1. Autonomy. The software entity should be able to decide for itself what to do, however, this should be within the constraints and limits of the production system.

2. Cooperative. The entity should cooperate with other entities to solve conflicting situations. A solution should be a situation that is acceptable for all involved entities.

3. Communicating. To cooperate, the entities must have the capability to communicate with each other and the environment.

4. Reactive. Based on information about the environment or the situation an entity encounters, the entity should react.

5. Pro-active. Entities should take initiative and cope with different situations.

In the next section, we will introduce the agent concept and show that the agent-approach fits well in our proposed software architecture based on autonomy.

1.6 Agents

There are many definitions of what an agent is. We use here a commonly accepted definition by (Wooldridge and Jennings, 1995)

Definition 4 (Agent)**.** *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.*

In Figure 1.20 we depicted an agent in its environment. An agent is sensing the environment an can perform actions on the environment. As stated in the definition, the actions the agent performs depend on the design objectives.

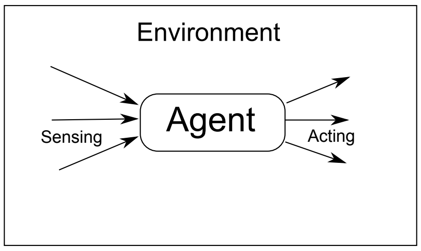


Figure 1.20: An agent in its environment

In Figure 1.20 the agent is a black box, so now we must take a look at the internal software structure of an agent. To do this we should first discuss the possible implementations of agents, but this is too broad a field to handle here. We will concentrate on the aspects that are important for our final software architecture.

1.6.1 Two types of agents

We will introduce the following two types of agents:

1. Reactive agents.
2. Reasoning agents.

A reactive agent senses the environment acts according to the information its get from this sensing. There is no internal state involved. A reasoning agent also senses its environment but does have an internal state. Depending on the sensing input and the internal state it will search for an action to perform, one could say it will reason for the action to perform. The sensing input will also change the internal state. A special type of reasoning agent is the so called belief-desire-intention-agent or BDI-agent. This type of agent has its backgrounds in the philosophy of Dennett and Bratman (Dennett, 1987)(Bratman, 1987). An internal schematic of a BDI-agent can be seen in Figure 1.21 (Wooldridge, 2009).

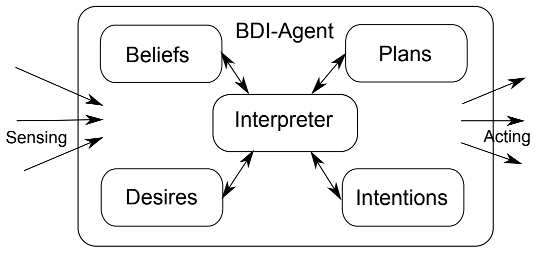


Figure 1.21: BDI-agent

The beliefs, goals, desires and intentions could be viewed as the mental states of a BDI-agent (Singh, 2003).

- From the inputs of its sensors the agent builds a set of *Beliefs*. Beliefs characterize what an agent imagines its environment state to be.

- *Desires or goals* describe the agent's preferences, that is what the agent prefers as its internal (agent) and external state (environment).

- *Intentions* characterize the goals or desires the agent has selected to work on.

An agent is equipped with a set of *plans*. These plans have three components.

1. The postcondition or *goal* of the plan. This is the agent's internal and external state that will be the result of carrying out the plan.

2. The precondition of the plan. This is the internal and external state that is present before the actions of the plan are carried out. Some of these states might be required for the plan to be carried out.

3. The course of actions to carry out. The actual actions contained in the plan.

An agent will deliberately choose a plan to achieve its goals.

1.6.2 Multiagent systems

A multiagent system (MAS) consists of two or more interacting autonomous agents. Such a system is designed to achieve some global goal. The agents in a multiagent system should cooperate, coordinate and negotiate to achieve their objectives. When we consider the use of a multiagent system we should specify abstract concepts such as:

- *Role*: what is the role of a certain agent in a multiagent system. Perhaps an agent has more than one role.
- *Permission*: what are the constraints the agent is tied to.
- *Responsibility*: this means the responsibility an agent has in achieving the global goal. A global goal consists in most situations of a set of sub-goals. An agent can be responsible for achieving one or more sub-goals.
- *Interaction*: agents interact with each other and the environment.

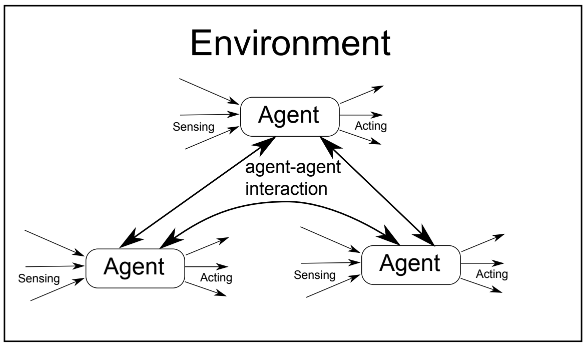


Figure 1.22: Multiagent system

When we map our five system requirements as mentioned in 1.5.4 to a MAS and single agent properties, we see there is a perfect match. The agent is autonomous, reactive and can be pro-active. In a MAS these agents can be cooperative and should communicate. What we must do to realize our software infrastructure is to define agents with a specific role in the system. These agents play their role and interact with other agents. We have interactions at two levels. First the actions of the agent within the environment and second the interaction between agents. When applied to our situation, these interactions and agent actions result in an agile production system as a whole.

1.7 Multiagent production system

To realize our system we define two agent roles. These roles are the main roles in the system and we should investigate if we need some extra minor roles. Every product is represented by a product agent and every equiplet with a certain frontend is represented by an equiplet agent. The product agent has the intention of the product being produced. The intention of the equiplet agent is to accomplish production steps. Every product is made according to a certain set of production steps, while every equiplet is capable to perform a certain set of production steps.

1.7.1 The agent-based automation piramid

The automation piramid for this design is given in Figure 1.23.

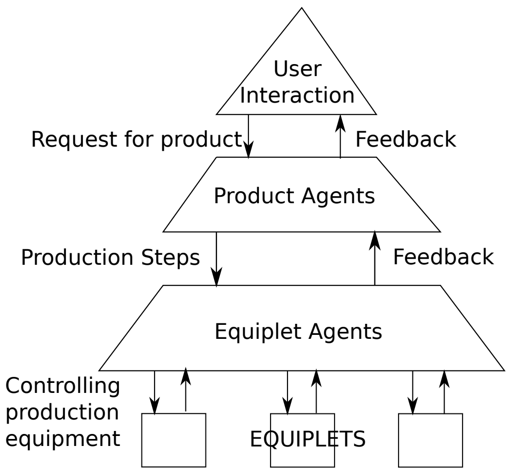


Figure 1.23: Multiagent production system

The product agent operates at what was in standard production called the MES layer, while the equiplet agent is more closely tied to the hardware. In this thesis a system will be described that is tightly coupled to one or more users by using a web-interface. This is one of the possibilities, but this approach results in a purely pull-driven manufacturing system.

1.7.2 Global description of the system

Now the system will be globally described. This description will reveal the problems to be solved and this will lead to the research questions related to the realisation of the system.

Agent platform

The two types of agents need to communicate and negotiate. To implement this, an agent-based platform must be built or chosen from existing solutions. Here one should keep in mind the remark made by Wooldridge in pitfalls in using agent technology (Wooldridge and Jennings, 1998), where the authors warn against reinventing and reimplementing an agent platform, because being a time-consuming and elaborate task the amount of work is underestimated. It is better to search for a platform with proven capabilities that fits the needs.

Planning and scheduling

Planning and scheduling is of an important part of the final realization. Figure 1.24 shows the way planning is accomplished by the participating agents. Equiplet agents need to communicate their set of production steps with the product agents. Based on this information the product agents choose the right equiplet agents to build the product they represent. Optimization is an interesting feature. For instance, in the example of Figure 1.24 we could have chosen to make step 1 on equiplet A, but this would result in an extra equiplet change for the product and its associated product agent.

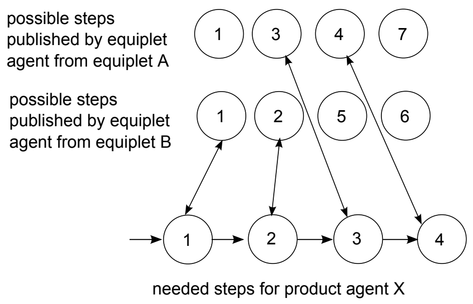

possible steps published by equiplet agent from equiplet A
1
3
4
7
possible steps published by equiplet agent from equiplet B
1
2
5
6
1
2
3
4
needed steps for product agent X

Figure 1.24: Selecting set of production steps

An important feature of a production step is the time it will take to complete. This could be a function of certain parameters for that step. Steps are published on a blackboard by the equiplet agents. The equiplet agent knows its capabilities and its building blocks, its building material etc. so it can announce its production steps. It is possible to add on the blackboard an overview of the time slots at which this step is available. It may happen that a production step is so popular that replication of this production step on another equiplet is advised. When we monitor the behaviour of the production grid it is easy to discover the bottlenecks. A product agent carries in its knowledge base all the production steps that are needed to complete the product together with the path (order) through these steps. XML might be suitable to express such a step overview.

The planning scheduling could be realized in a multi agent negotiation setup. The negotiation is entered by the product agents and is about taking possession of the production steps offered by the equiplet agents. One could search for a possible solution and afterwards start to bargain if the production as a whole could be optimized. Exchanging steps between product agents can be based on maximizing the sum of the utilities of the participating agents. So a product agent could decide to have a lower utility in favour of a much higher utility gain for another agent. To investigate this system, further simulations have been built and studied.

Transport during production

A flexible transport mechanism should be developed for the products to move along the equiplets. Another type of transport is bringing parts to the equiplets.

Benefits beyond production

After completing the product, a product agent is available that collected information about the design as wel as every single product step. An important aspect of our research is the investigation on what roles the product agent can play in other parts of the life cycle of a product.

Human interaction

Until now the equiplet agent has not been characterized. This could be a piece of software, but also a human-operated equiplet fits in this concept. In the latter case a piece of interaction software is needed to instruct and interact with the human operator and to participate in the multiagent-based production system. One could ask if it is now easily possible to let a software equiplet agent takeover the action of the human operator based on the production steps model we presented in this paper. A gradual takeover could be accomplished by learning the software agent step by step the production steps.

Reliability

For almost every production system downtime is expensive. So now we should focus on the reliability of our model. A list of hardware failures and possible solutions:

- Equiplet failure: by using redundancy, meaning that other equiplets offer the same production steps, the product agent can switch to another equiplet.
- Supporting system failure: this problem can be solved by setting up a networked failover system. In a networked failover system a networked device is replicated. One system is the primary network system, while the second system is a backup network system. The backup systems monitors the primary system and will take over the function of the primary system in case of failure. This takeover can be done transparently (Coile and Jordan, 2000). This technique is used to make systems more fault-tolerant. Failover is often a part of mission-critical systems that must be constantly available.
- Network failure: this problem can also be solved by using redundancy. Nowadays network components are not that expensive any more, thus making a redundant network not expensive.

Possible software failures:

- Equiplet agent failure.
- Product agent failure.
- Supporting software failure.

1.8 Research questions

The problem statement is: *How should the proposed production system be implemented and what could be the benefit to the whole life cycle of a product?* To answer the problem statement several topics should be dealt with in six different areas of research. These areas are discussed in Section 1.7.2. Each area results in one of the following research questions. Some research questions have a set of related subquestions.

RQ1 *How should we actually build the agents?*. Resulting subquestions are: What platform? Using an existing platform? How do the agents communicate?

RQ2 *What planning and scheduling system should be used?* Planning and scheduling is an important aspect of this new way of manufacturing. What are the constraints and how can we achieve optimisation?

RQ3 *How should the system be set up?* How to transport the products during manufacturing?

RQ4 *What are the possibilities and roles of the product agent when the product is finished?* What are the advantages to keep it alive? How to tie it to the product? What other roles can it play?

RQ5 *At what point are humans involved in this manufacturing system?* Here we should investigate the possibilities of the human interaction.

RQ6 *How to recover from errors and guarantee a reliable system?*

1.9 Research methodology

The research in this thesis can be qualified as applied research. In applied research, problems are solved by applying well known and accepted theories and principles. The methodology to come to the answers of the research questions is based on several well known research methods. A literature study has been the start, followed by investigation how methods and theories can be applied to find solutions for the research questions. For many research questions, software has been built to study the effect of the chosen solutions under different circumstances. Many simulations have been built to collect data about a specific problem. The simulations showed the strong and weak sides of the concepts. For other questions an analysis has been done that led to an architecture or solution that has been implemented as a proof of concept.

1.10 Overview of the thesis

In this chapter the basic ideas and background have been discussed. In the next chapters these ideas are more deeply investigated and elaborated. Chapter 2 is dedicated to the concept of production steps and product agents. A selection of available agent-platforms is made and based on this selection a simulation of the proposed manufacturing system will be presented. The answer to research question 1 is given. The simulation shows that the concept is feasible but an adequate and good performing planning and scheduling system is very important to make the manufacturing system work. This planning and scheduling is the subject of Chapter 3. This chapter will give an answer to research question 2. In Chapter 4 some ideas about the production infrastructure such as the internal transport are presented and explained.

Table 1.1: Relation research question and chapter

Chapter	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6
1						
2	✓					
3		✓				
4			✓			
5				✓		
6	✓	✓		✓	✓	✓
7	✓	✓	✓	✓	✓	✓

This leads to an answer to research question 3. In Chapter 5 the concept of the product agent is further elaborated and possibilities for the product agents in other phases of the life cycle of a product are investigated. This results in several case studies that serve as examples. Here the answer to research question 4 is given. Chapter 6 will give the results of the agile manufacturing grid. In this prototype a combination of a web-interface where the end-user enters his product requirements with the agent-based manufacturing system is realised. Research questions 5 and 6 are answered, but because this is the combination of previous results, additional information about previous research questions can also be found. Chapter 6 also introduces the manufacturing concept in a human-based working environment. This aspect also relates to research question 6. A final chapter is dedicated to conclusions and the relevance of the presented work in the thesis. Among the chapters some duplication is deliberately present to render the chapters more self contained. In table 1.1 an overview is given of the chapters in relation to the research questions.

1.11 Summary

In this chapter a global overview of the agile agent-based manufacturing system has been presented. The differences with standard production systems and special features have been discussed. The research questions to be answered are given as well as several definitions in the field of manufacturing and agent technology. As stated earlier the proposed system is not meant as a replacement for large scale production, but as a new paradigm for small scale agile production.

Chapter 2

Simulation of the production system

This chapter presents the first effort in realising the proposed agile manufacturing system. The concept as described in chapter 1 is elaborated in more detail and the definition of a production step is given and also discussed. Next the functional and technical requirements of the multiagent system and the selection of a suitable platform is presented. The realisation of a simulation of the concept where the equiplet agents are not yet connected to the real equiplet hardware, is given as well as results of the simulation. The goals of the simulation model were: testing the MAS functionality as a distributed system, testing cooperation between the agents, testing the migration of agents over the network.

Parts of this chapter have been published in the proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2011) (Moergestel et al., 2011) and the proceedings of the International Conference on Computer-aided Manufacturing and Design (CMD 2010) (Moergestel et al., 2010b).

2.1 Global roles of the Agents

To realize our agent-based system we have defined two roles ascribed to two types of agents. These roles are the main roles in the system.

Role 1 A product agent makes a certain product by searching and using a set of production steps.

Role 2 An equiplet agent Offers and performs production steps.

Every product is represented by a product agent and every equiplet is represented by an equiplet agent. A similar approach is also used by Jennings and Bussmann (Jennings and Bussmann, 2003), though there are some important differences that will be discussed in the section related work. The product agent has the purpose of the product being produced. The purpose of the equiplet agent is to accomplish the production steps. Every product is made by a certain set of production steps, while every equiplet is capable to perform a certain set of production steps. In other words: product agents know *what* steps are needed to make products while equiplet agents know *how* to perform these steps. In

2.2 Production step

This section starts with with a formal definition of what a production step is. In Section 2.2.1 the relation between production steps and the product agent is discussed. Having a limited set of production steps available, will also limit the possibilities of the manufacturing system. This will be discussed in Section 2.2.2. Sections 2.2.3 and 2.2.4 give additional information about production steps. Finally Section 2.2.5 shows the relation between frontends and production steps as well as the consequences for the equiplet software.

Definition 5 (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 realisation. The states of the product before and after the step is stable, meaning that the time it takes to do the next step is irrelevant and that the product can be transported or temporarily stored between two steps.*

An addition to this definition is that in the model presented in this thesis, the state of the equiplet is unaltered when the step is completed, apart of course from wear and inventory of raw material or parts. This means that the equiplet is ready for the next step to come. For example a pick and place action is considered one step.

Equiplets get a front-end. This is part of the initial grid hardware configuration. At that very moment it is clear what kind of production actions, resulting in production steps, they can perform. Let us assume that the grid offers a set S_{grid} of N production steps $\sigma_1 ... \sigma_N$. An equiplet offers a set of production steps that is a subset of S_{grid}. To make a product, a certain set of production steps should be available. This means that the set of needed production steps for a product is also a subset of S_{grid}. Because for a product the order of production steps is important, the product is characterized in its simplest form by a tuple of production steps: i.e. $< \sigma_4, \sigma_7, \sigma_2, \sigma_1 >$.

2.2.1 Product agent and steps

The ultimate desire or goal of a product agent is the product being completed. To do so, production steps need to be accomplished. Such a production step has a precondition, an action and an end-condition. The precondition of a step should be implied by the completion (end-condition) of the previous step or for the first step, the start situation. The precondition of the first step is TRUE if the product is going to be made. The end-condition of the step is the message from the equiplet agent that the step is completed. For the final step this means that the product as a whole is completed. Given these rules an agent will decide to start with the first step to change its belief about to state the product is in. This state will trigger the next step and so on until the final step is done. The action to perform is finding an equiplet agent to complete the step, visit this agent (residing in an equiplet) and collect production information in a product log.

2.2.2 Production constraint

It is possible to build a product X when the set of steps for this product P_X is a subset of the steps offered by the grid S_{grid}. This S_{grid} is the joined set of available M equiplets each offering a set $E_i : (i \in [1, ...M])$. This means that for every equiplet i:

$$E_i \subseteq S_{grid}$$

Assuming M equiplets, we have:

$$S_{grid} = E_1 \cup E_2 \cup ... \cup E_M$$

Thus:

$$P_X \subseteq E_1 \cup E_2 \cup ... \cup E_M$$

Or:

$$P_X \subseteq S_{grid}$$

Normally we do not need all the equiplet steps, so:

$$P_X \subset S_{grid}$$

Note that this is only a guarantee with regard to the existence of steps. It might be that these steps are not available for product agents due to the fact that they are already reserved by other product agents. It is also still possible that a product needs all equiplets, but not all production steps.

2.2.3 Classification of production steps

A production step could be seen as a statement in a computer program or more fundamentally as a instruction of a processor. To give the reader a feeling of what steps could be in the proposed manufacturing system we list four possible classes of steps.

1. Steps that alter the shape of a product by chemical, mechanical or other physical action (i.e., heating). Drilling, molding, cutting and milling are examples of mechanical shaping.
2. Steps that add one or more components to the product by inserting, gluing, welding, soldering, or another way of attaching. This includes steps that combine half-products, constructed by a separate step path.
3. Steps that inspect a product made so far.
4. Steps that will test a product.

2.2.4 Properties of the step concept

By defining steps we have a way to find out how we could construct a product. A step could be translated to instructions to a human operator or could be translated to low level actions of a part of the production system (i.e. an equiplet). This makes it possible to make a smooth transition from a hybrid system (Neef, 2006) where a human operator interacts with the equiplet agent to a complete software driven process. A step should be clearly defined (just like a statement or instruction in a computer program). A step can be performed if a set of preconditions is fulfilled. After the step has been completed, we have a new situation (postcondition) for the product agent.

In this chapter we do not yet go into detail about product path planning and scheduling though it is of course a very important part of the final realization. Figure 2.1 shows the way product path planning is accomplished by the participating agents for equiplet A and B in combination with a product agent X. As explained earlier, equiplet agents publish their set of production steps. The product agents choose the right equiplet agents to build the product they represent.

2.2.5 Production step frontend relation

At that very moment an equiplet gets its frontend, it is clear what kind of production actions they can perform. The production steps are linked to the frontends as well as to the products. So every specific frontend X is

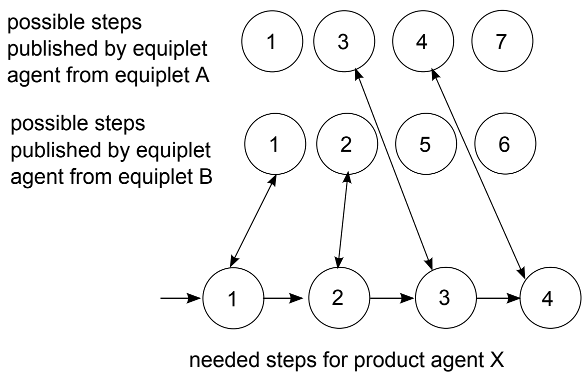

possible steps published by equiplet agent from equiplet A
1
3
4
7
possible steps published by equiplet agent from equiplet B
1
2
5
6
1
2
3
4
needed steps for product agent X

Figure 2.1: Selecting a set of production steps

connected to a set of production steps say F_X. For a product Y a set of production steps P_Y is needed to make that product. The control software or driver on the equiplet can be requested by the equiplet itself. The control or driver software is available on the supporting systems the equiplets are connected to. This software is not the equiplet agent but will enable the equiplet agent to act. The equiplet is part of the environment of the equiplet agent. Equiplet agent i has a set E_i of production steps it can perform. When an equiplet has only one frontend, which seems to be the default case, we have:

$$E_i = F_X$$

Because of the strong interaction with the equiplet driver software we expect the equiplet agent to run on the equiplet on-board system itself. These equiplet agents will publish their possible production steps in a global agent accessible space like a blackboard. Then they wait until a product agent wants to use its service.

2.3 Step path and product path

This section will define the concepts step path and product path. In Section 2.3.1 step path classes will be introduced and in Section 2.3.2 special cases of step paths are discussed.

Consider a situation where a product is built by 11 production steps. Let us assume that we have 3 equiplets A, B and C. Equiplet agent A offers production step set $E_A = \{1, 2, 3, 4, 8\}$, $E_B = \{5, 6, 7\}$ and $E_C = \{9, 10, 11\}$. The product agent representing our 11-steps product will choose equiplet A first to perform steps 1, 2, 3 and 4. Next equiplet B is used to perform steps 5, 6 and 7. Then we need again equiplet A for step 8 and finally equiplet C

for the last tree steps 9, 10 and 11. This so called *product thread* or *product path* is visualized in Figure 2.2.

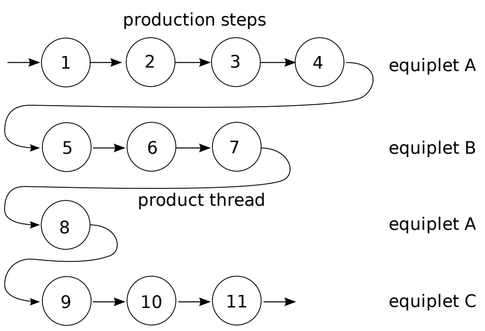


Figure 2.2: Product thread or product path

Definition 6 (Step path). *A step path is a path along a sequence of production steps or along a set of sequences of production steps that a product agent has to follow to complete a product.*

In the example that is visualised in Figure 2.2 where the product path is shown, another path emerges. This is the path along the equiplest involved. In case of the example it is a path from equiplet A to equiplet B, from equiplet B to A and finally from equiplet A to equiplet C. This type of path will be referred to as *product path.*

Definition 7 (Product path). *A product path is a path or set of paths along a sequence of equiplets that a product agent has to follow to complete a product.*

2.3.1 Step path classes

In the previous example of our 11-step product (Figure 2.2) the production steps are in line so our path is a single thread but one could think of other possibilities. Maybe the order of some production steps is irrelevant or a set of steps could be replaced by another set of steps. This is another advantage of the grid concept versus standard production. In standard pipeline-based production it is difficult to change the order of steps. Figure 2.3 shows some possibilities. When the order of subsets of steps is irrelevant, we start two or more parallel paths. These paths will join at some point. A special but often used case is a structure that starts with one or more parallel threads of steps. In this case it is sometimes possible to use real parallel production steps that are joined to complete the product. A special case is combining two half-fabricates.

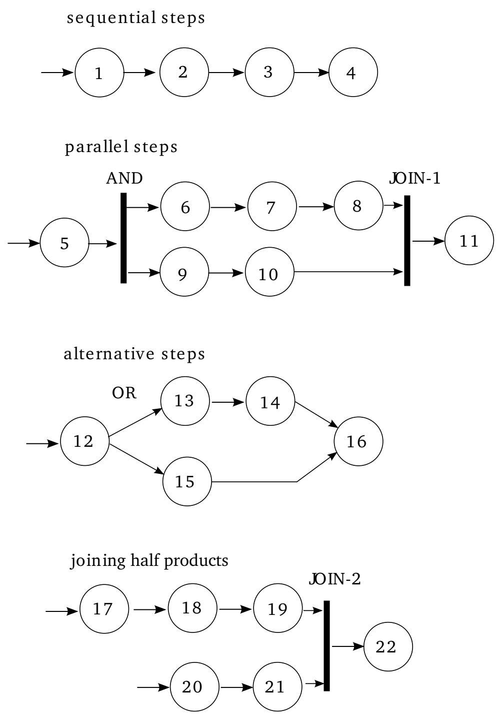


Figure 2.3: Different combinations of production steps

When these product paths as shown in Figure 2.3 are written in sets and tuples this results in:

- Single path: $< \sigma_1, \sigma_2, \sigma_3, \sigma_4 >$
- Parallel steps: $< \sigma_5, \{< \sigma_6, \sigma_7, \sigma_8 >, < \sigma_9, \sigma_{10} >\}, \sigma_{11} >$
- Alternative steps: $< \sigma_{12}\{< \sigma_{13}, \sigma_{14} > \vee < \sigma_{15} >\}, \sigma_{16} >$
- Joining half products: $< \{< \sigma_{17}, \sigma_{18}, \sigma_{19} >, < \sigma_{20}, \sigma_{21} >\}, \sigma_{22} >$

2.3.2 Special cases of step paths

When we introduced possible step combinations, we had the AND concept, where we could choose which branch to complete first (Figure 2.4). Because we start with a single production step and thus a single path we deal only with a single (incompleted) product after step 5, so we cannot use real parallelism in this case. The single product cannot be splitted to follow two different paths simultaneously. Because we want to treat the join as a special situation for real parallel manufacturing as will be introduced shortly, we re-map this AND construction to a choice of branches (OR) as can be seen in Figure 2.4

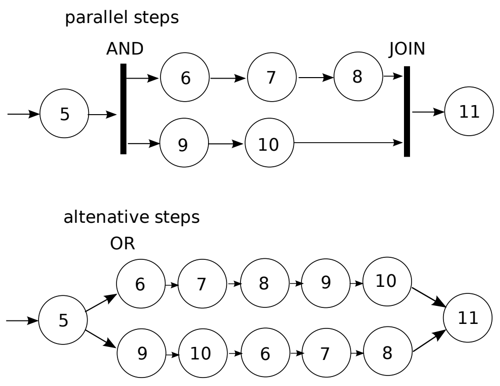


Figure 2.4: converting an AND to an OR

In formula, this means that we rewrite:

$$< \sigma_5, \{< \sigma_6, \sigma_7, \sigma_8 >, < \sigma_9, \sigma_{10} >\}, \sigma_{11} >$$

to

$$< \sigma_5, \{< \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10} > \vee < \sigma_9, \sigma_{10}, \sigma_6, \sigma_7, \sigma_8 >\}, \sigma_{11} >$$

. This means that the product agent can choose from two possibilities. In the planning it can select the possibility that turns out to be the fastest. It can have the other possibility at hand when a scheduling turns out to be infeasible.

Real parallelism can be achieved if the product has a tree structure as in Figure 2.5. On the left side of this figure, four incoming arrows each denoting the start of a production path can be seen. Each path will construct a subpart for the final product and because these paths are independent, these subparts can be made in parallel. At every join in the figure these sub-parts are combined to be input to the next step or steps.

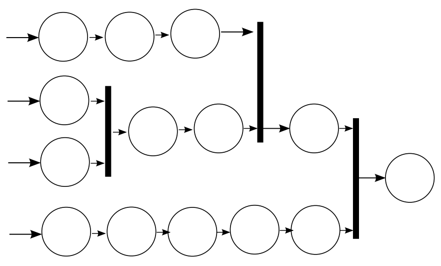


Figure 2.5: A tree of steps

In the situation of a tree structure a collection of product agents for a single product will be used. The start situation will be one agent, but this agent will spawn child-agents for the separate tuples. In Figure 2.5 we start at the righthand side and walk backwards to the beginning of the production on the left. At every join child-agents will be created. The parent will wait for its children to complete their subpart. This will be done for every join and will be repeated until the start of the tree structure. When all agents succeed in planning and scheduling, the production will start. At every join the child agents are absorbed by the waiting agent, taking over the collected information and continuing the path until the end is reached as a single agent. This situation arises many times, because most products consist of subparts (Figure 2.6). The product agent at the root of the tree will finally collect all information from its children. The effect of this decomposition of complex

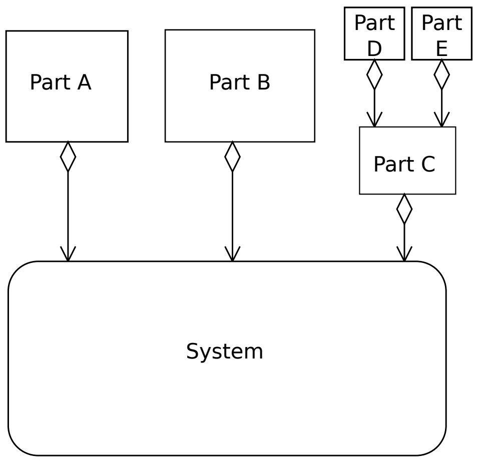


Figure 2.6: Product consisting of subparts

products is that every product agent only has to deal with a single tuple of production steps. The relationship of these product agents is the fact that they are working on the same product.

2.4 Considerations about the implementation

In this section we will summarize the global requirements for the equiplet agents (Section 2.4.1) as well as for the product agents (Section 2.4.2). Later on we go into more detail, especially the product agent will have much attention. In Section 2.4.2 implementation aspects of the product agent are discussed in relation with the following aspects.

1. A trusted product agent versus a foreign product agent.

2. Product agent for small batches

3. Living area for product agents.

2.4.1 Equiplet agent

Equiplet agents should:

- have a environment to run. For this agent running on the equiplet onboard hardware seems to be the best solution because this agent is closely related to the software that controls the equiplet hardware. Another advantage is that this results in a scalable distributed system. Every newly introduced equiplet will also deliver computing power for the equiplet agent to run.
- be capable to communicate its production steps. This is necessary to make the concept proposed here work. An equiplet agent that is not telling what it can do, will not get any requests from product agents.
- tell the product agent if a production step in combination with certain parameters is feasible. It turns out that a production step alone is still too generic. Before the decision to select a certain equiplet for a given step, the product agent will first check with the equiplet agent if it can perform the step with the parameters involved.
- tell the product agent how much time to reserve for a certain step with given parameters. The product agent needs to know or at least have good guess about the duration of a step. This information will play a role in the planning of the production by the product agent.
- be capable of human equiplet operator interaction if this is necessary. The model presented here is not restricted to autonomously operating equiplets, but can also be used in situations where a human operator is actually controlling the equiplet.
- have their software image residing on the central server to be downloaded at the required equiplet.

2.4.2 Product agent

Product agents should:

- have an environment to run. This environment also denoted as the living area of the product agent can be on the equiplet or a supporting system. This choice will be discussed further on.

- have a modular design. By having a modular design, all product agents can have the same global design, making the software maintenance easier.
- be mobile. This has to do with the fact that the living area may chance and the product agent has to move from one platform to another.
- collect product information during the production. This is a requirement for the product agent. Formerly production data were collected by the so-called MES-layer introduced in Chapter 1.
- know the production step sequence of the product they represent. This is the knowledge that a product agent starts with and will trigger the agent to reach its goal.
- be capable of negotiating to claim an equiplet to be used in the production process. First a product agent has to find an equiplet and then it should claim it. However other product agents might also be interested in that specific equiplet. A solution for this situation should be found.

Because of the strong interaction with the equiplet driver software we expect the equiplet agent to run on the equiplet on-board system itself. These equiplet agents will publish their possible production steps in a global agent accessible space like a blackboard(Corkill et al., 1987) or shared tuple space (Gelernter, 1985) or another way to publish globally for the MAS accessible data. Then they wait until a product agent wants to use its service.

When a product agent comes into existence in a grid environment, two possibilities exist for this situation. The agent was a mobile agent created somewhere and visiting the grid. This type of product agent is called a foreign agent and the discussion will be treated in the appropriate section. The other situation is that the agent is created in the MAS environment of the grid itself. This could be the case of an agent generated by a human operator using a web-interface to construct a desired product. This type of agent is created in an environment that is totally aware of the possibilities offered by that environment. The web-interface is equipped with tools to create a product that fit neatly with the production-steps available in the grid. It means, when the web interface is capable to generate a virtual product, the grid is capable to produce that product. The situation for a foreign agent is completely different. The foreign agent should be handled with care as it is not allowed to do things that might tamper the correct functioning of the grid. It means that this type of agent should be put in a sandbox environment and only connect to the grid by a mediator agent that is part of and trusted by the grid MAS.

Trusted product agent

The first thing that should be done by this type of product agent is check if the required production steps together with the required parameter are feasible. For every step involved this is an action consisting of two phases:

- Check the availability of a step and find out if the equiplet offering the step is really available. This means it is up and running.
- Check if at least one member of the set of equiplets offering the step is accepting the parameters required for the step.

After checking this for all needed steps it is known that the manufacturing is feasible within the grid. When it comes to the availability of parts needed for production three approaches for the manufacturing could be considered:

1. At the beginning of the manufacturing a box with all components needed for manufacturing is made. It means that the equiplets themselves do not need to keep these parts available. This situation come handy when there are many types of parts that can be handled by different equiplets. Such a situation will be called the pure building box approach. A filling station should be available at the beginning of the manufacturing, but it is not necessary to fill all components at once. A bit by bit approach is possible, but it introduces extra overhead.
2. The equiplets themselves offer parts during manufacturing. The equiplet will be kept responsible to keep parts in stock and to order new parts if a low water mark of local availability is reached. This situation is called the distributed parts approach.
3. There is a third hybrid situation where a building box is used, but the equiplet is also responsible for some additional material. This is the case of the situation where a glue dispenser is used by an equiplet and the equiplet is responsible for adhesive being available.

During production, this type of product agent could be in full control of the production itself. The planning, scheduling and production is directed by the product agent. The equiplets are the production machines. Should production fail at any point in the production tree or line, the problem should be reported to and handled by the product agent.

Foreign product agent

A foreign agent could be prepared in a distant unknown environment. This type of agents should be handled with care to prevent it compromising the grid production system. This can be achieved to give it a thread of execution in a sandbox container, where the only possibility is to talk with a mediator agent. However, it is also possible and in a certain sense desirable to keep this agent somewhere in cyberspace where it has been created.

The mediator agent takes over the role of product agent comparable to a trusted agent. There is however a significant difference. Being a representative of a foreign agent, this foreign production agent could possibly be not aware of the capabilities of the production grid. That is the reason why a thorough examination of the requested production steps and its parameters is required. This is a task for the mediator agent that actually replaces the product agent. After checking the required capabilities of the grid, the mediator agent will act the same way as a trusted agent. At the end of the production, the production information will be handed over from the mediator to the foreign agent.

Product agent for small batches

A special attention should be paid to the situation where not a single product but a small batch of similar products is required. In that case, the following considerations should be taken into account:

- If the products are completely similar, in the MAS subsystem an optimisation could take place by issuing requests one time for a multiple of products. The scheduling could also be done on the whole batch, however, this could be complicated in case of interference with other production requests, so in this case a batch planner and scheduler could be advisable.

- If it is the case that products in a batch are not similar, two possibilities arise:

    1. All products have the same sequence of production steps (tuples are the same), but the parameters are different for at least one step and possibly for more or even all steps. An example could be a batch of similar products but with different colours, so the paint colour parameter is the only difference.
    2. Products have a different sequence of production steps.

In the second case the approach of using a product agent for every single product seems adequate. The first case is comparable with the situation of a batch of similar products and a batch planner and scheduler can be used.

Living area for product agents

When we want to implement a system based on our hardware we have two choices for the product agent:

1. A product agent can be mobile and migrate to the onboard equiplet hardware to interact with the equiplet agent as in Figure 2.7.

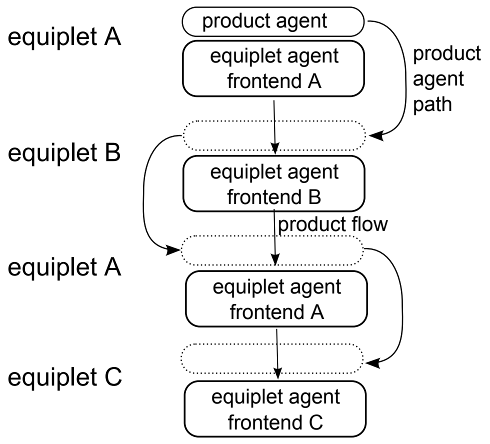

equiplet A
product agent
equiplet agent frontend A
product agent path
equiplet B
equiplet agent frontend B
product flow
equiplet A
equiplet agent frontend A
equiplet C
equiplet agent frontend C

Figure 2.7: Mobile product-agent

2. A product agent can run on the supporting systems and communicate with an equiplet agent by using the network address of an equiplet where the equiplet agent resides (Figure 2.8).

The size (in embedded technology also known as the footprint) of the product agent is important to decide what solution is the best. The advantage of a mobile agent is better as to the use of the distributed processing power. When all product agents reside on the central server, scaling to a big production grid will result in an overloaded supporting system. This supporting system could by duplicated to overcome this problem. We also have to consider the amount of data communication when all these mobile agents are travelling along the equiplets (Figure 2.9a). When this give rise to a loss of bandwidth of the network infrastructure a solution could be the use of a modular agent design. So most parts of the product agent software can reside on the equiplet and only the part representing the essential product information travels over the network (Figure 2.9b).

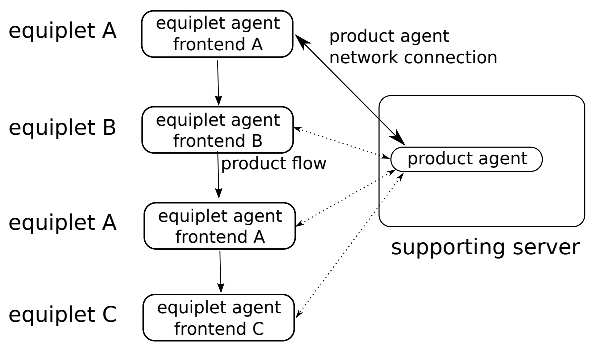

equiplet A
equiplet agent frontend A
product agent network connection
equiplet B
equiplet agent frontend B
product flow
product agent
equiplet A
equiplet agent frontend A
supporting server
equiplet C
equiplet agent frontend C

Figure 2.8: Product agent on supporting system

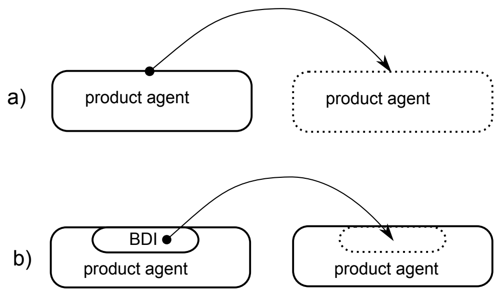


Figure 2.9: Mobile agent implementations

2.5 Interagent communication

The communication system of the agents has three requirements.

1. It should be based on the fact that agents act autonomously (Section 2.5.1).

2. If possible, it should adhere to standards (Section 2.5.2).

3. The implementation in the production domain has its requirements (Section 2.5.3).

2.5.1 Communication system

In multiagents typically three types of communication are often used (adopted from Wooldridge (2009)).

- Peer-to-peer communication where every agent is capable to send a message to every other agent. Each agent should know the names and addresses of all agents.

- Blackboard-based communication, where agents share a medium where messages can be published and read by other agents.
- Federation-based communication, where the agents of the MAS are divided into groups or federations, and in every group there are facilitator agents responsible for the communication between the federations.

In autonomous agent-based communication, these types of communication are asynchronous, meaning that the communication system does not depend on obligatory responses.

2.5.2 FIPA

As stated in Section 2.1, the product agent knows what production steps need to be done. The equiplet agent knows how to accomplish certain steps. To make a cooperation between these agents possible it is necessary to define a common language or ontology for these two agents. An important standard for interagent communication already exists. The FIPA organisations states on its website: *FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.* The FIPA standards are widely accepted and used in the field of agent technology. This is the reason why the MAS that will be built should be based on these standards.

2.5.3 Production domain communication

Let us consider what it means to give a production command to the equiplet. This command is actually a request to perform a production step. The product agent knows the status of the product in statu nascendi before and after the production step has been done. When these two states are communicated to the equiplet agent, the first thing the equiplet will do is to check and confirm the begin status. After that it is the responsibility of the equiplet agent to realize the desired end-status. By this the product agent does not have to take in account complex movements of several parts to the wanted position. Only the end-position is relevant to the product agent. The problem to avoid collisions during movements of parts is left to be solved by the equiplet agent. Let us show in more detail what it means for the information that should be transferred from product agent to equiplet agent. In Figure 2.10 a situation is shown of two parts A and B that must be glued together. Both parts are in a container. The product agent knows which parts are involved and their positions in the container. This is considered temporary data for manufacturing and should be stored with the product agent. It also knows at what position the parts should be glued together. The command to the equiplet agent includes information about the following nine items.

1. Adhesive type.
2. Where to add the adhesive.
3. The amount of adhesive to use (defaults to equiplet setting if omitted).
4. The adhesive dispenser type.
5. Which parts are involved.
6. Location in the tray of the parts.
7. Position of the parts after the step is completed.
8. Location in tray where the assembled (sub)product should be stored.
9. Hardening time.

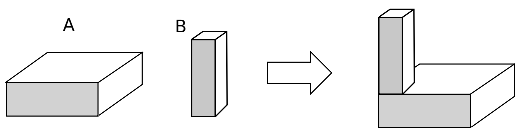


Figure 2.10: Glueing part A and B

The problem to be solved is: how do we communicate this information to the equiplet agent? For every step one or more parts are involved, Every step has a certain action to be done. Some steps require a certain resource (adhesive, paint). This can be represented in an XML-specification. The FIPA standard allows for embedding XML in its messages.

```
<step>
<parts>
<part>identifier, position</part>
<part>identifier, position</part>
</parts>
<action>
adhesive, where to dispense
part(identifier) part(identifier) (coordinates)
</action>
<resource>
adhesive(<type>, <amount>)
</resource>
</step>
```

From the example given here the equiplet needs to know about the shapes of components A and B to perform the required action. It should be capable to identify the surfaces where these components should be glued together. This is the *how*-part that was assigned to the equiplet agent. Finally, to realise the system, a start will be made that is simple and gradually increase the complexity by adding knowledge about more complex parts to the production system. This is the bottom-up approach used in our research.

2.6 Product agent functional requirements

The functional requirements describe what the product agent shoul do. This section has two subsections. Subsection 2.6.1 describes the functional requirements in the production environment. Subsection 2.6.2 descibes additional requirements for the product agent in the life cycle of a product.

2.6.1 Production environment

The primary role of a product agent during production is taking care of the production of the product. The responsibility of the product agent is building the product. To fulfil this role there are certain steps that should be take into account:

- From the design a global and abstract description of the product being built is available in combination with the actions involved. This is a collection of so called production steps. The first thing to do for the product agent is planning the production. To do so the steps must be parametrised and specified. Next the product agent should find out which equiplets offer the required steps. The next thing to do is investigating if the needed steps can really be performed by the equiplet. It means that the product agents asks the equiplet agent if it can perform the requested step with the given parameter set. When all steps are investigated a set of possible plannings can be made. To make the production planning an overview of steps by equiplets is needed.

- Optimization of the planning. In this phase the product agent will try to look for the most optimal planning. Taking into account the steps that can be performed in a row on a certain equiplet and the travelling time for a product between the equiplets. This will result in a list of plans sorted according to optimization.

- Scheduling the production plan. Starting with the most optimal production plan, the product agent tries to schedule this plan. What is needed here is a blackboard. Being a passive entity the blackboard is less vulnerable to malfunction than an active entity like a program or agent. The blackboard shows the availability of the equiplets. Every equiplet entry has several possible values:

  - $ID > 0$: in use or reserved by product with identifier ID.
  - 0: available for this timeslot.
  - -1: unavailable because of planned work.
  - -2: unavailable because of error or state (maintenance, switched-off).
  - -3: unavailable because of unknown reason.

- Starting the actual production according to the planning and the scheduling. During production, the equiplets will inform the product agent about the steps performed, thus building a unique product log.
- Error recovery. An important aspect of the production is that there can be failures. The product agent should have ways to recover from failures or to handle them in a neat way.
- Final inspection of the product (quality control). This should be done by using a special equiplet, because the product agent may be living in cyberspace and has no direct connections to the real production environment. It would be very nice if a kind of 3D match check could be done. Thus the checking equiplet builds a 3D image giving the product agent the opportunity to check the final product.

To fulfil its role, the product agent should communicate with the production infrastructure. This can be accomplished by running the agent on the same platform as the software for production, but also a networked connection fits this goal well.

2.6.2 Life cycle

When the product agent continues to play its role in the life cycle of the product as will be introduced in Chapter 5, we might need additional functional requirements.

- Mobility. Mobility is necessary if the product agent must be embedded in the product itself. At first there is no product at all and when the product is completed, the agent should be moved to hardware in the product.

- Wireless connectivity. A product that is freely moving around with an embedded product agent can use wireless technology to enable the product agent to communicate with the outside world.
- Reliable storage of important data. A product agent is responsible for the logging of all kinds of important data. This data should be stored for later use. The agent might use external storage and use the aforementioned wireless connectivity to transfer data.
- Role-changing. In Chapter 5 situations are described where a product agent will act in different roles. A role-changing mechanism should be provided.
- Extra security. This has also to do with a product agent embedded in a device that is already in use. Tampering with data and misleading the product agent should be prevented.
- Error and crash recovery. A product agent should recover from error situations and system crashes.
- Redundancy. In some situations redundancy might help to recover from serious crashes where the product agent lost critical information or even the product agent itself is lost.

2.7 Product agent technical requirements

The technical requirements state how the agent software should be built. In the MAS-based system, the realisation will focus on the platform where the agents will live. One of the pitfalls in using agent technology is developing a platform from scratch (Wooldridge and Jennings, 1998). To avoid this pitfall, research has be done to select a platform that seems fitted to our functional requirements.

2.7.1 Platform requirements

The agent platform has the following requirements:

- Preferably open source with an active community.
- Based on a widely accepted and standardized language.
- Capable to dynamically create agents.

- Capable to interact with software outside the platform runtime system.
- Support for a multiplatform networked environment.
- Support for FIPA.

2.7.2 Platform selection

Based on the requirements of 2.7.1 the following list of possible candidates has been built.

- Jade: is a widely accepted Java Agent DEvelopment framework. JADE is middleware system running in a distributed environment.
- Spade: SPADE (Smart Python multiAgent Development Environment) is a Multiagent and Organizations Platform based on the Python prgramming language and the XMPP/Jabber. The FIPA-ACL messages are embedded in XMPP. XMPP stands for Extensible Messaging and Presence Protocol and it is an open standard communication protocol for message-oriented middleware based on XML.
- 2APL: 2APL (Dastani, 2008) is the follow-up of 3APL (Hindriks, 2001). 2APL stands for A Practical Agent Programming Language and is pronounced double-a-p-l. 2APL is a BDI-based modular agent-oriented programming language that supports an effective integration of declarative programming constructs such as belief and goals, and imperative (style) programming constructs such as events and plans. 2APL is based on the Jade platform.
- Jadex: a short overview is given by (Braubach et al., 2004): *Jadex is a Java-based FIPA-compliant agent environment, and allows to develop goal-oriented agents following the BDI model. Jadex provides a framework and a set of development tools to simplify the creation and testing of agents.* This platform is actually based on Jade, so it provides the tools already offered by Jade.
- Jason: an introduction is given by (Bordini et al., 2005). This BDI-based implementation is a BDI-implementation based on Agentspeak(L). Agentspeak(L) was initially introduced by Rao and Georgeff as a language for specifying BDI agents (Rao and Georgeff, 1996).

When these platforms were investigated, it turned out that for Java software developers, Jade was the easiest accessible solution. The amount of time to get used to the platform was considerably lower than the other possibilities. In the research environment, groups of computer science students with a programming background in Java, work for at most four months on the project, so a learning time of several weeks to get used to the platform is not desirable. The question was if the support for BDI was such an important issue that this would overcome the steep learning curve for the other candidates. For the MAS needed for the research a stable and widely supported platform turned out to be sufficient. The agent specification can use concepts like, roles, beliefs and intentions that can also be implemented using Jade. The selected platform for the MAS was Jade. Six advantages are as follows.

1. The simulation is a multiagent-based system. Jade provides most of the requirements we need for our application like platform-independence and interagent communication.

2. Jade is Java-based. Java is a versatile and powerful programming language.

3. Because Jade is Java-based it also has a low learning curve for Java programmers.

4. In this first approach at least the equiplet agents are not that intelligent that we need special multiagent environments. The product agents should be capable to negotiate to reach their goals. Jade offers possibilities for agents to negotiate. If we need extra capabilities, the Jade platform can easily be upgraded to an environment that is especially designed for BDI agents like 2APL or Jadex (Bordini et al., 2005). Both 2APL as well as Jadex are based on Jade but have a more steep learning curve for Java developers.

5. Agents can migrate, terminate or new agents can appear.

6. Jade is widely accepted and has an active developers community.

The Jade runtime environment implements message-based communication between agents running on different platforms connected by a network. The communication is asynchronous meaning that the sender is not blocked until there is a reply from the receiver. Asynchronous communication is typical for agents since it preserves autonomy. In Figure 2.11 the Jade platform environment is depicted.
The Jade platform itself is in this figure surrounded by a dashed line. It consists of the following five components.

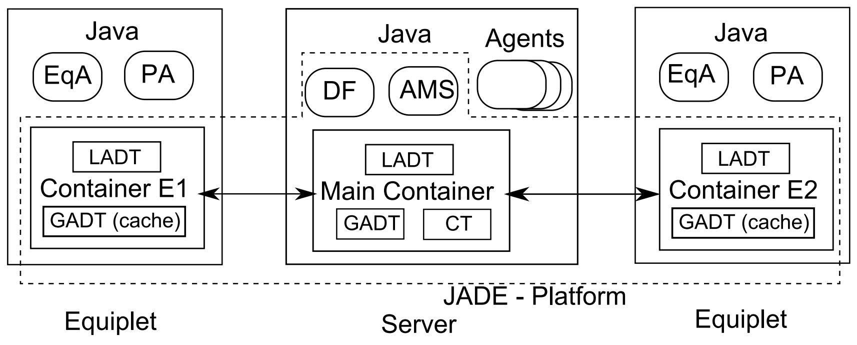


Figure 2.11: The Jade platform

1. A main container with connections to remote containers (in our case E1 and E2, representing equiplets).

2. A container table (CT) residing in the main container, which is the registry of the object references and transport addresses of all container nodes composing the platform.

3. A global agent descriptor table (GADT), which is the registry of all agents present in the platform, including their status and location. This table resides in the main container and there are cached entries in the other containers.

4. All containers have a local agent descriptor table (LADT), describing the local agents in the container.

5. The main container also hosts two special agents AMS and DF, that provide the agent management and the yellow page service (Directory Facilitator) where agents can register their services or search for available services.

In Figure 2.11 the product agents and equiplet agents are already denoted by PA and EqA, where the equiplet hardware is used for container E1 and container E2.

2.8 Jade-based simulation

The simulation presented in this section was based on the technical and functional requirements for production and assumptions as discussed in this chapter.

2.8.1 Basic assumptions

Following is a list of basic assumptions for the simulation.

- Equiplet agents are not yet connected to equiplet hardware. This means that no actual manufacturing will be done, but a production step is simulated by a delay. After a certain time, the equiplet agent will inform the product agent about the completion of the production step.

- The environment is a networked distributed system, where agents can migrate. This opens the opportunity to test the situation where agents travel over the network. The hardware used is a set of standard personal computers connected by a LAN.

- Transport from equiplet to equiplet is not yet covered, thus a product agent can switch without any delay from one equiplet to the next one.

- The product has a single tuple of steps to perform. This is a situation that is easier to implement as a first try.

- An equiplet agent can offer a set of steps. This is one of the principles of our manufacturing paradigm.

- Product agents select the equiplets and take the first opportunity for a step when available. In Chapter 3 a more detailed discussion of which step to take, will be presented.

- Product agents have a release time and a deadline. The release time is the moment that a product agent comes to life. From that moment on, the product agent begins to achieve its goal: the making of a product. The deadline is the time at which the product should be completed.

- In case of an infeasible scheduling, a simple negotiation scheme should be used to solve the problem when it only concerns a negotiation between two product agents. A more detailed solution will be presented in Chapter 3.

- The simulation should be based on a so-called scenario file that describes the simulation constraints. This way, several different scenarios can be easily investigated without changing the simulator.

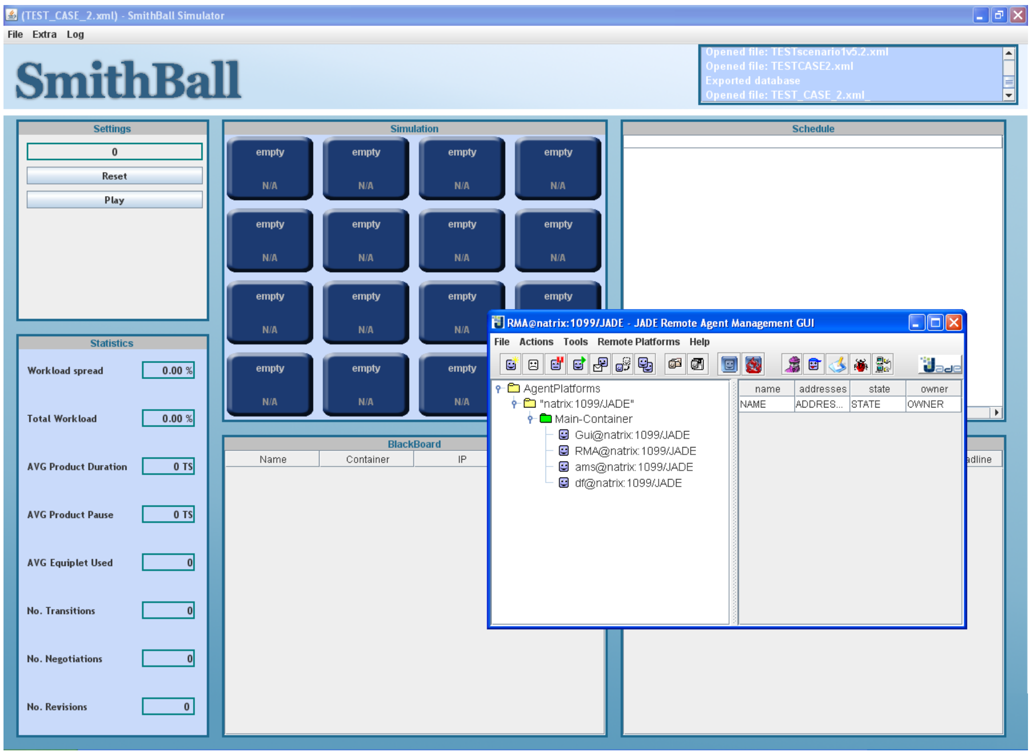
- A GUI should be available to observe the working of the simulation in realtime.

2.8.2 Implementation

For the implementation we set up a hardware infrastructure as in Figure 1.10. For testing our concept, the equiplet agents are not yet connected to the equiplet frontend hardware, so there is no real production and the production steps are virtual steps. This means that equiplets offer production steps and when asked to perform a production step they will enter a timing loop, faking a real production step. We used Jade (Bordini et al., 2005) as a platform. The reasons for choosing Jade are explained in Section 2.7.2. The equiplet agents (EqA) and product agents (PA) run on top of this platform (see Figure 2.11). Due to performance reasons, we decided to use our database-based blackboard instead of the DF of the Jade platform. DF turned out to be too slow for our implementation.

A: User interface and setup

The user interface of the simulation program is shown in Figure 2.12. Sixteen slots are available for equiplets and a play-button will start the actual simulation when a scenario has been chosen under the File menu.



Opened file: TESTscenario1v5.2.xml
Opened file: TESTCASE2.xml
Exported database
Opened file: TEST_CASE_2.xml

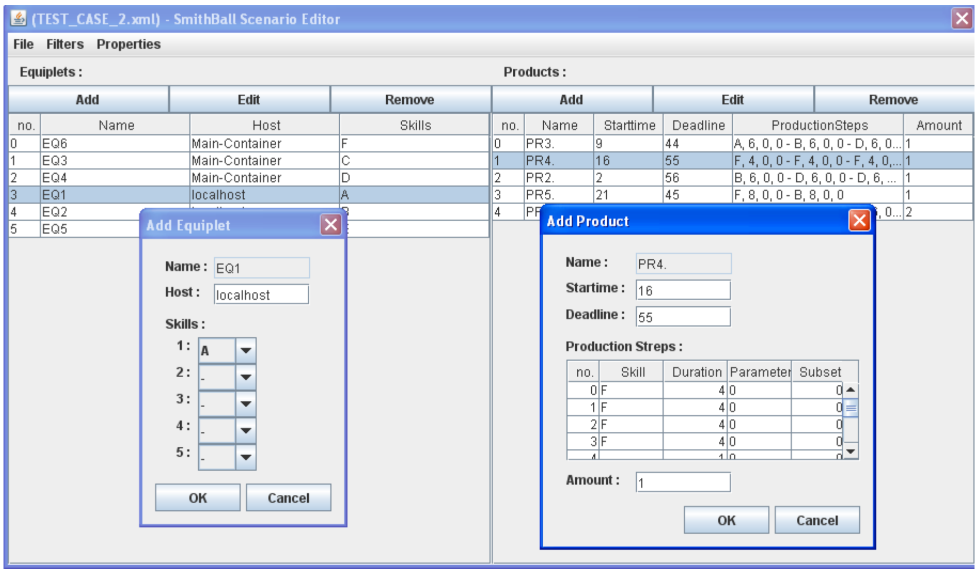
Settings: 0

Workload spread	0.00 %
Total Workload	0.00 %
AVG Product Duration	0 TS
AVG Product Pause	0 TS
AVG Equiplet Used	0
No. Transitions	0
No. Negotiations	0
No. Revisions	0

```
AgentPlatforms
  "natrix:1099/JADE"
    Main-Container
      Gui@natrix:1099/JADE
      RMA@natrix:1099/JADE
      ams@natrix:1099/JADE
      df@natrix:1099/JADE
```

name	addresses	state	owner
NAME	ADDRES...	STATE	OWNER

Figure 2.12: GUI of the simulator


(TEST_CASE_2.xml) - SmithBall Scenario Editor

File Filters Properties

Equiplets :

no.	Name	Host	Skills
0	EQ6	Main-Container	F
1	EQ3	Main-Container	C
2	EQ4	Main-Container	D
3	EQ1	localhost	A
4	EQ2		
5	EQ5		

Products :

no.	Name	Starttime	Deadline	ProductionSteps	Amount
0	PR3.	9	44	A, 6, 0, 0 - B, 6, 0, 0 - D, 6, 0...	1
1	PR4.	16	55	F, 4, 0, 0 - F, 4, 0, 0 - F, 4, 0,...	1
2	PR2.	2	56	B, 6, 0, 0 - D, 6, 0, 0 - D, 6, ...	1
3	PR5.	21	45	F, 8, 0, 0 - B, 8, 0, 0	1
4	PR			, 0...	2

Add Equiplet

Name : EQ1
Host : localhost
Skills :
1 : A
2 : _
3 : _
4 : _
5 : _

Add Product

Name : PR4.
Startime : 16
Deadline : 55
Production Streps :

no.	Skill	Duration	Parameter	Subset
0	F	4	0	0
1	F	4	0	0
2	F	4	0	0
3	F	4	0	0

Amount : 1

Figure 2.13: GUI of the scenario editor

To use the simulator, a scenario editor is included. The editor is shown in Figure 2.13. The user can add products as well as equiplets. For every equiplet, a container can be specified, making it a distributed system. This can be done by filling in a computer name in the filed Host. The steps the equiplets can offer can also be specified (as skills, represented by capital letters). For the products, the set of production steps can be specified. In the simulation, the duration of the steps is specified in the configuration of the product. It has to do with the fact that the equiplet agent is still a stripped-down implementation. An option available under the Properties menu is to make agents visible during the simulation. This option can be selected in the scenario editor by setting GUI to TRUE. We will now take a closer look at the agents.

B: Equiplet agent

Every equiplet agent is residing in the on-board hardware of the equiplet. Equiplet agents control the equiplet hardware and perform production steps. To do this, the equiplet agent waits for a product agent to contact the equiplet agent. The product agent will inform the equiplet agent by a message of the production step to perform. The equiplet agent will inform the product agent about the actions taken and, if this is relevant, the building material used. As already mentioned, the equiplet agent is not yet connected to the equiplet hardware and performs only virtual production steps and informs the product agent about successful completion of the steps.

The architecture is a layered system. This approach for our software model makes it easily maintainable, expandable, testable and modular. The agent layer contains the main software part of the agent and an asynchronous event handler is used for communication events like messages from and to other agents. It also contains a graphical user interface or GUI that can be disabled. This GUI is nice for testing purposes, because it can be used to check the behaviour or internal state of the agent during the simulation run. The data layer contains a database handler. In our system the database is used as a global publishing system, resembling a blackboard. The equiplet agent has two behaviours or roles. It publishes the possible production steps in its role as publisher. After publishing it will enter the role of executor. In this role it enters a waiting state for product agents to arrive. As we will see in the next subsection, the product agent is a mobile agent that will visit the equiplets according to its scheduling in its role as walker. If a product agent arrives it will send a message to the equiplet agent and this equiplet agent will start producing, thus actually performing the requested production step(s). When the production step is finished, it will inform the product agent about the production but keeps its role as executor, waiting for the next product agent to arrive.

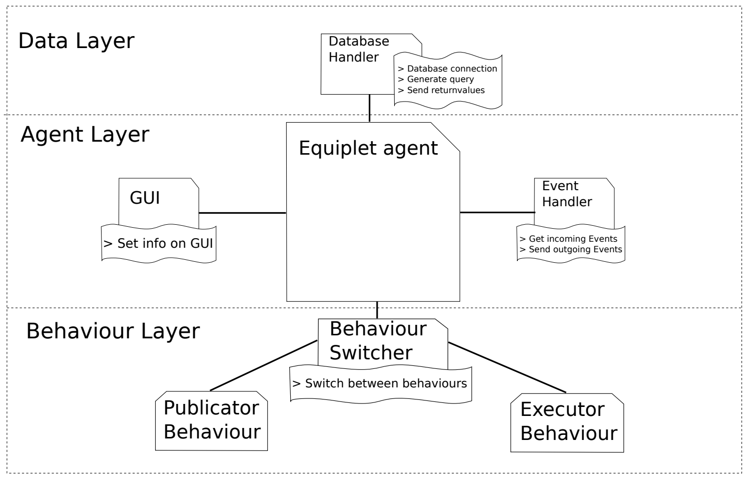


Figure 2.14: Equiplet agent architecture

C: Product agent

The product agent will search for production steps published by the equiplet agent on the blackboard. It will schedule the production and thus claim production steps on certain equiplets. Next it will travel along the equiplets asking the equiplet agents to perform the needed steps until the product is finished. The product agent has the same layered architecture as the equiplet agent, extended with an extra layer that is used for scheduling and optimization. The agent layer has the same components as the agent layer in the equiplet agent,

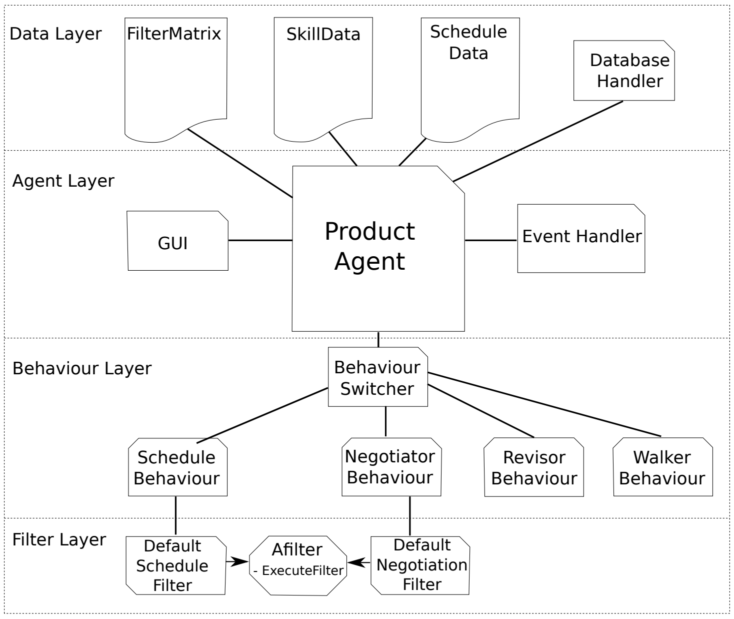


Figure 2.15: Product agent architecture

The configuration of both the product agent as well as the equiplet agent is given as a set of parameters during startup. The steps are stored in SkillData. ScheduleData contains the production scheduling. FilterMatrix is used for data that is used during production scheduling.

The product agent comes to life at the central server and will try to schedule its production steps in its role as scheduler. It will check the blackboard to see if all needed steps are available to complete the product before the deadline. If all steps are available the scheduling will succeed and the production will start by the product agent switching from scheduler to walker behaviour (it walks along the equiplets).

The product agent is a mobile agent, walking from equiplet to equiplet as depicted in Figure 2.9. During walker behaviour, the agent is still open for messages from other product agents. If scheduling fails due to the fact that one or more production steps are not available, the product agent will check the blackboard to see which product agents it should negotiate with. It builds a list of agents to negotiate with and it will ask these production agents with walker behaviour to negotiate about a certain production step.

If a walker agent is willing to revise its scheduling it will enter the role of reviser and try to revise its scheduling thus making place for the step needed by the agent with the failed scheduling. Several scenarios have been built to test this concept and it works to our expectation (Moergestel et al., 2010b). When the simulator is running the screen looks like Figure 2.16. In this case the GUI option for all agents is TRUE so small windows representing the agents are visible.

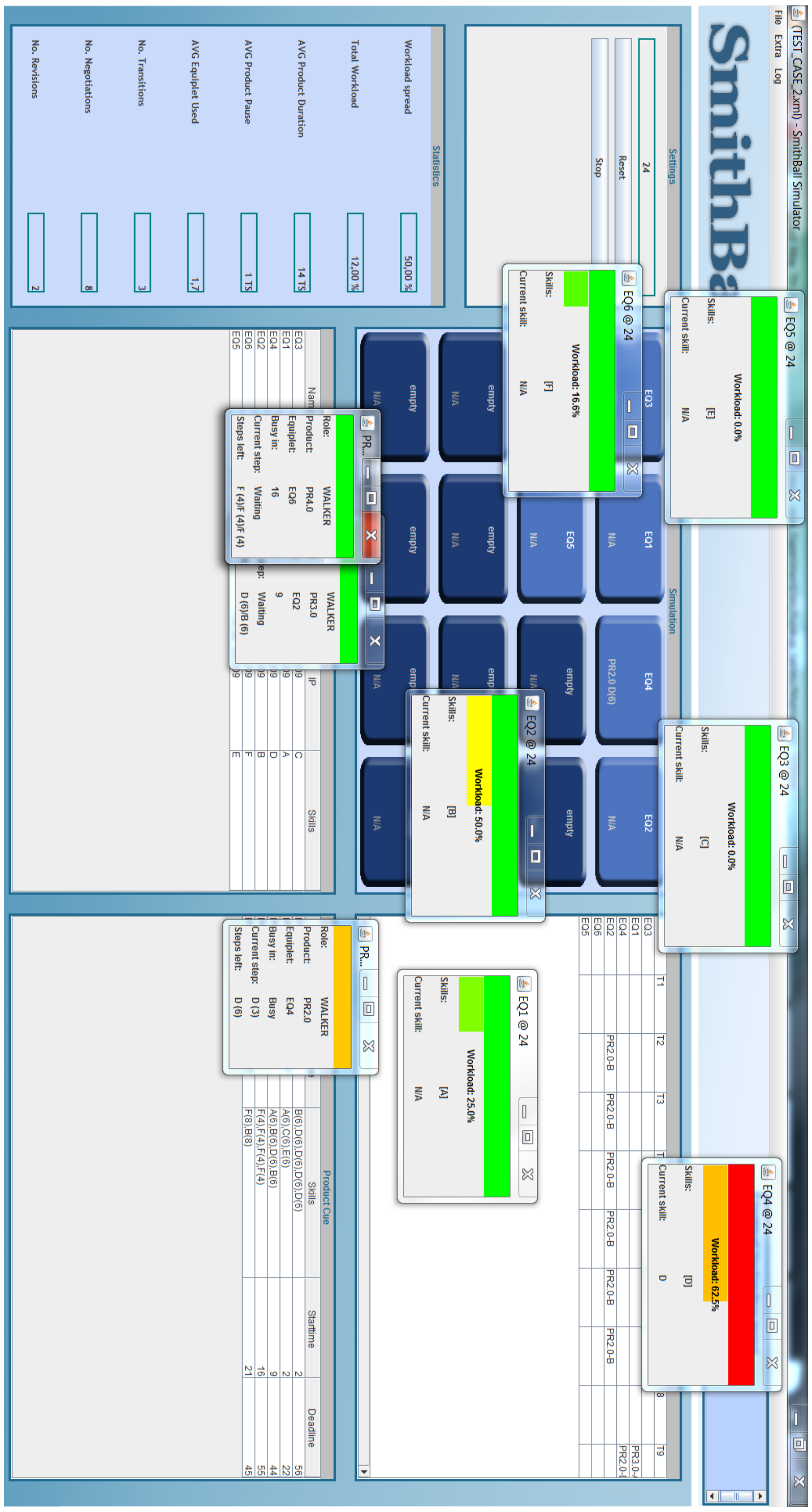
To understand the behaviours or roles of this agent we can look at the way scheduling is performed and how it starts the production. In Figure 2.17 there are two product agents involved. Agent 1 is just starting, Agent 2 is already busy with production. Only relevant states are represented in the diagram.

If all steps for Agent 1 are available the scheduling will succeed and the production will start by the product agent switching from scheduler (S) to walker (W) behaviour (it walks along the equiplets). During walker behaviour, the agent is still open for messages from other product agents. If the scheduling fails, then the Agent 1 switches to negotiator (N) and asks walker agents to negotiate about giving up steps at a certain time en thus revising their schedule. Agent 1 chooses a list of interesting walkers to negotiate with. It will ask a walker if it is willing to negotiate. The response can be yes, no or maybe (in this case the negotiator could try again later). In case the answer is no the negotiator will try the next walker. If the answer is yes, the walker (Agent 2) will switch its behaviour to reviser (R) and see if there is a possibility to change its claimed step without losing its successful scheduling. It will inform Agent 1 about success or failure. In case of success Agent 2 will adjust its new scheduling in its role of scheduler and continue its role as walker. Agent 1 will also return to its role of scheduler. And if it has a feasible scheduling it will switch to walker. When scheduling including negotiating fails the agent will report failure. Agent 2 will only revise its scheduling if there is a new feasible scheduling.

The approach proposed here works in simple situations. However when many product agents are involved and the load of the equiplets is over 50% some problems pop up:

- An agent that is asked to abandon a step at a certain timeslot is not capable of finding a feasible solution by itself, so it could ask other agents whether they can offer a new slot by replacing a given slot to another position. This would result in an avalanche of negotiations and should be avoided.

- If we avoid the previous problem, in a heavy loaded grid, the amount



Workload spread 50,00 %
Total Workload 12,00 %
AVG Product Duration 14 TS
AVG Product Pause 1 TS
AVG Equiplet Used 1,7
No. Transitions 3
No. Negotiations 8
No. Revisions 2

EQ5 @ 24
Workload: 0,0%
Skills: [E]
Current skill: N/A

EQ6 @ 24
Workload: 16,6%
Skills: [F]
Current skill: N/A

EQ3 @ 24
Workload: 0,0%
Skills: [C]
Current skill: N/A

EQ2 @ 24
Workload: 50,0%
Skills: [B]
Current skill: N/A

EQ1 @ 24
Workload: 25,0%
Skills: [A]
Current skill: N/A

EQ4 @ 24
Workload: 62,5%
Skills: [D]
Current skill: D

PR...
Role: WALKER
Product: PR4.0
Equiplet: EQ6
Busy in: 16
Current step: Waiting
Steps left: F (4)|F (4)|F (4)

Role: WALKER
Product: PR3.0
Equiplet: EQ2
Busy in: 9
Current step: Waiting
Steps left: D (6)|B (6)

PR...
Role: WALKER
Product: PR2.0
Equiplet: EQ4
Busy in: Busy
Current step: D (3)
Steps left: D (6)

Figure 2.16: The simulator, showing several agents

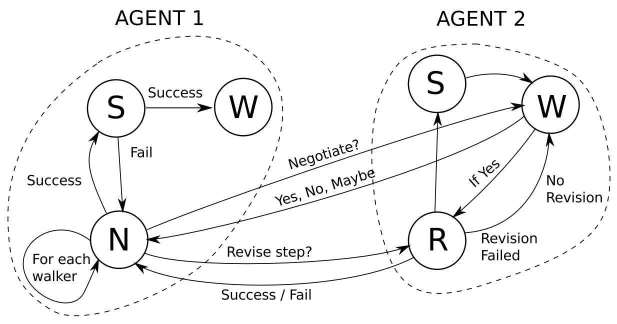


Figure 2.17: Negotiating between product agents

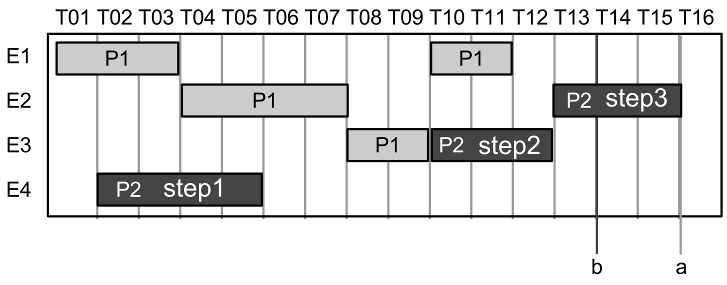


Figure 2.18: Missed deadline

of failures increases when a load of about 50% is reached. This situation compares with the first implementations of ethernet (Metcalfe and Boggs, 1976), where collisions decrease the performance of the network.

- Every time an agent encounters a scheduling problem for a certain step. it has to enter the procedure described already again. A better solution could be an approach that solves the scheduling problem for a product as a whole. This will be a topic of the next chapter.

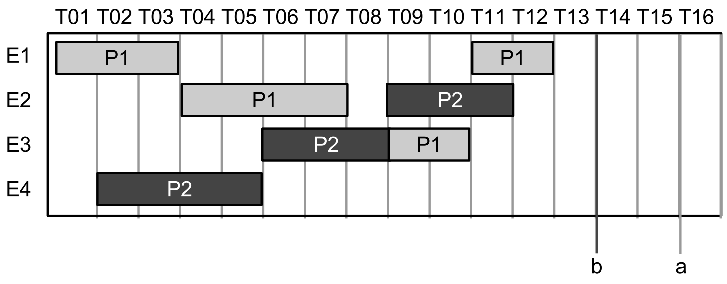


Figure 2.19: Feasible schedule

2.9 Related Work

Using agent technology in industrial production is not new though still not widely accepted. Important work in this field has already been done. Duffie and Piper (1986) introduced the non-hierarchical control approach where agents represented physical resources, parts and human operators. Parunak (1998) introduced Yet Another Manufacturing System (YAMS). It applies a contract net method in a hierarchical manufacturing model. The shop floor is represented by agents. Paolucci and Sacile(Paolucci and Sacile, 2005) give an extensive overview of what has been done in the field of agent-based agile manufacturing. Their work focuses on simulation as well as production scheduling and control. The main purpose to use agents in (Paolucci and Sacile, 2005) is agile production and making complex production tasks possible by using a multiagent system. Agents are also introduced to deliver a flexible and scalable alternative for MES for small production companies. The roles of the agents in this overview are quite diverse. In simulations agents play the role of active entities in the production. In production scheduling and control agents support or replace human operators. Agent technology is used in parts or subsystems of the manufacturing process. Agent technology in the manufacturing workflow control is treated in the work of(Montaldo et al., 2002). Their work describes the MAKEIT approach. We on the contrary based the manufacturing process as a whole on agent technology. A survey given by (Leitão, 2009) discusses agent-based manufacturing control. It compares the traditional approach with agent-based solutions. It also tries to explain why the agent-based approaches are not fully adopted by the industry. Bussmann and Jennings (Bussmann et al., 2004)(Jennings and Bussmann, 2003) used an approach that compares to our approach. The system they describe introduced three types of agents, a workpiece agent, a machine agent and a switch agent. There are however important differences to our approach:

- The production system is a production line with redundant production machinery and focuses on production availability and a minimum of downtime in the production process.
- The roles of the agents in their approach are different from our approach. The workpiece agent sends an invitation to bid for its current task to all machine agents. The machine agents issue bids to the workpiece agent. The workpiece agent chooses the best bid or tries again. This type of cooperation is well known in agent technology as the contract-net protocol. In our system the negotiating is between the product agents.

- They use a special infrastructure for the logistic subsystem, consisting of conveyor belts controlled by so called switch agents. In their implementation, bidirectional transport is possible and a product can change its production path. In our situation an even more agile transport system will be needed. This will be discussed in Chapter 4.

We have developed a production paradigm based on agent technology in combination with a production grid. This model uses only two types of agents and focuses on agile multiparallel production. The work of Bussmann is already implemented in real production environments, while our work is still based on prototypes.

In agent-based manufacturing the term holon is often used. While agent technology emerged from the field of computer science, the concept holon has its origin in computer integrated manufacturing (CIM) (Leitão, 2009). The concept was proposed by Koestler (1969). Parts of a system can be autonomous and stable on their own, but by cooperation they may form a bigger whole. This bigger whole could again be a part of an even bigger whole. A holon is both a part and a whole. A holon can represent a physical or logical activity. In the domain of manufacturing this can be a production machine, a production order or a human operator (Bussmann and McFarlane, 1999). Agent technology can be used to implement a holon. Fisher (1999) uses a holonic approach for manufacturing planning and control. His work is based on the use of the Integration of Reactive behaviour and Rational Planning (InterRRap) agent architecture (Müller, 1996). Agents represent the holonic manufacturing components, forming a multiagent system. A difference with our approach is the design phase. A holonic design starts with identifying the holons and next it will map this to agents or other software entities like objects. A holon itself could by its nature be a multiagent system.

Our implementation is based on the way that humanity has made products for centuries. Some humans have the capability to perform some actions for production or making something, while others have the need for a certain product or thing. The latter group of people knows what should be done, but lacks the skills to perform the needed actions, so they will search for people having the set of required skills. By the nature of the production grid, consisting of equiplets having what could be called skills, this approach seems to fit seamlessly to our needs. This is the reason we used agent technology itself as a starting point thus skipping the holonic design phase. The design and implementation of the production platforms and the idea to build a production grid can be found in Puik (Puik and Moergestel, 2010).

2.10 Conclusion

The multiagent-approach for the equiplet-based production system is a feasible one. The Jade agent-platform performs well in the simulation environment. The product agents are capable to select the equiplets and can also communicate with each other to solve an infeasible scheduling. The scheduling type used however needs to be improved, because the scheduling problem solving is only limited to ad hoc solutions. In the next chapter a better solution for the scheduling problem will be proposed.

2.11 Summary

In this chapter the concept of the production step has been defined and several scenarios for combining these steps in product paths were presented. The selection of an agent platform has been done based on the functional requirements of the multiagent production system. To investigate this platform, a simulation has been built where the product agents operate in a distributed environment, cooperating with dummy equiplets agents, that are not connected to equiplet hardware, but can be distributed over the network.

Chapter 3

Planning and Scheduling

This chapter is dedicated to planning and scheduling in the production grid. An agent-based scheduling system will be proposed and results from simulations of different kinds of scheduling methods will be discussed. Planning is described in Section 3.1. Sections 3.2-3.5 are dedicated to scheduling. Section 3.6-3.8 present the scheduling simulation and its results. Section 3.9 turns the focus on scheduling for batches. The chapter ends with an overview of related work, a conclusion and a summary.

Parts of this chapter have been published in the proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2011) (Moergestel et al., 2011) and the proceedings of the International Conference on Intelligent Agent Technology (IAT-2012) (Moergestel et al., 2012).

3.1 Path planning

A product agent should plan a path along the equiplets. This path will depend on the product steps to be done and the equiplets involved. In this section, a graph-based model of the production is presented (3.1.1) followed by matrix-based representations (3.1.2). These matrix-representations are derived from the graph-based model. In Section 3.1.3 optimisation is explained and Section 3.1.4 is dedicated to complex step paths.

3.1.1 Graph representation

The production system can be represented using special classes of graphs, such as a bipartite graph and a tripartite graph.

Definition 8 (Bipartite graph). *A bipartite graph is defined as a triple $G = (V_1, V_2; E)$ where V_1 and V_2 are two disjoint finite sets of vertices and $E =$*

$\{(i_k, j_k) : i_k \in V_1, j_k \in V_2; k = 1...d\}$ *is a set of edges.*

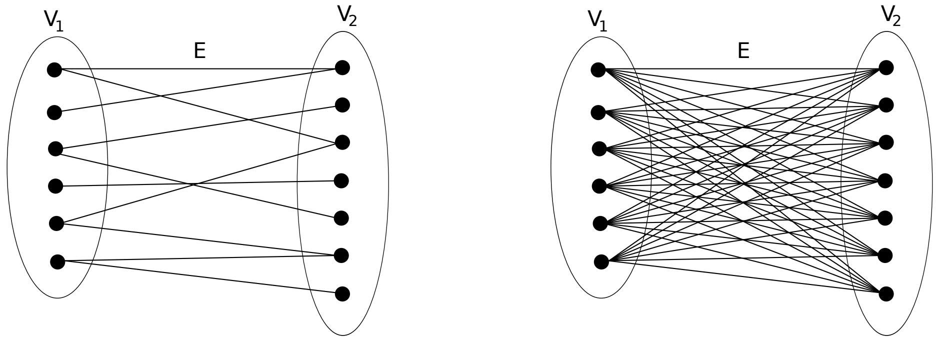


Figure 3.1: Bipartitegraph and complete bipatitegraph

If all vertices of V_1 have edges to all vertices of V_2 the graph is called a complete bipartite graph. If $|V_1| = m$ and $|V_2| = n$ this is denoted by $K(m, n)$.

Definition 9 (Product set)**.** *Let V_1 and V_2 be two sets. The product set of V_1 and V_2 is the set of all ordered pairs (i, j) such that $i \in V_1$ and $j \in V_2$. This is written as $V_1 \times V_2$*

By definition of the product set, it means that a bipartite graph is complete if $E = V_1 \times V_2$. The product set is also called Cartesian product.

Definition 10 (Tripartite graph)**.** *A tripartite graph is defined as a quintuple $G = (V_1, V_2, V_3; E_1, E_2)$ where V_1 and V_2 and V_3 are three disjoint sets of vertices and $E_1 = \{(i_k, j_k) : i_k \in V_1, j_k \in V_2; k = 1...d_1\}$ and $E_2 = \{(j_n, h_n) : j_n \in V_2, h_n \in V_3; n = 1...d_2\}$ are two sets of edges.*

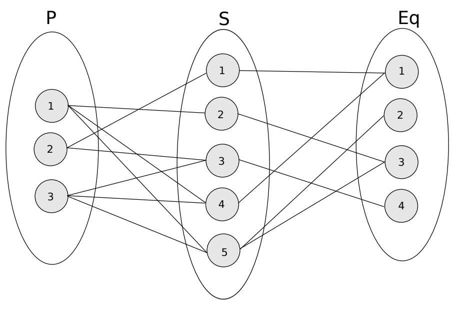


Figure 3.2: A tripartite graph as it occurs in the production system

In Figure 3.2 the situation is shown for the agile production system. On the left side the products to be made are in set P. In the middle, the set S

of steps is displayed and on the right the set Eq of equiplets. Edges show the connection between the products and the steps as well as the steps and the equiplets. The step path for product P_1 is:

$$< \sigma_5, \sigma_2, \sigma_4 >$$

There are four equiplets, where Equiplet 1 offers steps σ_1 and σ_4, Equiplet 2 offers step σ_5, Equiplet 3 offers steps σ_2 and σ_5 and Equiplet 4 offers step σ_3.

3.1.2 Adjacency matrix and travel time matrix

Consider a grid G of N equiplets, together offering M production steps, this grid can be described by a matrix. This matrix is called *adjacency matrix* and is actually a matrix description of a bipartite graph consisting of the set of equiplets, the set of product steps and the connecting edges. In our graphs the edges have no weight and this will result in an adjacency matrix containing only 0 or 1. The adjacency matrix G_{step} shows the mapping of equiplets to production steps.

$$G_{step} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{pmatrix}$$

In this matrix, $a_{ij} = 1$ if equiplet E_i offers step σ_j, otherwise $a_{ij} = 0$. Another matrix G_{time} that will be usefull in planning, shows the time to travel between equiplets in the grid:

$$G_{time} = \begin{pmatrix} 0 & \tau_{12} & \dots & \tau_{1N} \\ \tau_{21} & 0 & \dots & \tau_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \tau_{N1} & \tau_{N2} & \dots & 0 \end{pmatrix}$$

Here τ_{ij} is the time in time steps to travel from equiplet E_i to E_j. Remark that in general $\tau_{ij} \neq \tau_{ji}$.

3.1.3 Optimisation

A sequence of production steps was defined in Chapter 2 as a step path. For instance, consider a product to be built with three production steps, this product has step path:

$$< \sigma_5, \sigma_2, \sigma_4 >$$

Let us assume a simple grid with four equiplets E_1, E_2, E_3 and E_4, each offering a set of steps. The steps offered by an equiplet are denoted between parentheses as in $E_1(\sigma_1, \sigma_4)$. This grid can be described by this set of equiplets:

$$\{E_1(\sigma_1, \sigma_4), E_2(\sigma_5), E_3(\sigma_2, \sigma_5), E_4(\sigma_3)\}$$

This situation can also be described by the adjacency matrix G_{step}.

$$\begin{array}{ccccc} & E_1 & E_2 & E_3 & E_4 \\ \sigma_1 & 1 & 0 & 0 & 0 \\ \sigma_2 & 0 & 0 & 1 & 0 \\ \sigma_3 & 0 & 0 & 0 & 1 \\ \sigma_4 & 1 & 0 & 0 & 0 \\ \sigma_5 & 0 & 1 & 1 & 0 \end{array}$$

A product agent will make a selection of these equiplets based on the production step or steps that must be performed to construct the product. Next, the product agent will ask the equiplet if the steps offered are feasible given the parameters for the steps. The positive response from the equiplet agent contains an estimated time to complete a given step. This information about the duration of a step will be used in the scheduling phase. When a negative response is received by the product agent it will discard the equiplet. Several solutions to map the steps to equiplets may exist. A sufficient solution for the given situation with a minimum of transitions is:

$$< E_3(\sigma_5), E_3(\sigma_2), E_1(\sigma_4) >$$

Theoretically, the number of solutions may be very high if the equiplets each offer a big set of production steps. However in practice an equiplet offers only one or perhaps two different steps. Keeping in mind that only one solution is needed, the maximum number of solutions that are calculated by the product agents is in our situation for practical reasons limited to four. One solution might be not enough, because in case of an infeasible scheduling, other solutions should be at hand. These solutions can also be useful in case one of the already scheduled equiplets breaks down. Mechanisms to optimise the scheduling are as follows.

- Minimising the amount of movements of the product between equiplets.
- Taking care of the load of a certain equiplet, so an alternative equiplet is searched for in case an equiplet has a high load.
- Avoidance of unreliable functioning equiplets.

To find some efficient solutions we try to minimise the transitions between different equiplets, this is done by using a so called production matrix. This production matrix can be derived from the adjacency matrix by selecting the rows of the production steps in the same order as in the tuple that describes the step path $< \sigma_5, \sigma_2, \sigma_4 >$. The production matrix is a reduction of the adjacency matrix.

$$\begin{array}{ccccc} & E_1 & E_2 & E_3 & E_4 \\ \sigma_5 & 0 & 1 & 1 & 0 \\ \sigma_2 & 0 & 0 & 1 & 0 \\ \sigma_4 & 1 & 0 & 0 & 0 \end{array}$$

The production matrix is reduced by eliminating the colums that contain only zeros. In this case the column under E_4. This results in a matrix where for every σ_i in this step path a row of a production matrix is created:

$$\begin{array}{cccc} & E_1 & E_2 & E_3 \\ \sigma_5 & 0 & 1 & 1 \\ \sigma_2 & 0 & 0 & 1 \\ \sigma_4 & 1 & 0 & 0 \end{array}$$

The rows have the same order as the sequence of steps. Matrix element α_{ij} gives the relation between equiplet E_j and production step σ_x at row i. If the step σ_x at row i is supported by equiplet E_j then $\alpha_{ij} = 1$. Not supported steps result in $\alpha_{ij} = 0$.

Optimization should result in a new matrix where α_{ij} has a slightly different meaning and can be different from 1 or 0, giving the product agent a clue for its choice. The product agent will choose the equiplet corresponding with the highest value of α_{ij}. A quite useful optimization is minimizing the transitions for a product from equiplet to equiplet. The optimizing algorithm will search for columns j with sequences of $\alpha_{ij} = 1$ and increment the values in a given sequence by the length of the sequence minus one. This will be done for all columns starting with $\alpha_{ij} = 1$ The matrix of the example has a column under E_3 with a length of 2, with the result that the values of this sequence will be incremented by 1. The matrix transforms to:

$$\begin{array}{cccc} & E_1 & E_2 & E_3 \\ \sigma_5 & 0 & 1 & 2 \\ \sigma_2 & 0 & 0 & 2 \\ \sigma_4 & 1 & 0 & 0 \end{array}$$

Based on this matrix, the product agent will choose equiplet E_3 for steps σ_5 and σ_2. The optimization algorithm works stepwise. First the best starting point is searched for. This will reveal the best equiplet(s) to start with. Let us assume that we have n steps in the step path. This results in a production matrix of n rows. Suppose that the algorithm reveals a set of k steps to be completed by one equiplet as a start. This means that after completing this sequence of k steps, $n - k$ rows, representing $n - k$ steps, should still be done. We reached this point of $n - k$ steps to be done, with the minimum of movements of the product between equiplets. The algorithm is applied to the remaining part (the $n - k$ rows) of the product matrix, without taking into account the previous k rows. We reach a new situation where the number of rows is again reduced. This is repeated (iteration) until the number of remaining rows is 0. Because of the fact that after every iteration we reach a situation with the minimum of movements of the product between equiplets, the final situation, where the number of rows to be done is 0, will also be reached with the minimum of movements.

A second optimization can be achieved by introducing the workload of the equiplets. A high load of an equiplet could decrease the associated values for the columns associated with that equiplet. This will help to balance the load among the available equiplets. In our model, every equiplet has a buffer with a number S_t of time steps. These time steps are used for the production planning by the product agent. This number S_t is constantly updated, because when a time step has passed by, a new empty time step will be added at the end of the buffer. By using a circular buffer, this concept can be easily implemented in software. A useful way to use the load is to multiply the column values by: $(1 - S_r/S_t)$ Where S_r is the total number of reserved time steps for an equiplet and S_t the aforementioned total number of time steps offered by that equiplet. If S_r equals S_t this means that all time steps are reserved and the multiply factor becomes 0, while if all steps are free $S_r = 0$, resulting in a multiply factor of 1.

The next matrix that plays an important role in finding a set of solutions for the product path is the matrix containing transport times. This travel time matrix can be used for the third optimisation. The matrix holds numbers τ_{ij} that represent the time for a product to travel from one equiplet-i to equiplet-j. For obvious reasons $\tau_{ii} = 0$. Added to this matrix are two extra columns having the values τ_{0i} representing the time to travel from the starting point for a product into the grid to equiplet i and τ_{i0} representing the time to travel from equiplet i onto the exit of the production grid. The production step to transport a product from equiplet i to equiplet j is $\sigma_t(i, j)$. The time this step $\sigma_t(i, j)$ will take is τ_{ij}. For the example $< \sigma_5, \sigma_2, \sigma_4 >$, the resulting planning path will be:

$$< \sigma_t(0, 3), \sigma_5(3), \sigma_2(3), \sigma_t(3, 1), \sigma_4(1), \sigma_t(4, 0) >$$

In this formula, apart from the special transport step σ_t the parameter for the production step is the ID-number of the equiplet that will perform this step. By comparing the results for the different planned paths, the fastest path can be selected first.

The result from the planning phase will now be one ore more product paths. The next phase will be the scheduling of the planned product path. When the scheduling is completed, the actual production phase can start. The scheduling uses a planning board where the availability of all equiplets is visible. The process scheduling is an atomic action of the product agent. The product agent will schedule its complete path. During that scheduling phase, the product agent will update the planning board according to the situation that is newly arrived at.

3.1.4 Planning and scheduling complex step paths

In the previous section the focus was on products with a single sequence of production steps. In practice however, most products are manufactured by starting with the production of half products that are combined to make the final product. In Figure 3.3 a simple example of this situation is presented. In a more formal approach it means that the production steps are still a tuple, but the members of the tuples could be sets of tuples. The step path of Figure 3.3 can be presented by:

$$< \{< \sigma_1, \sigma_2 >, < \sigma_3, \sigma_4 >\}, \sigma_4, \sigma_7, \sigma_2, \sigma_1 >$$

To realise such a product the product agent will spawn child agents that will be responsible for the manufacturing of the half products. When these child agents are finished, they will transfer the production information to the parent agent that will finish the production. There is a catch in this situation concerning the scheduling. If the product must be completed before a certain deadline, the parent agent should coordinate the scheduling of the children with its own scheduling. All children will plan and schedule their product paths and report this planning to the parent. The parent will schedule its own remaining part of the product path. This will result in a total schedule. If the total scheduling is feasible, all child agents will get a go for production and the parent product agent will wait for the children to complete their path. Afterwards the parent will complete the product.

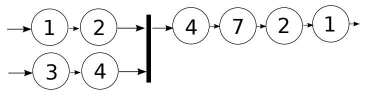


Figure 3.3: Combining two half products

3.2 Scheduling concepts

Scheduling is the process of allocating resources at a certain time to a certain requester. Scheduling is a subject of many publications. Before a possible scheduling solution for the production grid is introduced, an overview of scheduling is presented here. First of all we need to state that a product has a certain release time. This is the time when an order for a product arrives or when the need for making a product is there. In our implementation it is the moment the product agent is born. The product will also have a deadline. The product should be completed before the deadline. If it is not possible to produce the product before the deadline the production is considered not feasible. In this case the product should be delegated to another production grid or an alternative production method or perhaps the deadline should be reconsidered.

In the literature about scheduling there are different concepts. To obtain a global understanding of scheduling we will discuss some concepts here. In our context, a task is considered a production step that should be performed and completed. In a more general treatment of scheduling we consider a task as an entity that claims a resource for a certain time. First we consider some properties of the way scheduling is implemented. The description of these realizations also depend from what perspective the scheduling is seen: the requester of the resource or the resource itself.

- Static versus dynamic scheduling. From the viewpoint of the resource: in case of static scheduling, there is no real scheduling algorithm, but we just use a list of tasks to start and complete at certain fixed times and in the same order as the list requires. Dynamic scheduling on the other hand requires an approach where the list of tasks is dynamically generated by some kind of scheduling algorithm. From the viewpoint of the requester: in case of static scheduling the requester is an entry in the list with a fixed time slot. In case of dynamic scheduling, the requester asks for a resource at a certain moment and will be granted a start time to access the resource by the scheduling algorithm.

- Off-line versus on-line. The scheduling of tasks can be done before the system actually starts to work. In this case we can use a very complex algorithm with a lot of optimization, because we are planning before we actually start. On-line scheduling is more a just in time approach, where we need a fast algorithm to decide what will be the next task to work on.

- Centralized versus distributed. In a centralized scheduling system we have a central scheduling system that makes the planning for all the requesters requiring a resource. In a distributed system the scheduling is based on negotiating between the stakeholders (requesters or resources or both). This approach fits better in an agent-based environment.

3.3 Production scheduling

Production scheduling is the scheduling of manufacturing or assembling products. There are many things that play a role in this type of scheduling. We mention seven of them.

1. Optimal use of available resources. By this production workers as well as production machinery are meant.

2. Shifts. The planning of human resources according to their constraints like maximum hours per day, capabilities etc.

3. Batches. Most production scheduling is based on batch production. In this type of production bulk manufacturing is accomplished. Large batches of the same product are planned and produced.

4. Maintenance. Normally maintenance should be taken into account when scheduling the resources. If this aspect is neglected a long interruption of the production process can occur.

5. Repair. Even though maintenance should prevent repair of equipment during production, this aspect should be taken into account. A short repair time can be considered if a system fails during production and a thorough inspection and repair should be planned right after the production batch. This aspect is also called disaster recovery and has to do with main time to repair (MTTR) (Gnedenko et al., 1999).

6. Commodities and production material. Commodities to make the product and production material that is needed and used for production.

7. Storage of half products. Normally storage of half products is considered wasted money, because if it can be avoided the result of the production is still the same, without this extra overhead.

Most production scheduling systems focus on the sequencing or scheduling of parts processed in high-volume. Such production systems are also called repetitive manufacturing systems. In such settings, one can look to just-in-time (JIT) (Schonberger, 1982) and lean manufacturing principles for how to control production. These approaches generally do not need the same type of production schedules discussed here. In the grid manufacturing the focus is on agile production of small volumes. Section 3.3.1 gives an historic overview of production scheduling followed by Section 3.3.2 that describes the modern approach of production scheduling.

3.3.1 Historic overview

For analyzing existing production scheduling systems a background in the development of production scheduling is useful. Therefore a short overview of the history of scheduling is presented here. An elaborated overview is given by (Herrmann, 2006). Wight (Wight, 1984) defines scheduling as "establishing the timing for performing a task". Priorities and capacity play therefore an important role. One could say: *what should be done first* and *who should do it.* Cox (Cox et al., 1992) define detailed scheduling as "the actual assignment of starting and/or completion dates to operations or groups of operations to show when these must be done if the manufacturing order is to be completed on time." The next observation is that there are different levels of scheduling in a manufacturing environment.

1. Master scheduling or order scheduling.
2. Operations scheduling.
3. Shop scheduling.

The order or master schedule is the highest level of scheduling in production and plans the production of an order that could have one or more batches. Operations scheduling are short-time plans to realize the order- or master schedule. These plans state which jobs should be done, possibly in parallel or at certain times. Finally shop scheduling is the most detailed scheduling, but only concerns one production unit. So its scope is limited. However, this scheduling led to the notorious job-shop scheduling problem. The job-shop scheduling problem is about scheduling a set of jobs to a set of machines, where every machine can only handle one job at a time. Every job has a specific order through the machines. The objective is to schedule the job so as to minimize the sum of their completion times (Applegate and Cook, 1991).

The first factories as we would call them today appeared during the middle of the eighteenth century. These factories were quite simple and small and designed to produce a small number of products in large batches. Foremen coordinated the activities needed for the limited number of products for which they were responsible. During the nineteenth century factories grew bigger but not more complex. The scheduling problem was only concerned with the fact when an order should begin or when the order is due. There was no formal method needed for this type of manufacturing. Around 1890 this situation changed. Factories made a wider range of products (Herrmann, 2006) and this led to complexity. Foremen who were responsible for their own so called production cells were not in the position to handle this complexity and a separation of planning from execution emerged as introduced by Frederick Taylor. Planners in a production control office took over scheduling and formal scheduling methods were introduced. Production planners created a master production schedule, taking into account the production capacity and the orders from customers. So called shop orders were derived by work order planners to provide information about production to the foremen running their production cell. Fine tuning and disaster recovery could be done at in the production cell itself. Here we see the aforementioned three levels of scheduling appear.

Henry L. Gantt introduced a formal approach to scheduling by recognizing the need for such a system to coordinate activities to avoid what he called "interferences" (Gantt, 1903). He introduced charts for production control. A definition of a Gantt Chart given by Cox (Cox et al., 1992) is: "A Gantt chart is a control chart designed to show graphically the relationship between planned performance and actual performance." Gantt introduced different types of charts (Gantt, 1916). We will introduce some of them that are still useful in a modern production environment.

- Daily balance of work shows the amount of work to be done and the amount that is done.
- The man's record shows what each worker should do and did do.
- Machine record shows the amount of work done and to be done by a production machine.
- The layout chart specifies when jobs are to be begun, by whom, and how long they will take.

During time more variants of Gantt charts were introduced (Mitchell, 1939). Today these variants are also used in project management software. The charts give a graphical way to visualize schedules and states of the production facility in all its aspects. Tools like planning boards (also called control board, dispatching board or schedule board) were used to design the scheduling of jobs, human resources and production hardware.

The next step in scheduling was computer-based scheduling. Large project scheduling was the first type of scheduling to use computer algorithms (1956), Computer-based production scheduling appeared ten years later. Around 1965 computer-generated dispatch lists from input factors like processing time, due time, number of operations were used for production scheduling. Every production cells received its list from the scheduling system telling workers what to do at what time. This first generation computer-supported scheduling involved a lot of paperwork. Interactive computer-based scheduling appeared later. These systems used graphical displays to interact with the planner and were based on data in a database system, a schedule generation routine, a schedule editor and a scheduling evaluation routine. By that time computers also appeared on the production floor. First as standalone machines, but later these systems were connected by a network. The paperflow from planning to production changed to dataflow.

These approaches are nowadays used in modern manufacturing planning and control systems like ERP or MES. In more complex systems next to production planning, also production planning-related issues like material requirements, storage and logistics were introduced.

3.3.2 Modern approach

In modern production systems, the scheduling task is a subsystem of larger software systems, like MES (manufacturing execution system) or ERP (enterprise resource planning). These systems are mostly based on a large central database system where all kinds of production information and available resources are real time kept up to date. These systems offer possibilities for human interaction and fine tuning. In complex situations it is possible to schedule production for all the three aforementioned levels, like master scheduling, operational scheduling and shop scheduling. By using computer systems on the production floor, it is also possible to have real time production information feedback available at all layers in the automation pyramid. Although the architecture is layered, we still do not have a real distributed scheduling system that is needed for agent-based production in a grid. In the next section we will discus special properties of agent-based scheduling.

3.4 Scheduling with agents

Research on scheduling in agent-based systems focusses on distributed scheduling where agents try to schedule a process by exchanging information needed to come to a feasible schedule. In such a situation a centralised scheduling system or scheduling agent might be considered. The big advantage of such an approach is that this scheduling system can collect all relevant data and generate a schedule for all agents involved.

In a distributed scheduling approach the performance or quality of the schedules generated can be compared with the central scheduling approach and this gives a clue how good the distributed scheduling actually is (Heydenreich et al., 2010). In the case of scheduling in an agent-based system the following considerations should be taken into account:

- Selfish agents versus cooperative agents.
- Trusted or foreign agents.
- Open agents or agents keeping secrets.

A big advantage of distributed agent-based scheduling can occur in situations where a problem pops up. A central scheduling system might become overloaded with rescheduling requests while in a distributed system the problem can be kept to a subset of agents involved in the problem. This is what is called islanding and is often the real reason to use a distributed agent-based scheduling system.

3.5 Multiagent production system

The multiagent-based production system has already been described in the previous chapters. The scheduling will be done by the product agents. Product agents will arrive at random times and each agent will have its own particular set op production steps to be performed. In the real implementation, we expect that every equiplet offers only one or a small set of steps. In this chapter the assumption of one unique step per equiplet is made. There are important differences with the aforementioned job-shop scheduling problem. Our aim is to make sure that a product is completed before its deadline. Though not yet used here, equiplets can offer more than one step. Equiplets can be redundant and equiplets can be reconfigurable. Requests for product steps arrive at random.

3.5.1 Multiagent-based scheduling in the grid

In a multiagent-based scheduling system a number of agents planning a schedule for a certain part of the system as a whole will mostly share parts of the environment. This results in an often distributed scheduling system where agents work in parallel on parts of the scheduling. An important aspect is the awareness of these agents of other agents and their willingness to cooperate. A selfish agent will try to schedule its subsystem without taking in account the other agents. A more social agent is willing to negotiate with other agents even if it has already claimed its resources. The system that we need for the grid scheduling and that will be studied here has the following characteristics:

- Resources are available to all agents.
- All actions in the grid take an integer value of a basic time unit.
- Agents should be willing to cooperate to help each other in case of non-feasibility.
- An already scheduled subsystem will only be given up by an agent if there exist a feasible alternative for that agent.
- there is a reliable data communication infrastructure between the agents.
- Agents do not have secrets or private information that is unavailable to other agents.

Normally in production-based scheduling, there is a central scheduling system. This centralized solution has the drawback of introducing a single point of failure, bad scalability characteristics and the fact that the responsibility for making the product is partly taken away from the product agent. The system that will be described here makes it possible for every product to use its own scheduling approach so it is not tied to the possibilities or limitations of a centralized scheduling system. This also comes handy when a product is made of subparts. As explained earlier there is a parent-child relation between the production threads and when something goes wrong in one of the already scheduled child agents, the problem to find a solution can be restricted to the subset of agents involved.

The globally shared information is the aforementioned blackboard, containing the available steps and the timeslots for every step. Every timeslot shows a number that is the unique number of the product agent that has claimed this slot. If the number in the timeslot is zero, it is still available.

3.5.2 Objectives of the scheduling

The scheduling has five objectives.

1. It should offer a best effort to schedule products that will arrive at random times.

2. It should schedule products at high grid loads.

3. It should be fast and reliable. The scheduling should take a small amount of time.

4. It should introduce only a small intercommunication overhead. This will mean that the amount of interagent messages should be kept low.

5. It should be fair. When a product is scheduled for production with a feasible scheduling, meaning the product will be completed before the deadline, the scheduling will guarantee that this scheduling will not be changed to an infeasible scheduling by the scheduling system.

3.5.3 Picking a timeslot

In the grid, all equiplets offer production steps. For every grid a sequence of timeslots will show the availability of the equiplet. A timeslot can be free or claimed by a product agent. The first decision that must be made is selecting which timeslot of all free slots will be used. It might be tempting to use the first available slot, but we should investigate if that is really a good choice, keeping in mind that perhaps a just-in-time (JIT) scheduling might also be a good choice, because it will keep the amount of products in the grid low. The following four free slot selection methods were tested in our simulation:

1. First fit. Take the first slot in time available.

2. Just in time. Schedule the last step as near to (but before) the deadline and from there move backwards in time for all steps to be scheduled. Placing the steps as close to each other as possible.

3. Equally distributed. The steps are scheduled between release time and deadline leaving if possible an equal distance if possible between the steps.

4. First fit plus. This scheduling is the same as first fit, but an extra timeslot has been inserted between every two steps to recover from unexpected delays.

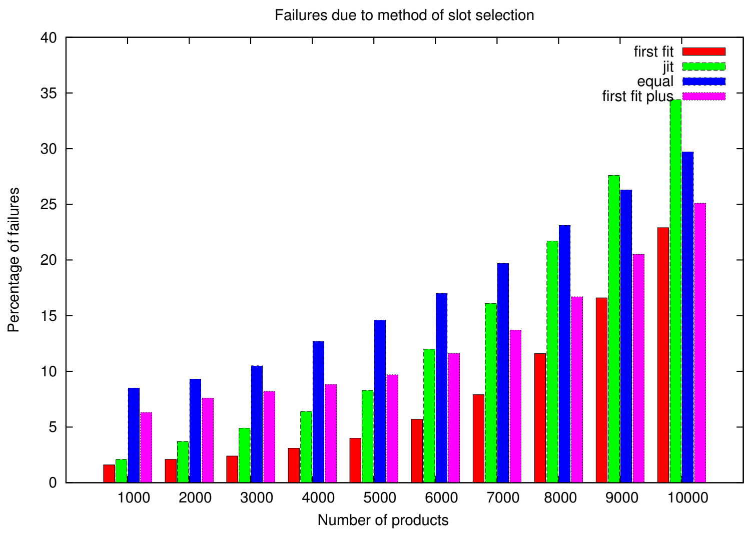


Figure 3.4: The influence of making a choice for a free slot

These four approaches have been used in scheduling ten sets of products, starting with a set of 1000 products and increasing the number of products for every next set by 1000 upto a set of 10000. No rescheduling has been applied, so only the failures in getting a feasible scheduling at once are counted. All tests use 10 equiplets, each offering a number of 10000 timeslots. Products will use different number of timeslots, but on the average it will be 10 timeslots per product. Every product has a deadline and on the average, the number of timeslots between release time and deadline will be 10 times the number of timeslots needed by the product. Figure 3.4 shows the results. As can be seen from the figure, the first fit approach is giving the lowest number of failures and thus performs the best. JIT performs only good at a very small load. This can be understood by the fact that a product will immediately start to be produced and taking timeslots near the deadline will skip earlier timeslots available that might disappear when time goes by. The other two methods perform worse than first fit for comparable reasons. First fit plus takes care of unexpected delay recovery but it is better to recover from these kind of problems in a different way because recovery should not be taken care of at this stage of the production, but only in situations where it is needed. Picking a timeslot has only to do with the method that will be used of selecting a timeslot from a set of timeslots. This might result in a feasible schedule, but as shown in Figure 3.4, even for small numbers of products, there are failures. To reduce the failures, a scheduling method should be provided. In the next section, the scheduling will be examined.

3.5.4 Real time scheduling

When multiple product agents are to be scheduled, this can be considered a multiresource real time scheduling problem. In contrast with a scheduler in a multiprocessor environment where all the resources (i.e. processors) offer the identical service of code execution, this scheduling must handle resources that offer different types of service. Every product agent has its release time and its deadline and claims resources in between. There are several solutions proposed for real time scheduling. The type of solution depends on the given situation. Some situations can be solved before starting the system and a fixed (static) scheduling scheme can be used. In our case however we need a dynamic type of scheduling. The tasks to be scheduled are unpredictable. This type of task is called a spurious task in real time scheduling jargon. This type of scheduling can be handled by different scheduling schemes. To explain some schemes, some symbols used in expression should be defined:

P is the product set. A single product is denoted as P_i.
r_i is the first timeslot after release of product P_i
d_i is the timeslot for the deadline of product P_i
τ is the current timeslot
$s_i(\tau)$ is the number of timeslots of product P_i that is left to be done.

Five well known scheduling schemes are as follows.

1. Fixed priority, FP. Every task is assigned a priority depending on the task type. The highest priority tasks are completed before the lower priorities are run.

2. Earliest deadline first, EDF. The task with the first deadline to come gets the highest priority and is handled first.

3. Least slack first, LSF. The task with the minimum slack gets the highest priority and is handled first. Slack is defined as the total time available until the deadline minus the time to complete the task. The slack for product P_i at timeslot τ can be written as $(d_i - \tau) - s_i(\tau)$.

4. Smallest critical ratio first, CR. The critical ratio is defined as the total number of timeslots available divided by the number needed. For a product P_i at timeslot τ: $(d_i - \tau)/s_i(\tau)$. If this number turns out to be 1, all timeslots should be used. If it is lower than 1, the scheduling is infeasible. A high number shows that many slots are available for a relative small number of needed timeslots.

5. Shortest process first, SPF. The task with the shortest time to complete get the highest priority.

All these types can be used in conjunction with what is called preemption. By this is meant that when a higher priority task arrives, another already running task is paused (preempted) to make way for the higher priority task. After completion of the higher priority task, the preempted task is resumed. Because all agents are equal, fixed priority is inadequate as a scheduling scheme for the production grid. Both EDF and LSF are considered optimal in the sense of feasibility: if there exists a feasible schedule, EDF or LSF will find it (Cottet et al., 2002). However this is only true for the situation where a single resource is scheduled among requesters, as is the case in a single processor computer system, where the processor time is scheduled among different tasks. In the next section a model will be described where multiple resources are available as is the case in the production grid.

3.6 Scheduling simulation

To investigate the behaviour of the scheduling system for the grid, a simulation was built. First the model will be described. It is assumed that every product agent has a single tuple consisting of the number of production steps between one and MAXSTEP. Every step takes one or an integer multiple of a basic time unit. We only consider equiplets with one unique production step. Travelling from equiplet to equiplet can also be seen as a production step claiming one of the transport resources.

3.6.1 Generating test sets

Several test sets were used in the simulations. To generate a test set a tool has been developed. Parameters can be set by the user: the number of products, the maximum number of production steps for a product, the number of time steps, the maximum production time for a product and the average shape of the distribution of the production request in time. The minimum production time Pt_{min} is calculated by summing the time of the individual steps. The maximum production time Pt_{max} is:

$$Pt_{max} = Pt_{min} + R \times Pt_{min} \tag{3.1}$$

Where R is a range value that is a parameter for a test set. R introduces a time buffer that can be used for rescheduling possibilities. The minimum time is the total time of the number of production steps. The production time for products in the test sets is for a certain product a random value between Pt_{min} and Pt_{max}. The same so-called Monte Carlo technique is used to generate a random number of production steps, a random release time and

Table 3.1: Example of simulation parameters

Parameter	Value
Number of product agents	10000
Range	0-20
Number of steps per product	1-20
Number of different steps in the grid	10
Average distribution shape	F, I, D

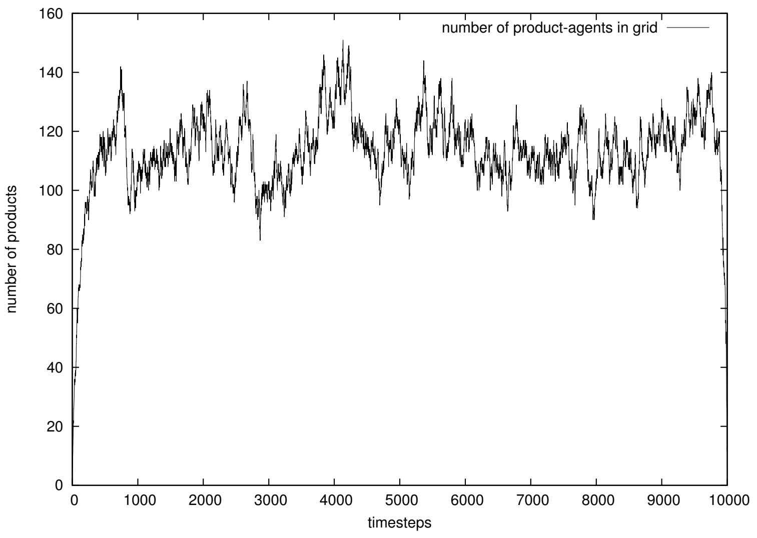


Figure 3.5: Test set with flat distribution of the number of product agents

a random deadline within the limits of minimum and maximum production time for every product. Table 3.1 shows the parameters used for the test sets. We generated an even distribution, a linear increasing distribution and a linear decreasing distribution for our scheduling scenario tests.

In figures 3.5 and 3.6 these distributions are plotted. The number of product agents in these plotted distributions was chosen such that not all products could be produced. The test sets of Figure 3.6 and Figure 3.7 were introduced to study the behaviour of the scheduling system under an increasing or decreasing load. In all three types of test sets, the total number of required steps is more than the steps available. This will automatically result in an infeasible test set. The sets to be scheduled turned out to have an infeasible scheduling indeed.

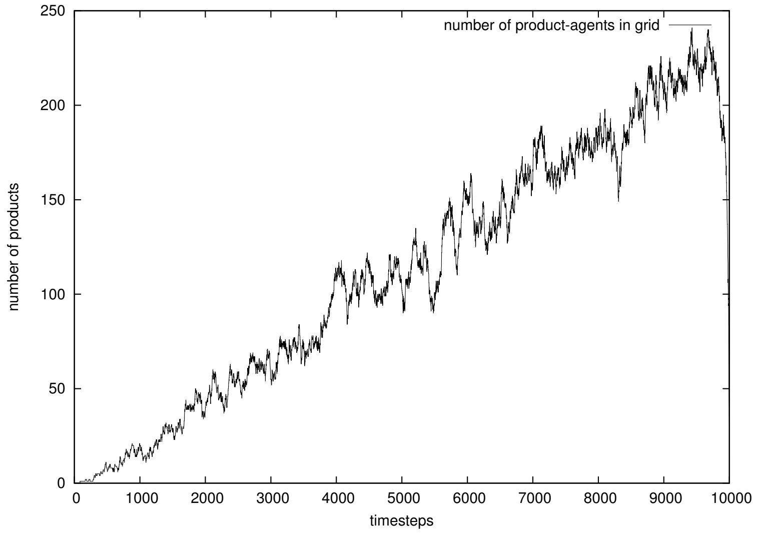


Figure 3.6: Test set with increasing distribution of the number of product agents

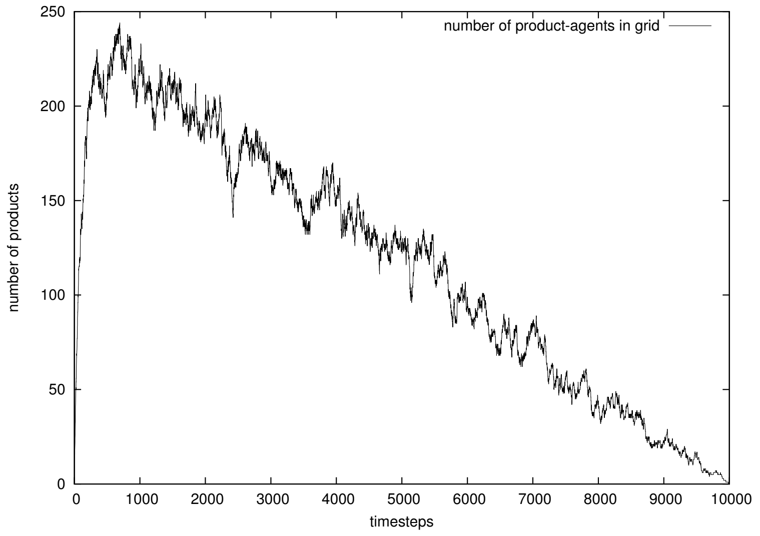


Figure 3.7: Test set with decreasing distribution of the number of product agents

3.6.2 Preparing the test sets

Under normal circumstances the system should receive a production request by the release time of every product. To investigate the feasibility of the test sets, alternative sets were generated by sorting the sets according to deadlines. This results in test sets that use earliest deadline first. The scheduling results of the EDF-sets were used to generate a feasible set by excluding the failing production schedules. However this feasibility only holds for the set sorted according to deadlines. To generate the actual feasible test sets used for the simulation, the feasible EDF-sets were sorted to release time thus mimicking the real situation where product requests arrive according to release time.

3.6.3 Negotiating

By negotiating the interaction between two or more product agents is meant to come to a revised scheduling or the conclusion that scheduling is not feasible. The following two assumptions about the agents are made.

- The agents will do their best to cooperate as well as possible. So hostile behaviour should not be assumed.
- An agent will not give up its scheduling if there is no feasible alternative. So after negotiating, an agent that is already actively producing a product will surely complete the product before the deadline.

These assumptions will result in a rather simple negotiating scheme. In the following text, we use EDF as an example, but EDF can be replaced, *mutatis mutandis*, by LSF, CR or SPF. The test sets were used in three conditions for inter agent negotiating:

1. No negotiating. Product agents try to schedule just by looking at the information available on the blackboard.
2. Negotiating in case of failure. If the scheduling based on the information on the blackboard fails, the product agent will start to negotiate with other active agents in the production system. The negotiating is very simple. In case of EDF it will ask all active production agents with a later deadline to temporally give back their claimed production steps starting from the release time of the failed product agent. This failed product agent will now try to schedule its production. In case of a new failure it will report failure and the other active product agents will reclaim their steps. In case of success, the agent that has now a feasible

schedule, will try to reschedule, in order of deadlines, all involved active agents that had temporally gave up their schedules. Only if all of the reschedules for these agents are successful, the new scheduling for all participating agents as well as the newly arrived agent is accepted. In this thesis this type of scheduling is called *weak EDF*.

3. Negotiating in case of an earlier deadline. In this situation, there is always negotiating between the agents and all active agents with a later deadline. So every newly arriving agent will start to negotiate with all active agents with a later deadline. The disadvantage of this approach is that the overhead of negotiating is probably much higher than in the previous situation. In this thesis this type of scheduling is called *strong EDF*.

The implementation of the latter two approaches is done by sending broadcast messages as well as agent to agent messages. In case of *negotiating in case of failure*, an agent with a failed scheduling will broadcast its deadline to all active product agents. In reply to this broadcast, agents with a later deadline will send their claimed production steps to the new agent and this will try to schedule its production assuming these claimed steps are now available. If the scheduling succeeds it will try to reschedule all other active agents. In case of success it will adjust the blackboard information and send the new schedules to the participating agents. By locking the access to the blackboard, this scheduling action is atomic.

3.7 Simulation software

Apart from some minor tools, the simulation software has three separate main tools. The first two are command-line tools, running on Linux or Windows. The third one is a graphical tool that uses the Qt-library to make it a multi platform tool.

1. A scenario generator tool. This tools is based on Monte Carlo techniques to generate production scenarios. It generates a list of an amount of products to be made and for every product, the release time, the deadline and the list op production steps needed for that specific product. The type of scenario, the amount of products and other parameters can be specified. The output is an XML file containing the information for every single product.

2. A scheduling simulator. This tools simulates the multiagent scheduling system. As input is given the output from the scenario generator. The

Table 3.2: XML-tags for the scenario files

XML-tag	meaning
$< P >$	product
$< TR >$	release time
$< TD >$	deadline
$< NPS >$	number of product steps
$< STEPS >$	product steps

type of scheduling and other parameters as well can be specified by the user. The output will show all kinds of information about the actual scheduling. An XML file can be generated to actually see what happens every time step.

3. An analysing tool. The analysing tool is a graphical tool that displays the actual scheduling over the available equiplets in a gant-type diagram. It will show which products have been rescheduled and also the failures.

3.7.1 XML output file of the scenario generator

A typical part of the output of the scenario generator looks like:

```
<P>P7677<TR>3</TR><TD>15</TD><NPS>2</NPS>
<STEPS>2,7,</STEPS></P>
<P>P5885<TR>5</TR><TD>263</TD><NPS>18</NPS>
<STEPS>6,5,9,2,6,9,8,9,8,7,0,6,4,2,4,6,4,6,</STEPS></P>
<P>P138<TR>6</TR><TD>119</TD><NPS>10</NPS>
<STEPS>7,2,8,5,0,4,0,7,6,8,</STEPS></P>
```

Table 3.2 shows the meaning of the XML-tags used. This shows the data for three products. Product P7677 having release time 3, deadline 15, two production steps, tuple $< 2, 7 >$ followed by two other products. The output is in this case sorted to release time, but other possibilities exist.

3.7.2 XML output file of the scheduling simulator

Another type of XML file can be optionally generated by the scheduling simulator. When the scheduling simulation is generating its output to an XML file it starts with information about the scenario properties. This looks like:

Table 3.3: XML-tags for simulator output file

XML-tag	meaning
$< CONFIG >$	simulation configuration
$< TSS >$	number of time steps
$< EQS >$	number of equiplets
$< PRS >$	number of products
$< SCHED >$	scheduling information
$< TS >$	start time tag
$< P >$	product
$< TR >$	release time
$< TD >$	deadline
$< NPS >$	number of product steps
$< EQ >$	equiplets
$< TIMESTEPS >$	timeslots used
$< F >$	failing scheduling attempt
$< FAILED/ >$	failed scheduling

```
<CONFIG>
<TSS>10000</TSS>
<EQS>20</EQS>
<PRS>10000</PRS>
</CONFIG>
```

In this case we consider 10000 time steps, 20 equiplets (each having its own unique production step) and 10000 products. The next part of XML output shows the successful scheduling $S4698$ of product $P4698$. This product has only two production steps and these are scheduled at timeslots 167 and 168.

```
<SCHED>S4698
    <TS>T167</TS>
    <P>P4698
       <TR> 167</TR>
       <TD>189</TD>
       <NPS>2</NPS>
       <EQ>4,8 </EQ>
       <TIMESTEPS>167,168 </TIMESTEPS>
    </P>
</SCHED>
```

The scheduling $S9134$ of product $P9134$ will at first fail for its last production step due to its very narrow scheduling window. This failure is visible by the value -1 in the set of time steps. So this product should be rescheduled. This rescheduling involves products $P5649$ and $P9597$ having later deadlines. First a successful scheduling of $P1934$ is realised and for the other two involved products a feasible schedule is also possible as shown in the following XML-sniplet.

```
<SCHED>S9134
 <TS>T545</TS>
<F>
<P>P9134<TR> 545</TR><TD>554</TD><NPS>9</NPS>
<EQ>3,1,6,5,0,1,6,1,4</EQ>
<TIMESTEPS>545,546,547,548,550,551,552,553,-1 </TIMESTEPS>
</P>
</F>
<P>P9134<TR> 545</TR><TD>554</TD><NPS>9</NPS>
<EQ>3,1,6,5,0,1,6,1,4</EQ>
<TIMESTEPS>545,546,547,548,549,550,551,552,553 </TIMESTEPS>
</P>
<P>P5649<TR> 539</TR><TD>599</TD><NPS>6</NPS>
<EQ>6,3,4,5,9,8</EQ>
<TIMESTEPS>539,541,542,543,544,545 </TIMESTEPS>
</P>
<P>P9597<TR> 533</TR><TD>793</TD><NPS>17</NPS>
<EQ>2,4,9,0,7,5,7,3,0,8,4,2,9,4,2,6,0</EQ>
<TIMESTEPS>533,534,535,536,537,538,539,540,541,542,
543,544,545,546,547,548,550 </TIMESTEPS>
</P>
</SCHED>
```

In the final example of the XML scheduling output, a failed scheduling is presented. At the first try, all steps will fail. This failure triggers a rescheduling where many other products are involved (not all shown here). For one of the involved products ($P3973$) there is no feasible schedule possible after revision, so the scheduling of product $P4005$ fails and the other products continue with the scheduling as already planned.

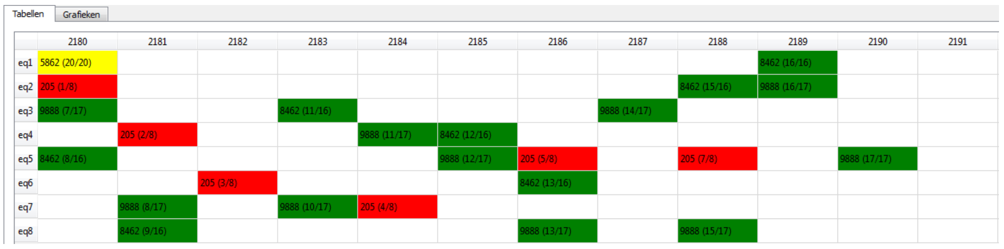
```
<SCHED>S4005 <TS>T9695</TS>
<F>
<P>P4005<TR>9695</TR><TD>9754</TD><NPS>17</NPS>
<EQ>0,4,9,5,9,5,1,3,5,6,5,8,9,7,4,9,7</EQ>
<TIMESTEPS>-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,-1,-1,-1,-1,-1,-1 </TIMESTEPS>
```

```
</P>
</F>
<P>P4005<TR>9695</TR><TD>9754</TD><NPS>17</NPS>
<EQ>0,4,9,5,9,5,1,3,5,6,5,8,9,7,4,9,7 </EQ>
<TIMESTEPS>9719,9727,9733,9734,9735,9737,9738,9739,
9740,9741,9742,9743,9744,9745,9746,9747,9748 </TIMESTEPS>
</P>
<P>P4198<TR>9405</TR><TD>9756</TD><NPS>20</NPS>
<EQ>9,2,4,2,8,0,5,9,1,4,5,7,6,9,3,8,2,1,0,5</EQ>
<TIMESTEPS>9545,9708,9710,9713,9714,9723,
9725,9737,9739,9742,9743,9744,9745,9746,9747,
9748,9749,9750,9751,9752 </TIMESTEPS>
</P>

/* many reschedules in between */

<P>P2076<TR>9411</TR><TD>9765</TD><NPS>18</NPS>
<EQ>7,8,3,1,5,4,5,4,7,1,5,9,5,1,0,6,3,7</EQ>
<TIMESTEPS>9552,9617,9646,9724,9727,9750,9751,
9752,9753,9754,9756,9758,9759,9760,9761,9762,
9763,9764 </TIMESTEPS></P>
<F>
<P>P3973<TR>9497</TR><TD>9766</TD><NPS>20</NPS>
<EQ>4,1,3,4,2,3,6,9,8,6,3,8,0,5,7,4,6,7,9,7</EQ>
<TIMESTEPS>9713,9725,9740,9751,9752,9753,9754,
9755,9756,9757,9758,9759,9760,9761,
9762,9763,9764,9765,-1,-1 </TIMESTEPS>
</P>
</F>
<FAILED/></SCHED>
```

By having all this data available, it is possible to replay the scheduling using a visualisator. This is the aforementioned analysing tool. A screenshot of a part of this tool is given in Figure 3.8. By using this graphical user interface it is possible to go through the scheduling system step by step and see the results of the rescheduling.

3.8 Results

In this section the results of the simulation are discussed. First weak versus strong EDF is investigated for infeasible and feasible test sets. In the second



	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
eq1	5862 (20/20)									8462 (16/16)		
eq2	205 (1/8)								8462 (15/16)	9888 (16/17)		
eq3	9888 (7/17)			8462 (11/16)				9888 (14/17)				
eq4		205 (2/8)			9888 (11/17)	8462 (12/16)						
eq5	8462 (8/16)					9888 (12/17)	205 (5/8)		205 (7/8)		9888 (17/17)	
eq6			205 (3/8)				8462 (13/16)					
eq7		9888 (8/17)		9888 (10/17)	205 (4/8)							
eq8		8462 (9/16)					9888 (13/17)		9888 (15/17)			

Figure 3.8: Visualisator screenshot

Table 3.4: Number of failures in scheduling infeasible sets, N = 10000

Negotiating type	flat	increasing	declining
None	2240	4040	4193
Weak EDF	1042	3558	3547
Strong EDF	1033	3554	3578
Sorted to deadline	569	1904	1800

part, the results for different scheduling schemes as well as the influence of parameter R on the scheduling success rate are presented and discussed.

3.8.1 Weak versus strong EDF

The number of product agents with infeasible scheduling are given in Table 3.4. This set uses the infeasible sets of product agents over time as shown in Figure 3.5 and 3.6. There is no significant difference between strong and weak EDF. In the table is also included the failures for a set that is sorted to deadlines. This is called 'Sorted to deadline'. What one should keep in mind that the situation for the production system will always be similar to the simulation set with a set sorted to release time. This is the real life situation where product agents pop up at random moments. By sorting the test set to deadlines one could see what could be achieved if we had the possibility of scheduling according to deadline.

The results of both weak and strong EDF are better than no negotiating but still much higher than the situation where we have EDF scheduling in the test set that is sorted to deadline. Remarkable is the fact that weak EDF scheduling is only slightly different from the strong multiagent EDF implementation. Table 3.5 shows the results for feasible test sets.

Here again can be seen that weak EDF scheduling is only slightly different from the strong multiagent EDF implementation. In case of the flat test set the difference is 1 in favour of strong EDF. For the increasing set the

Table 3.5: Number of failures in scheduling feasible sets

Negotiating type	flat N = 9431	increasing N = 8096	decreasing N = 8200
None	1499	1115	1300
Weak EDF	24	32	33
Strong EDF	23	30	36
Sorted to deadline	0	0	0

Table 3.6: Number of failures in scheduling feasible sets

Number Products	Pre-empt. Strong	Agents Involved	Pre-empt. Weak	Agents Involved	Succes Weak	Fail Weak
100	27	31	0	0	100	0
200	106	172	0	0	200	0
500	377	1123	0	0	500	0
1000	856	4899	4	4	1000	0
2000	1823	20012	15	34	1999	1
3000	2812	45213	37	138	2998	2
4000	3779	80559	83	468	3996	4
5000	4767	126867	171	1414	4995	5
6000	5758	181986	327	3785	5992	8
7000	6749	248227	647	11522	6987	13
8000	7736	323590	1519	46690	7969	31
9000	8730	409487	4235	237205	8819	181
10000	9719	504481	6336	338108	8958	1042

difference is 2 also in favour of strong EDF. However, for the decreasing test set the difference is 3 in favour of weak EDF. So on the average the difference is even zero. This raised the question what is the overhead of strong EDF versus weak EDF. To measure the overhead, ten different test sets have been used ranging from 1000 to 10000 products in 10000 time steps. The focus was on counting how many times a reschedule would occur and the total amount of agents that are involved in rescheduling. Table 3.6 shows the numerical results. A graphical representations is shown in Figure 3.9 and 3.10. Weak EDF uses much less preemptions and has a considerable smaller amount of agents involved in rescheduling. Only at large values for the number of products a sharp increase will occur. This is the situation where the grid of equiplets becomes overloaded and the number of failed schedules

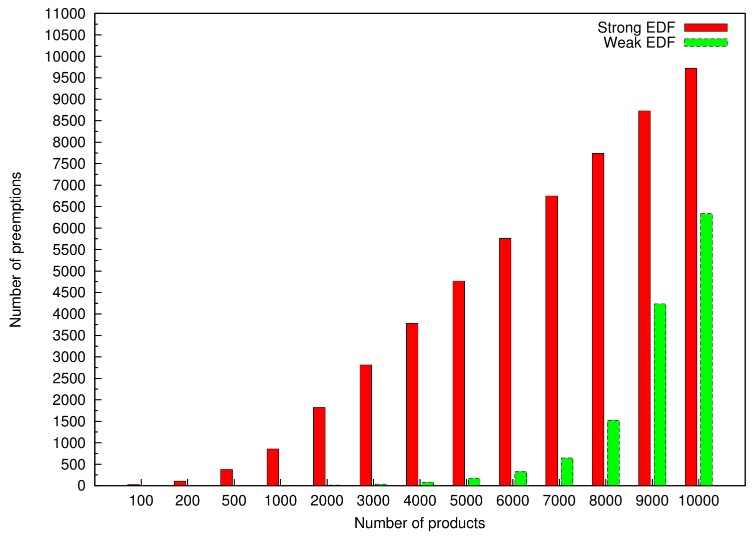


Figure 3.9: Number of preemptions for strong and weak EDF

also increases very rapidly. The strong version shows a steady increase for both rescheduling as the number of agents involved from the start. This can be understood because of the fact that every time an agent enters the grid, all agents with a later deadline are required to temporarily give up their schedule, even if it is not necessary in the given situation. When many agents are involved, this will mean that the scheduling time requires much overhead that will mostly be used for communication between the agents.

3.8.2 Different scheduling algorithms

In this section only the weak implementations of the scheduling algorithms are used. Now the focus is on different scheduling algorithms. Table 3.7 shows the number of failing schedules of the infeasible test sets. It shows that for a flat test set both EDF and LSF perform well reducing the amount of failures when there is no negotiating. SPF is not useful as it is only slightly better than no negotiation. For increasing and decreasing test sets the gain for negotiating is much less due to the fact that both distributions have a high peak (either at the beginning or at the end), resulting in much more scheduling failures and also less possibilities during this peak load to find a feasible schedule. EDF performs a bit better than LSF but not significantly better. CR is worse and SPF is in all situations comparable to no negotiating. For feasible test sets the results are shown in Table 3.8. In this case LSF is the winner, being slightly better than EDF, while SPF is now better than no negotiating but still far behind EDF and LSF. Negotiating does not necessar-

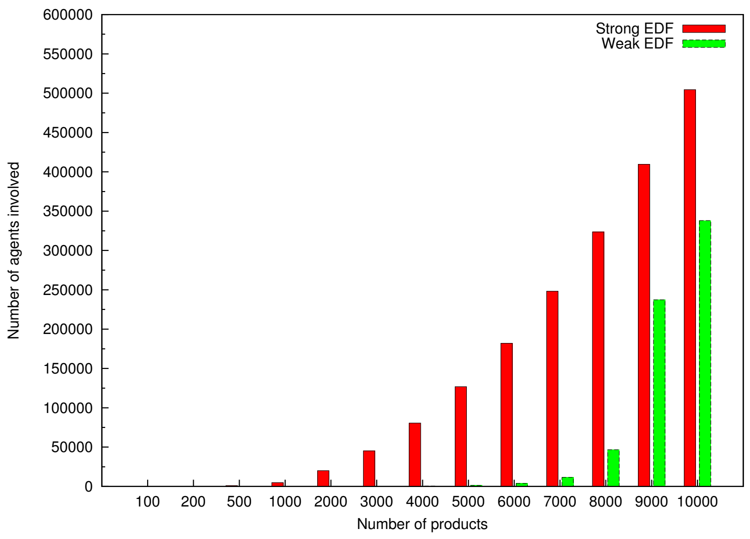


Figure 3.10: Agents involved for strong and weak EDF

Table 3.7: Number of failures in scheduling infeasible sets, N = 10000

Scheduling type	flat	increasing	decreasing
None	2240	4040	4193
EDF	1042	3558	3547
LSF	1082	3606	3604
CR	1450	3679	3838
SPF	1945	3975	4066

Table 3.8: Number of failures in scheduling feasible sets

Scheduling type	flat	increasing	decreasing
	N = 9431	N = 8096	N = 8200
None	1499	1300	1115
EDF	24	32	33
LSF	4	16	23
CR	476	300	439
SPF	1087	849	958

ily result in a feasible scheduling, but the number of failures is less than 0,4% in the worst situation when we consider EDF and LSF. Figure 3.11 shows the number of failed schedules for test sets of ranging from 1000 to 10000 products with range value of 20 using 5 schemes: no negotiating, weak-SPF, weak-CR, weak-LSF and weak-EDF, as a function of the number of products. What can be seen is that until 8000 products, EDF and LSF are capable of scheduling almost all products and at 9000 and 10000 these two schemes give almost the same result. For a low value for the number of products CR is not bad, but it becomes worse compared with EDF and LSF when more products are involved. SPF is worse and no negotiating gives an impression of what is actually achieved by the multiagent cooperation scheme.

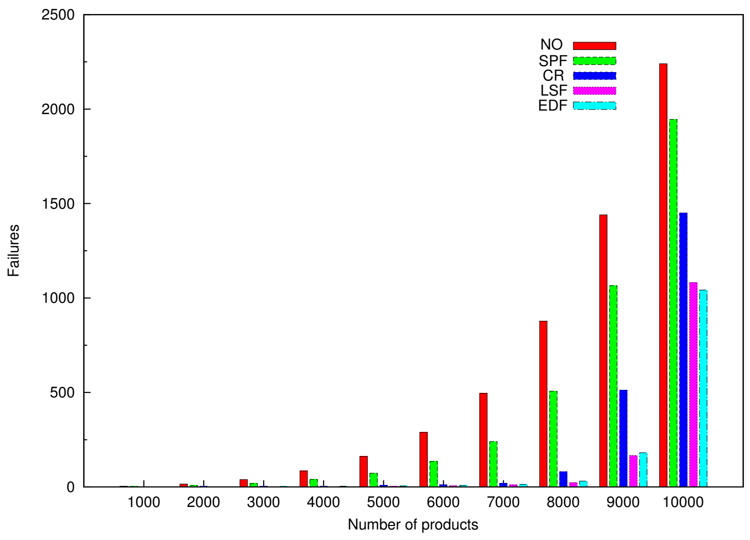


Figure 3.11: Failure-count for different scheduling algorithms

3.8.3 The effect of the R-parameter

The figures 3.15, 3.16 and 3.17 use EDF as scheduling scheme for 10000 products and show the effect of the range R as introduced in equation 3.1. The total number of basic production time steps performed by the grid (Figure 3.15) increases at first very fast by incrementing R, however for larger values of R it increases only slowly. The number of products in the grid (Figure 3.16) will increase linear in R. This means that for higher values of R an increasing storage capacity for products waiting for equiplets is needed. The percentage of failures (Figure 3.17) shows a sharp decrease in the beginning when we use a higher value for R. This figure shows a complimentary

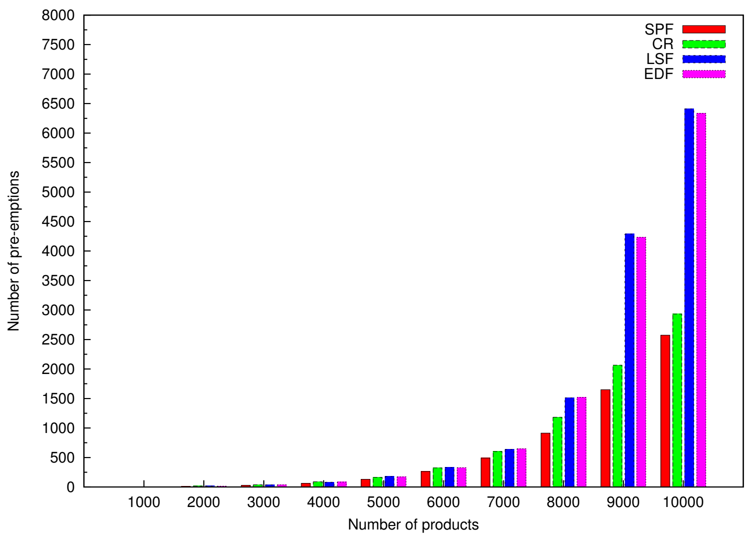


Figure 3.12: Number of preemptions for different scheduling

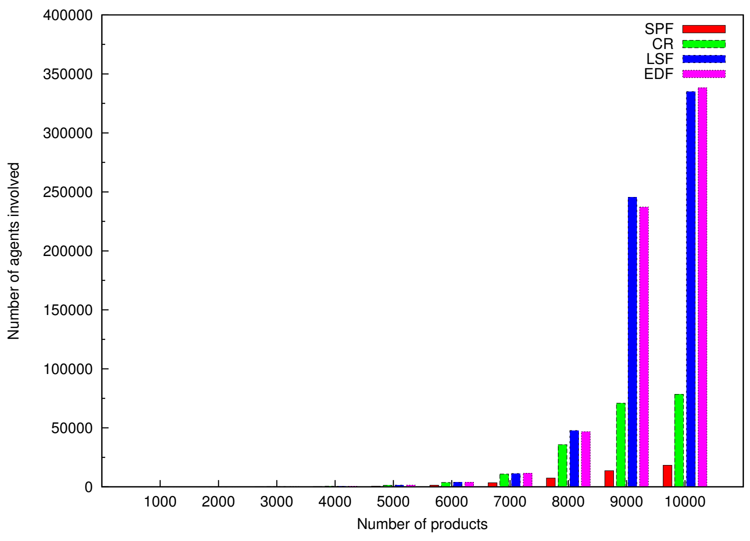


Figure 3.13: Number of agents involved in rescheduling

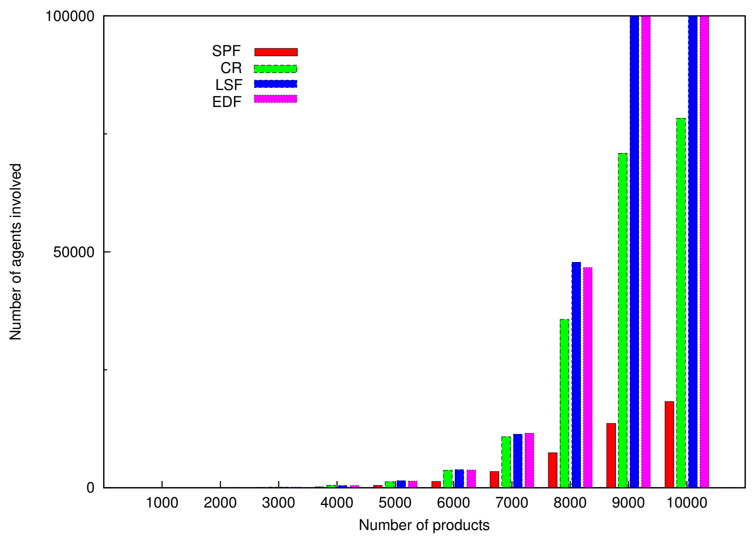


Figure 3.14: Detail of number of agents involved in rescheduling

behaviour of Figure 3.15.

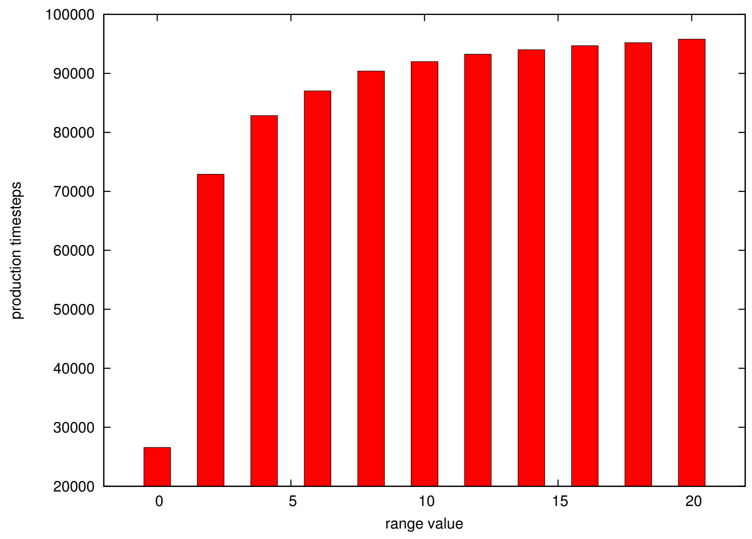


Figure 3.15: Total number of production time steps for different values of R

3.8.4 Effect of extra transport steps and multiple timeslots for steps

Two aspects are discussed here.

1. What is the effect of having steps that take a multiple of the unit time step used so far?
2. What is the effect if we take into account that in between two steps a transport slot is needed?

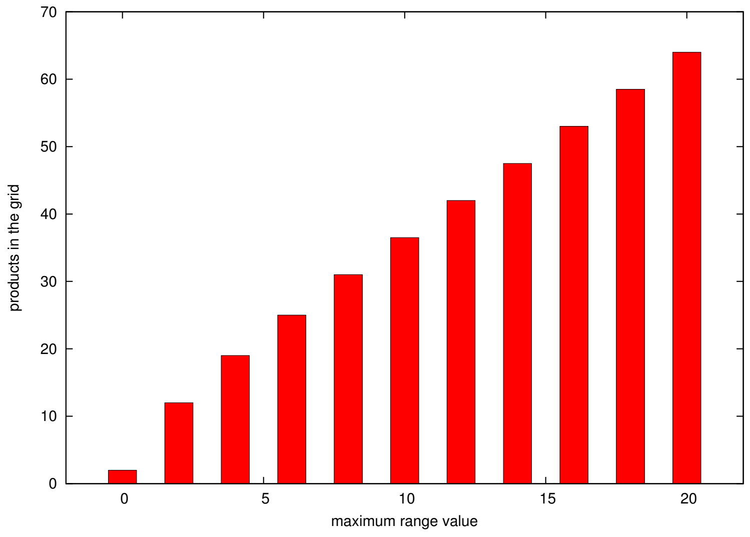


Figure 3.16: Average number of products in the grid for different values of R

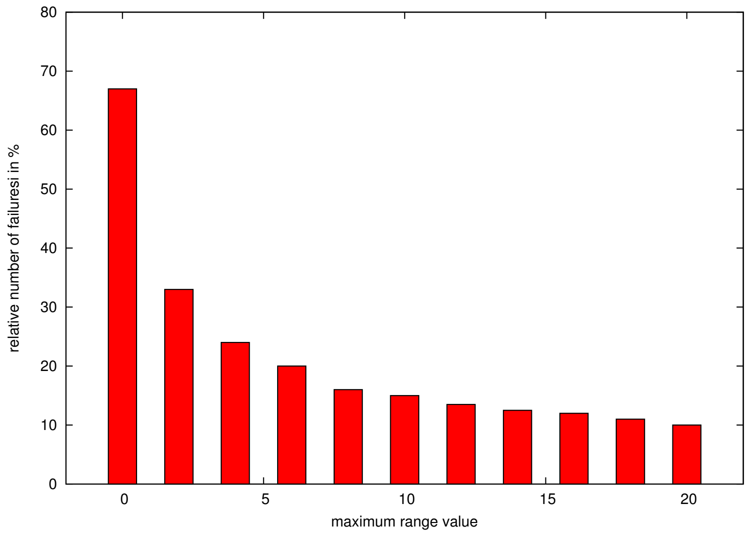


Figure 3.17: Percentage of failures for different values of R

The result of having varying length of the production step is shown if figures 3.18, 3.19 and 3.20. A non feasible test set has been used and the failures are counted for different scheduling types. As can be seen from the figures, the results are about the same, though a little bit worse than the one time unit step test in the situation of a flat and increasing test set. A longer step time is a stronger constraint, so the result being a bit worse is not surprising. In the decreasing test set, the number of agents per time-unit will be lower, making it easier for longer production steps to be scheduled.

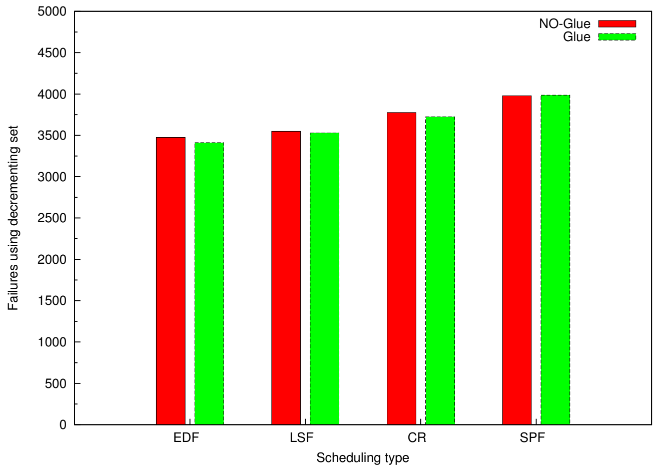


Figure 3.18: Effect of varying step length for different scheduling schemes for a decreasing set

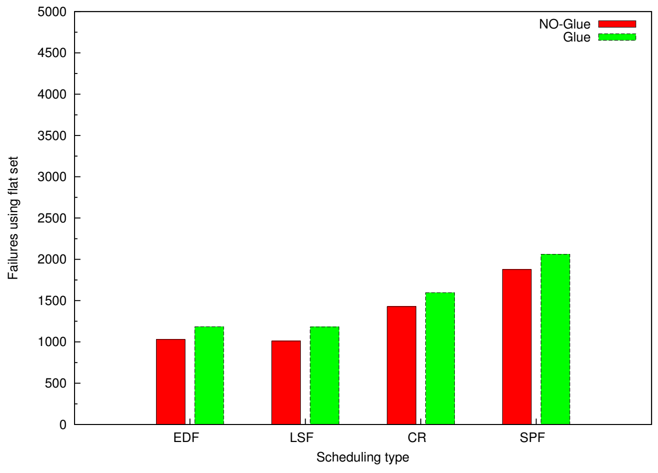


Figure 3.19: Effect of varying step length for different scheduling schemes for a flat set

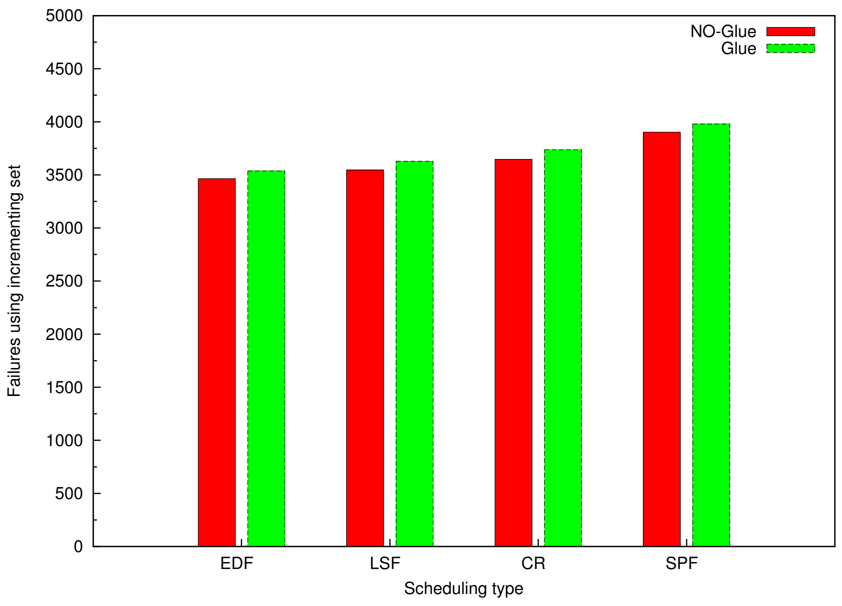


Figure 3.20: Effect of varying step length for different scheduling schemes for an increasing set

The three figures 3.21, 3.22 and 3.23 show the results when we introduce a time unit for transport between two consecutive steps. This will be the case in the real production grid, where a product has to travel from equiplet to equiplet. Like in the previous test, the simulation has been done for an increasing, a flat and a decreasing test set. Looking at the results, we see that for some scheduling types for the increasing and decreasing test set the number of failures is even lower than when no transport time unit has been used. Again the difference is not that big, but a remark should be made. To compensate for the extra time for transport, this transport time has been added to the so called range factor effecting in a later deadline. Every product has its deadline extended by the amount of timeslots needed for transport. If this had not been done the results would have been much worse compared to the situation where the transport time is neglected. That this result is worse when there is no compensation for the range factor R is also visible in the earlier firstfitplus approach depicted in Figure 3.4. In that situation there was also extra time inserted between production steps, but there it is has not been compensated for. In such a situation, it is more difficult to schedule the product before the deadline, due to the extra time units between production steps.

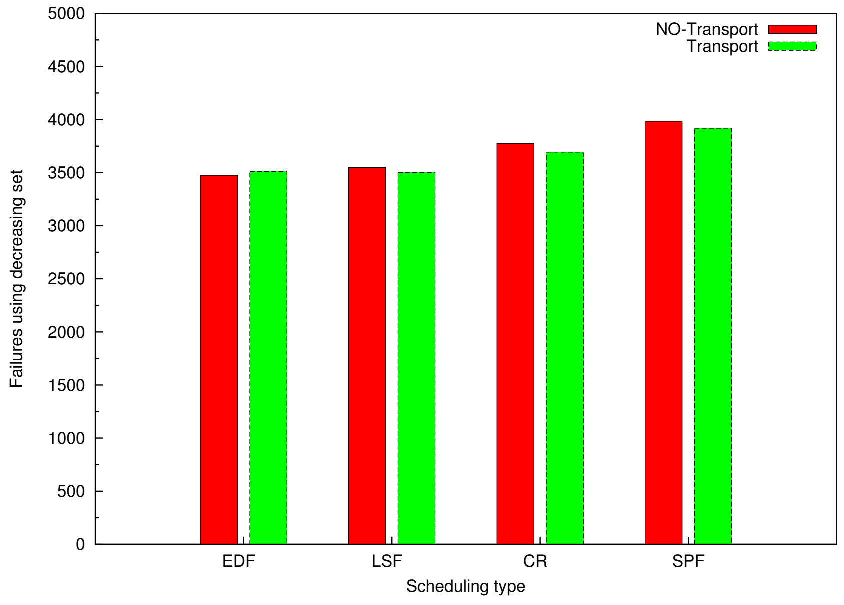


Figure 3.21: Effect of varying step length for different scheduling schemes for a decreasing set

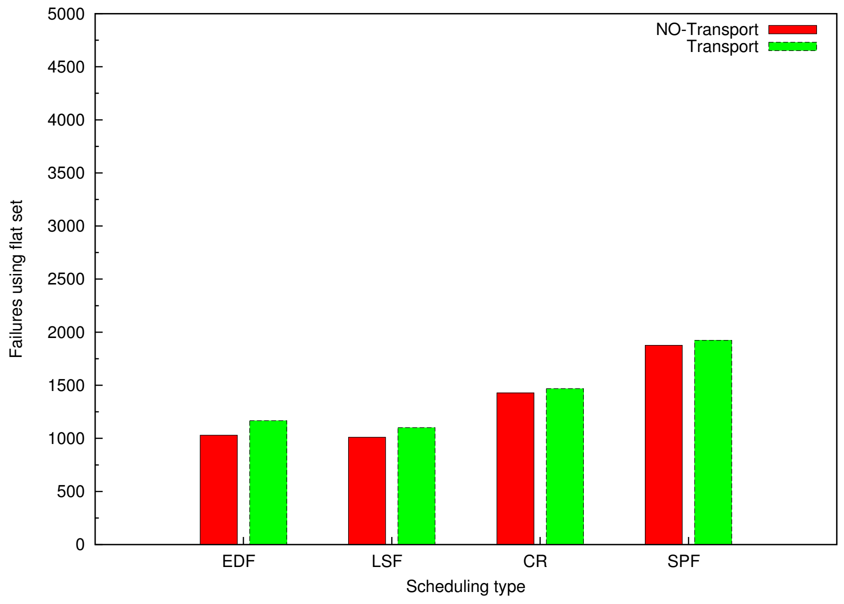


Figure 3.22: Effect of varying step length for different scheduling schemes for a flat set

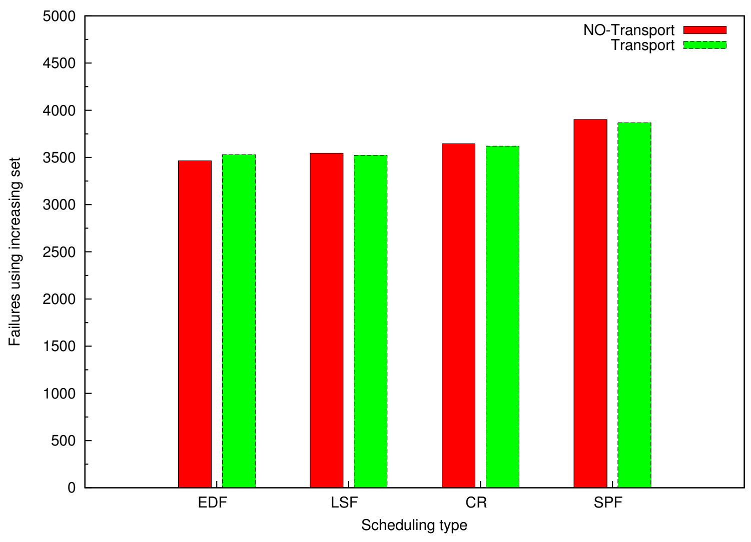


Figure 3.23: Effect of varying step length for different scheduling schemes for an increasing set

3.8.5 Grid behaviour under load

An important question about the scheduling behaviour is how many product agents are actually present in the grid during production. This aspect has already been studied when the effect of the range value was investigated. Now we will study the behaviour of the grid in the situation of three different test sets. These three sets are an increasing test set, a flat test set and a decreasing test set. Again 10000 time steps are considered in a grid of 10 equiplets, each offering a single unique production step. The range factor is again 20 and products have a set of steps ranging from 1 to 20. In the worst-case situation, the amount of agents can be calculated by taking the release-time and the deadline for all successfully scheduled agents and take these values as the begin time and the end-time of the agents in the grid. This results in figures 3.24, 3.25 and 3.26. When these figures are compared with the figures of the test set (figures 3.5, 3.6 and 3.7, a change in slope for the increasing and decreasing set can be seen. This is due to the fact that the grid is saturated and product agents are rejected at a high load. It is however a bit pessimistic to expect that all agents will live in the grid during all the time between release and deadline. If a product is finished before the deadline, the agent can leave the grid. So to calculate the actual number of agents in the grid we have to take the time from release to completion. This results in the plots shown in figures 3.27, 3.28 and 3.29. Interesing is to see that for the increasing load until about time step 4000, the number

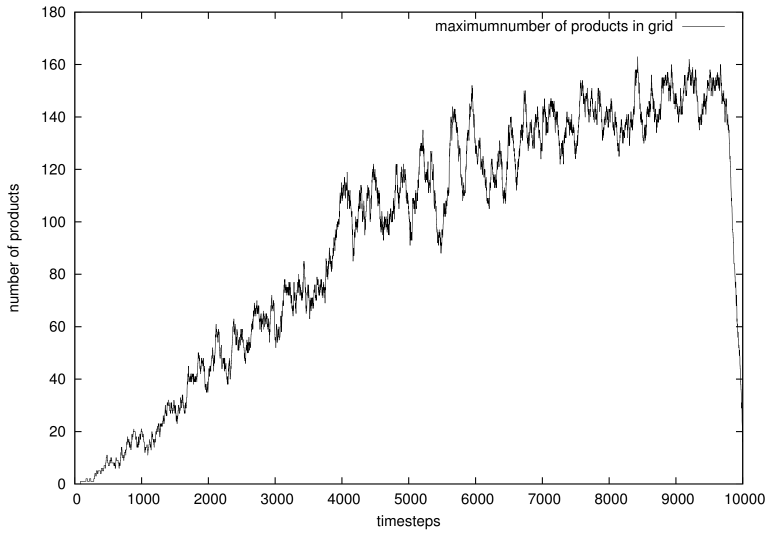


Figure 3.24: Worst case situation for increasing load

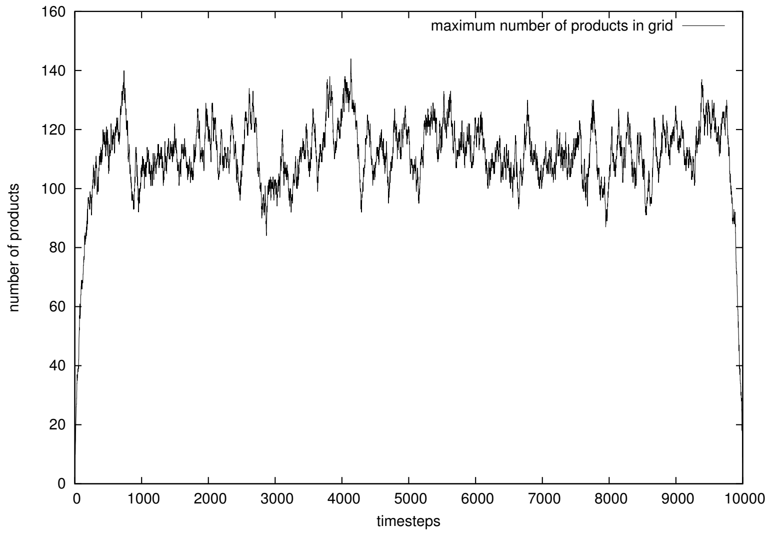


Figure 3.25: Worst case situation for flat load

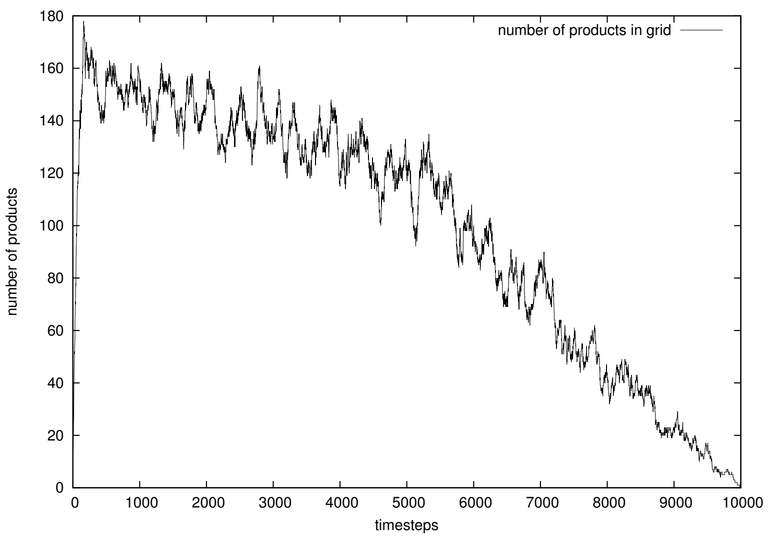


Figure 3.26: Worst case situation for decreasing load

of product agents is smaller than the number of equiplets and then it will quickly grow to a saturation value of about 65. A saturation value of 65 means that 55 products should be stored while only 10 products are actually worked on by the equiplets. To see what value results in a more feasible situation, a simulation was made by using again sets running from 1000 to 10000 agents and now again computing the average number of agents in the grid using the same assumption, that the product agent is present in the grid from release time to completion time. The result is shown in Figure 3.30.

In Figure 3.31 the average time to compute a feasible schedule in the simulator for a given set of product agents is plotted. The simulation was run on a standard low-end desktop PC with an Intel(R) Core(TM)2 CPU running at 1.86GHz, 2GByte of memory and Linux version 3.0.0-32-generic. The plot shows a comparable figure as the plots seen before. Scheduling a product in a heavy loaded grid takes much more time. Because there is a big difference between the maximum and minimum time in the figure, a table with the actual values is also included (see Table 3.9).

3.9 Some considerations about batches

So far scheduling of single unique products has been considered. The grid is also expected to work for small or medium-sized batches. The model introduced so far can also be used in the situation that many similar or almost similar products should be produced. The agent for such a small batch will

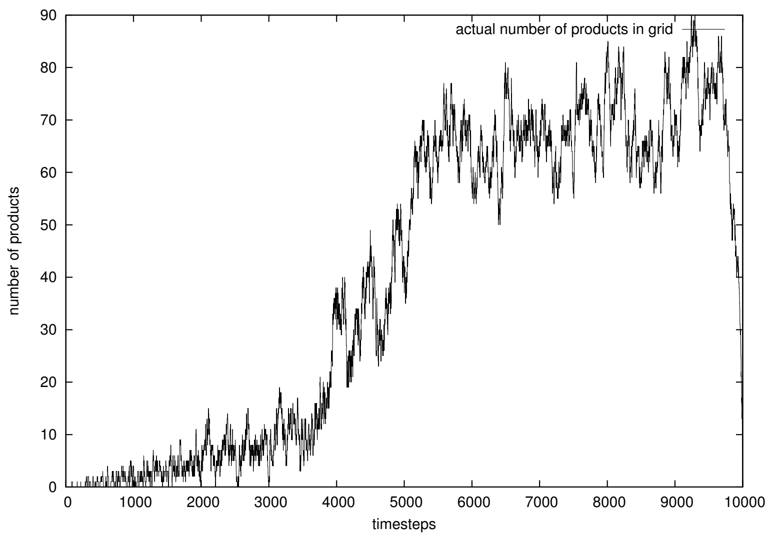


Figure 3.27: Actual situation for increasing load

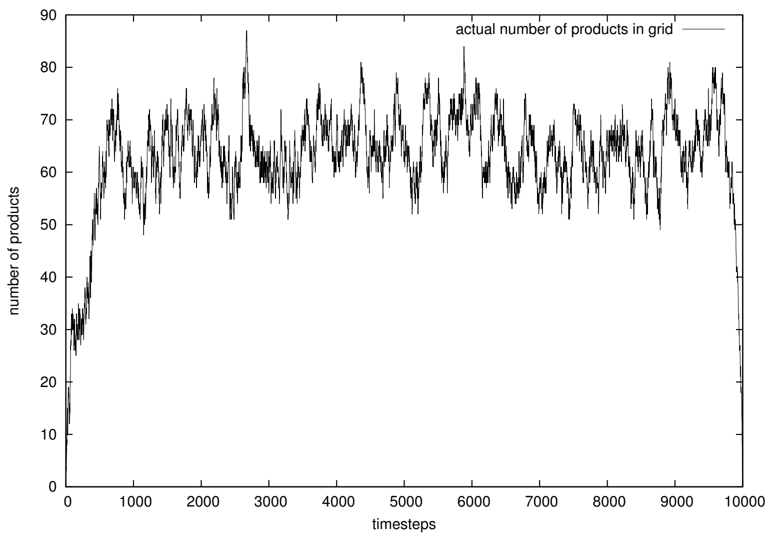


Figure 3.28: Actual situation for flat load

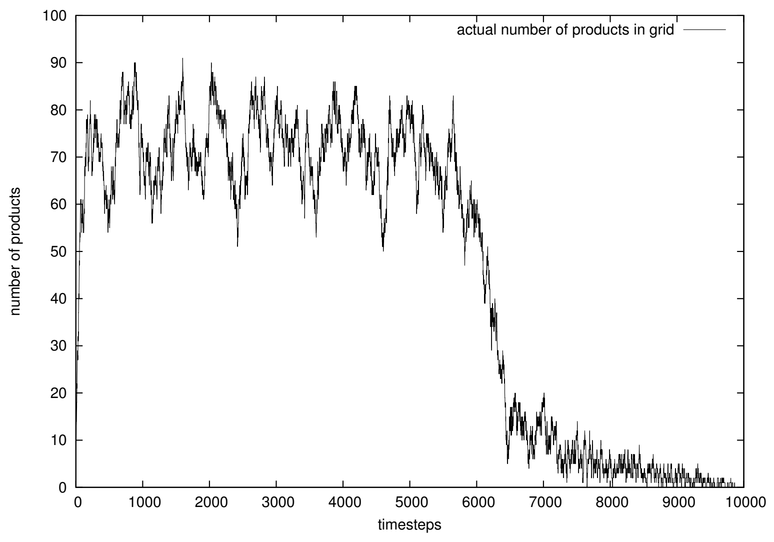


Figure 3.29: Actual situation for decreasing load

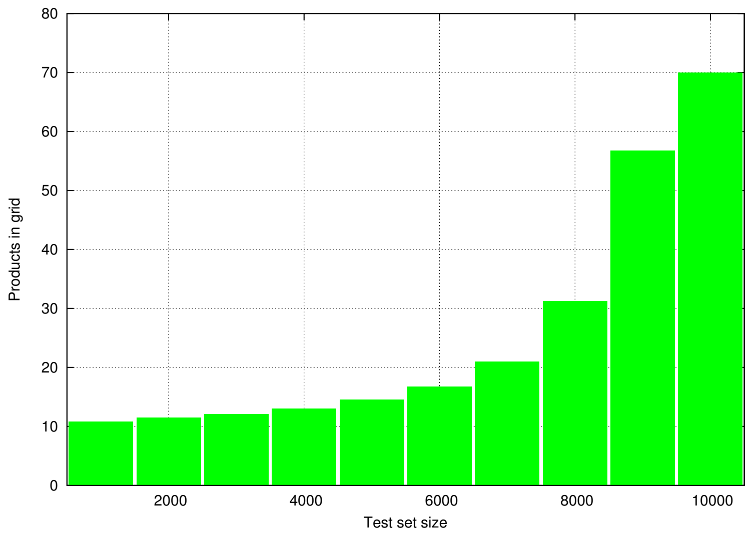


Figure 3.30: Actual number of products in the grid for different sizes of test sets

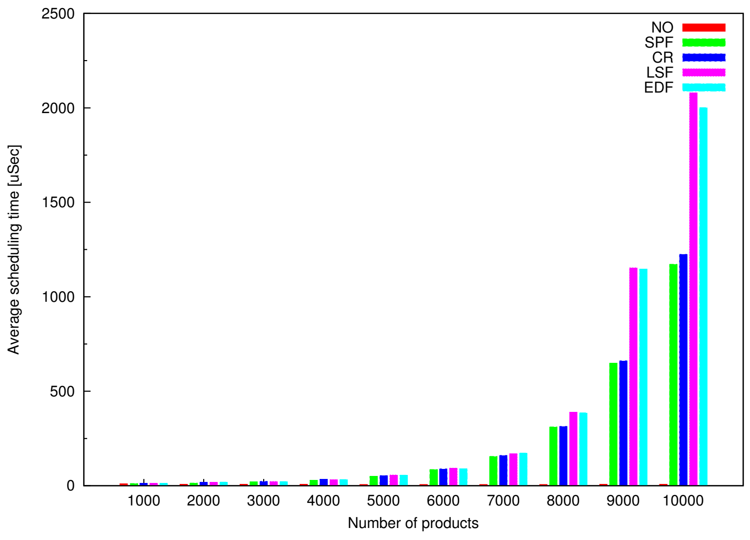


Figure 3.31: Scheduling time in $\mu Sec.$ for different scheduling algorithms for different sizes of test sets

Table 3.9: Average time in $\mu Sec.$ to schedule a product

Number of Products	NO	EDF	LSF	CR	SPF
1000	10	13	13	12	12
2000	8	19	18	18.5	14
3000	7.4	22.3	22.7	23	20.8
4000	7.2	33	32.5	34.5	29
5000	6.8	55.9	57.2	53.5	50.1
6000	6.7	90.1	93.3	89.2	86.1
7000	6.9	172.7	169.7	160.3	155.6
8000	6.9	386	389.7	314.3	311.2
9000	7.4	1146.6	1153.4	661.2	648.9
10000	7.7	2001.5	2079.8	1225	1172

spawn child agents to do the actual production guidance. To prevent the situation where hundreds of agents try to schedule the production, a minor adjustment should be make to the scheduling system. Two situations should be considered. When all steps take the same amount of time, the spawning of child agents will be at intervals having the same time as the time it will take to do a step. The first child agent will also communicate its scheduled product path and, if they are available, alternatives to the parent agent that

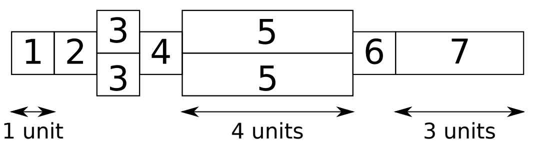


Figure 3.32: Actual number of product in the grid for different sizes of test sets

can hand this over to the second and all other child agents to be spawned. This way the communication overhead is reduced.

If the time for the production steps is not equal, the first child agents will find this out and report to the parent the amount of time for every step and also the amount of equiplets capable of performing a certain step. The parent will compute:

$$T_{max} = MAX(t_{step}/n_{eq.step})$$

This time will be the delay between the re-spawning of child agents, because this is the minimum time between the production of two successive products in the batch. In Figure 3.32 the situation is explained for a path of 7 steps. Step 3 can be performed by two equiplets. The same is true for step 5 that takes 4 time units. Because of the fact that two equiplets can handle this step it will reduce to effectively 2 time units. Step 7 takes 3 time units, but is only offered by one equiplet. In this situation $T_{max} = 3$

3.10 Related work

A good overview of multiagent scheduling is given by Weerdt and Clement (Weerdt and Clement, 2009). Several situations and restrictions for scheduling in multiagent systems are discussed. Special cases of multiagent scheduling are described in (Krogt et al., 2005) and (Weerdt et al., 2001). Other publications on multiagent scheduling include work of Rabello and van Hoeve. In (Rabello et al., 1999) innovative and balanced perspectives of multiagent approaches to agile scheduling are discussed, and several achieved results and developments are described. This paper is a good overview of what has been achieved but does not focus on the type of production scheduling that is needed for the production grid. In (Hoeve et al., 2007) an efficient method to compute provably optimal solutions for centralized deterministic multiagent scheduling problems is presented. The work is based on a multiagent system with a common goal and interdependency between the agents. In our situation the interdependency between agents is lacking and the scheduling problem itself is different. In (Ouelhadj et al., 2000) a multiagent monitoring is presented. This work focusses on monitoring a manufacturing plant. The approach we use monitors the production of every single product. The work of Xiang and Lee (Xiang and Lee, 2008) presents a scheduling multiagent-based solution using swarm intelligence. This work uses negotiating between job-agents and machine-agents for equal distribution of tasks among machines. In our system there might be no need for balancing the load between equiplets, because these production platforms are cheap and their use depends on what kind of production steps are needed at a certain moment. There are however possibilities discussed in Chapter 6 to introduce a way of load balancing. Karageorgos (Karageorgos et al., 2003) is based on agents for optimisation of production planning and logistics in a standard production environment, not a co-design of a production environment and its software infrastructure as in our case. The work of Wang (Wang et al., 2003) is also based on a standard production environment. A good overview of agile manufacturing and the use of agent technology can be found in (Paolucci and Sacile, 2005), where the focus is on the replacement of standard production management software by agent-based software. Standard production management software is reliable but not easy to reconfigure. Our work however proposes a new production paradigm based on a co-design of hardware as well as agent-based software.

3.11 Conclusion

In the given situation the scheduling in a multiagent-based system can use both LSF as well as EDF. Both are proven successful dynamic scheduling methods in real-time operating systems, and are rather easy to implement in a cooperative multiagent system as used in this production system. The disadvantage of EDF and LSF in real-time operating systems is unpredictable behaviour in case of an infeasible scheduling. This problem has been partly overcome by giving up an infeasible scheduling and thus not disrupting already claimed schedules. However care should be taken not to overload the grid, because infeasible scheduling has an dramatic impact on the performance of the production system. The scheduling scheme where negotiating is only used in case of failure introduces much less overhead than the scheme where negotiating is always performed between agents. The former approach seems to be the best solution so far for the grid production system. If the actual production time for all products is much less than the time between release time and deadline, a high load of the production grid can be achieved. However, this is only possible if the grid is capable of storing a certain amount of unfinished products.

The scheduling scheme presented here makes a mix a scheduling schemes for different product agents possible. This will be a subject for further research.

3.12 Summary

This chapter has been devoted to planning and scheduling. A description of the problem was presented and a description of scheduling concepts and production scheduling was given. A description of a simulation tool and its results and interpretation ended this chapter.

Chapter 4

Flexible transport in the grid

An important aspect of the production grid is the mechanism to transport all products under construction from one equiplet to the other. In standard line-based production, this is more straightforward, because the sequence of production steps is similar for all the products. The first part of this chapter is dedicated to research about the topology and position of the equiplets within the grid. The second part is dedicated to the transport itself. Some solutions offering agile transport are discussed with their advantages and disadvantages.

Parts of this chapter have been published in the proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2014) (Moergestel et al., 2014).

4.1 Introduction

In standard mass production, batch processing is widely accepted. The advantage of batch processing is that production equipment can be placed in a production line. A product only has to follow this line and all production steps will be performed. However, this set-up is not adequate for low cost small quantity production. For agile production of small quantities in a grid of reconfigurable production machines, equiplets, a different approach is needed. One of the challenges in this approach is the transport of the product between the equiplets during production. This chapter starts with a description of some heuristic methods to reduce the average path a product has to follow in the production grid. Another challenge is the stream of raw material or parts for production. For this challenge a specific solution is presented.

As discussed in the previous chapters, the production model that is presented in this thesis consists of a set of manufacturing machines. However the production is not pipeline-based because the aim of this model is to produce different products in parallel. Every product needs its own, possibly unique, set of manufacturing machines. Because the production is not pipeline-based, the transport between the manufacturing machines becomes an important issue.

4.2 Grid manufacturing

In grid production, manufacturing machines are placed in a grid topology. Every manufacturing machine offers one or more production steps and by combining a certain set of production steps, a product can be made. This means that when a product requires a gives set of production steps and the grid has these steps available, the product can be made. The software infrastructure that has been used in our grid, is agent-based. Agent technology opens the possibilities to let this grid operate and manufacture different kind of products in parallel, provided that the required production steps are available as explained in chapter 2.

4.2.1 Manufacturing model

The manufacturing machines that have been built in our research group are cheap and versatile. These machines are called equiplets and consist of a standardized frame and subsystem on which several different front-ends can be attached. The type of front-end specifies what product steps a certain equiplet can provide. This way every equiplet acts as a reconfigurable manufacturing system (RMS) (Koren et al., 1999).

The equiplet is in software represented by a so called equiplet agent. This agent advertises its production steps to a blackboard that is available in a multi agent system where also so-called product agents live. A product agent is responsible for the manufacturing of a single product and knows what to do, the equiplet agents knows how to do it. A product agent selects a set of equiplets based on the production steps it needs and tries to match these steps with the steps advertised by the equiplets.

4.2.2 Similarities and differences between batch and grid production

Both batch and grid production are based on the concept of a production step. In a batch environment these steps have the same sequence for all products. Also in batch production the duration of steps is normally the same, so a pipeline of a chain of production steps is easy to implement and effective. The drawback is that all products should be similar to make this concept work. In grid production the duration of steps can vary without disturbing the production. Also the sequence of steps can vary among products opening the possibility to produce several different products in parallel. The drawback here is the complication of different paths along the production machines. Instead of a transport belt or a similar solution, a much more complicated transport system is required (Bussmann et al., 2004). The transport system can be optimised if the position of the production machines within the grid is adapted to the set of paths that are required for production. This is the subject of the research described in the first part of this chapter.

The equiplets are reconfigurable machines. The product agents make their planning according to the capabilities offered by the equiplets. Combining this information the question arises: is it possible to adapt the positions of the equiplets in the grid, so that the average length of the paths of the products is shorter than in case of a random walk within the grid? The length of the path in the grid is also referred to as the amount of hops, where a hop is a path between two adjacent nodes. In our model the length of a path between two adjacent nodes is 1.

To explain in a more formal way the differences between batch production and grid production, consider a batch production system. This system can be represented by a tripartite graph as depicted in Figure 4.1. Every step (member of set S) matches one single production machine (member of set E). All products (P) use all available steps in a sequence, one by one. This

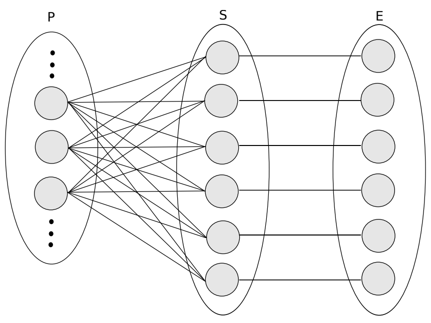


Figure 4.1: A batch process as a matching tripartite graph

tripartite graph can be transformed to the bipartite graph of Figure 4.4, where only products (P) and production machines (E) are involved. The production in a grid can be represented by the tripartite graph of Figure 4.3.

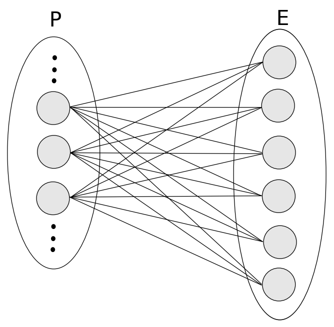


Figure 4.2: A batch process as a matching bipartite graph

Here it can be seen that not all products use all the available steps and some production machines (equiplets, denoted by E) offer more than a single production step. After the planning phase, the product agents have chosen

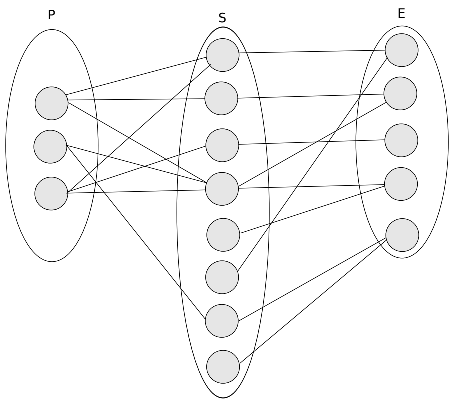


Figure 4.3: Grid-based manufacturing system

their set of equiplets and the tripartite graph can be transformed to the bipartite graph of Figure 4.4. This bipartite graph is in this case the result of a certain planning. If a step is offered by two or more equiplets and a product agent selects a different equiplet to perform a step, the resulting bipartite graph is also different. In case of batch-based production, there are no choices of this kind. Apart from the fact that this bipartite is not necessarily a complete graph (where every node from set P matches with all nodes from set E), there is another important difference. The edges of the graph are not used in a fixed sequence (in Figure 4.2 from top to bottom for every product), but the time they are active should be scheduled among all other edges involved. This planning and scheduling is described in the previous chapter.

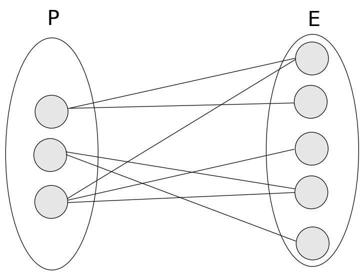


Figure 4.4: Grid-based manufacturing system

4.3 Adaption of the grid

There are several ways to adapt the grid to the production paths. Two possibilities used in this research are:

1. The grid can be configured or reconfigured according to information about the load or usage of the equiplets.

2. A grid configuration can be calculated according to the amount of inter-equiplet hops used by the production paths.

For both approaches an alternative brute force method could be used. For a reasonable sized grid (e.g. 4×4 or bigger) this requires a huge amount of calculation because of the fast increasing set of possible configurations being in the order of $(N \times N)!$ for an $N \times N$-grid. A better solution would be a heuristic approach that might lead to an acceptable result. To get a feeling for what heuristic might be a good approach, this research used the two aforementioned possibilities.

4.3.1 Reachability of nodes

The basic idea is based on the fact that nodes in a grid have different average values for reaching other nodes in the grid. For a 5×5-grid these values are shown in Figure 4.5. This means that from a corner point, the average path to any other node in the grid is 4, while the node in the center has an average path of 2.4 to any other node. This means that it is wise to place the most heavily used equiplet at the center and then grouping other heavily used equiplets around it. For this grouping two patterns have been used. The first pattern, grid pattern 1, is shown in Figure 4.6. Here we start at the hot-spot in the middle of the grid and construct a path among other nodes also having a low value for the average path, but we construct a path that has only one hop between two consecutive nodes. In Figure 4.7 an alternative

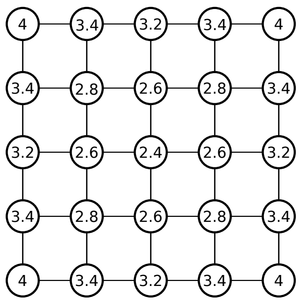


Figure 4.5: Reachability of nodes in the grid

path, grid pattern 2, is shown. This path follows the lists of shortest average paths that can be derived from Figure 4.5.

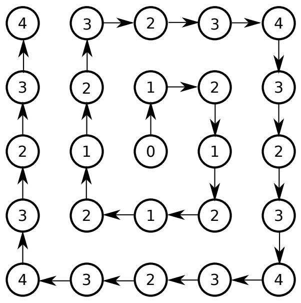


Figure 4.6: A path along the nodes

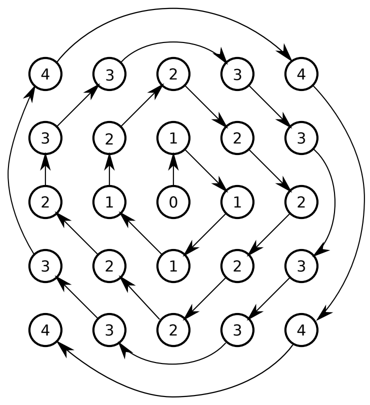


Figure 4.7: An alternative path along the nodes

We expect both patterns to give an improvement under certain circumstances. Pattern 2 because of the fact that heavily used equiplets are placed at easily reachable positions from any point in the grid. Pattern 1 looks similar, but has the order of its sequence separated by only one hop.

To test our approach, several scenarions are generated using a Monte Carlo method. We generated sets of production steps needed for a product and mapped these to the available equiplets. A set containing many different products was thus generated. From these artificially generated production sets a matrix (4.1) is constructed that has all the transitions between all pairs of equiplets. This matrix of transitions consists of elements α_{ij} having the number of transitions from equiplet i to equiplet j while α_{ji} shows the number of transitions from equiplet j to equiplet i.

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1n} \\ \alpha_{21} & a_{22} & \dots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \dots & \alpha_{nn} \end{pmatrix} \tag{4.1}$$

For computing purposes another matrix was also constructed using the values of matrix 4.1. In this matrix we only look at the transition between equiplets neglecting the direction of the transitions. This matrix is not an optimisation, but a different representation. This results is a matrix (4.2) having only non-zero values in the lower left triangle below the diagonal. Where the non-zero values $\beta_{ij} = \alpha_{ij} + \alpha_{ji} : \forall j < i$. In the next sections this type of matrix is referred to as a triangle matrix. In one of the computations in section 4.4 this triangle matrix is the starting point.

$$\begin{pmatrix} 0 & 0 & \dots & 0 \\ \beta_{21} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{n1} & \beta_{n2} & \dots & 0 \end{pmatrix} \tag{4.2}$$

4.3.2 Scenarios

To test the adaption software, several scenarios were generated. All scenarios are based on 10000 products that could use 25 equiplets in a 5×5 configuration. Following is a description of the scenarios:

A All products paths are randomly generated in a flat distribution without making some equiplets special. The usage is almost equally distributed over all equiplets as shown in Figure 4.8.

B Again a randomly generated set of product paths, but now there is a linear increase of usage among the equiplets, making equiplet 25 much more popular than equiplet 1. Figure 4.9 shows the distribution of the equiplet usage. The equiplets are numbered from 1 to 25.

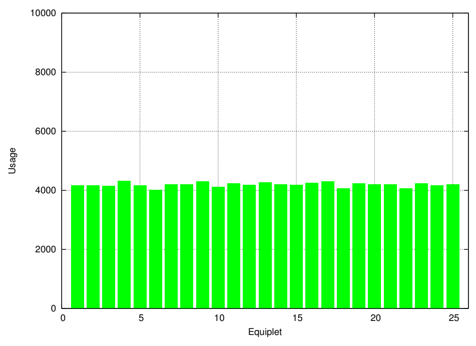


Figure 4.8: Equiplet usage distribution for scenario A

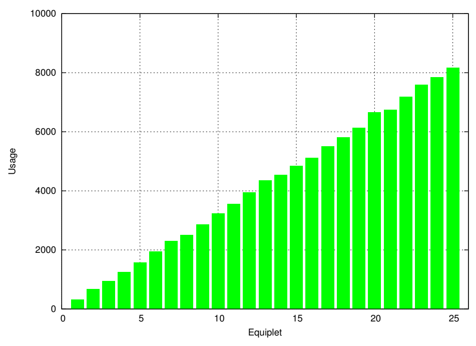


Figure 4.9: Equiplet usage distribution for scenario B

C In this set of product paths 25% of the equiplets are used twice as much. This might be the case if equiplets offer more than one production step. (see Figure 4.10).

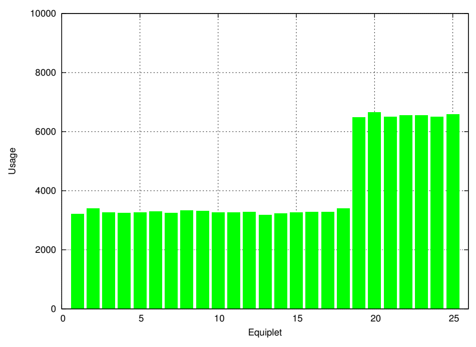


Figure 4.10: Equiplet usage distribution for scenario C

D A test set that is purely batch-based. 10000 products using all the 25 equiplets equally in a batch production situation (see Figure 4.11.

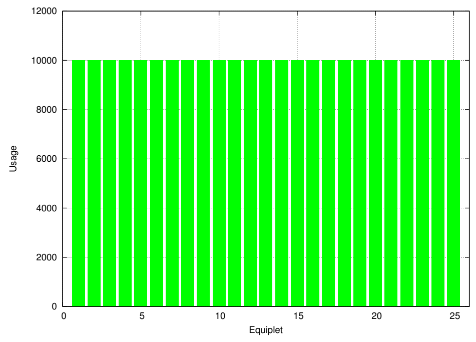


Figure 4.11: Equiplet usage distribution for scenario D

E A test set having several different products with comparable paths, but not of the same length (see Figure 4.12).

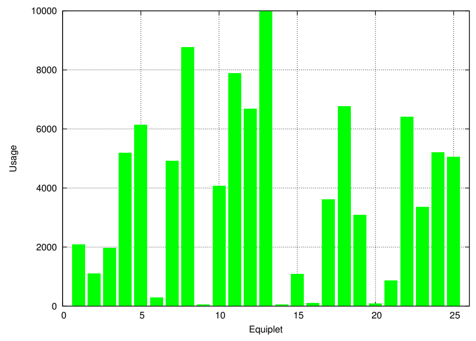


Figure 4.12: Equiplet usage distribution for scenario E

F A test set having 10 different products, resulting in 10 sets of 1000 similar products. (see Figure 4.13).

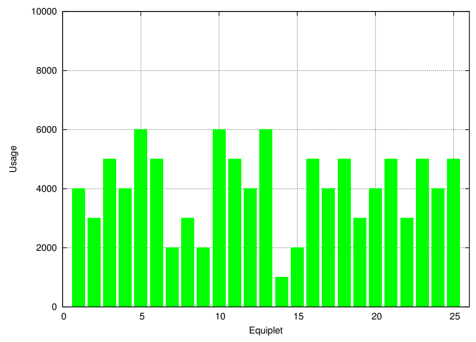


Figure 4.13: Equiplet usage distribution for scenario F

4.4 Computations

The first approach only looks at the usage of the equiplets and puts the most popular equiplet at the hot spot. The stepwise description of the computation looks like:

```
Construct the matrix of transitions
Transform it to a triangle
Calculate the total usage of an equiplet
Make a list of usage and equiplet-number
Sort this list according to usage,
   putting the highest on top
Generate a grid using the list and a
   grid pattern (1 or 2)
Use this grid to calculate the actual
   average pathlength
```

If the transitions are taken into account, the situation is a little bit more complicated.

```
Construct the matrix of transitions
Make a list of triplets of all transitions:
  #num eq-src eq-dst
Sort this list, putting the highest number
  (#num) on top
Create list of equiplets from this list
  starting at the top and from there
  following eq-dst as the next eq-src
IF a loop is detected, use next unused
  triplet in the list.
Use the list of equiplets to generate a
  grid pattern (1 or 2)
Use this grid to calculate the actual
   average pathlength
```

4.4.1 Grid versus line and circle

Before discussing the results of the computations described in de previous subsection, we first made some calculations on the average number of hops for a random path between nodes on a line, on a circle and in a grid. In Figure 4.14 the number of hops is plotted against $\sqrt{N}$, where N is the number of nodes among the line, the circle or in the grid. The increase of the average path length (number of hps) is the highest for nodes put on a line. So a random walk along a line is behaving bad, when the number of nodes

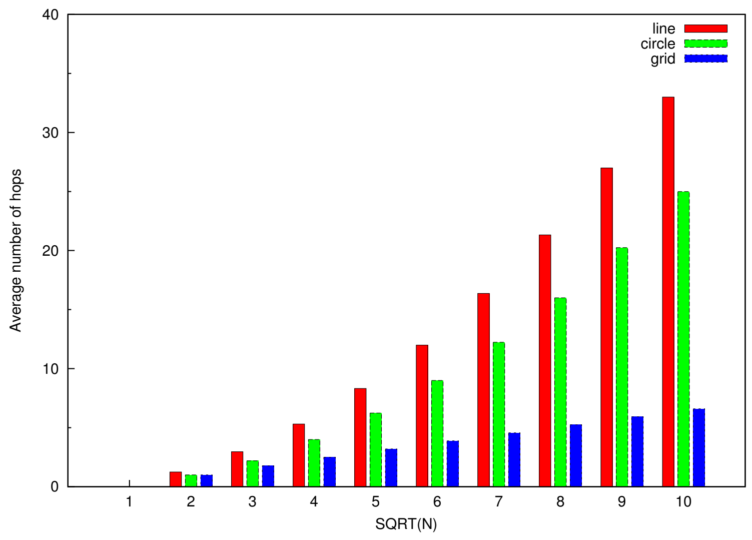


Figure 4.14: Number of hops for different configurations of N nodes

increases. When the nodes are placed on a circle, there is some improvement because of the effect that the largest distance now is over only halfway around the circle. When the same calculation is done for the grid, a slow and almost linear increase will be the result as shown in Figure 4.14. Thus from these three possibilities, the grid is by far the best choice.

4.4.2 Results

The results of the calculations are plotted as histograms. Every histogram shows the results for one scenario. The scenarios are already introduced and denoted by capital A to F. The numbered bars represent the following tests:

1. Random grid configuration, used as a reference measurement.

2. Using grid pattern 1 from Figure 4.6 with equiplets ordered according to usage.

3. Using gridpattern 2 from Figure 4.7 with equiplets ordered according to usage.

4. Again a random grid configuration (different from 1).

5. Using gridpattern 1 from Figure 4.6 with equiplets ordered according to transition frequency.

6. Using gridpattern 2 from Figure 4.7 with equiplets ordered according to transition frequency.

Figure 4.15 shows the results for the purely random situation. In this case no gain is possible, because all equiplets have almost the same load and all transistions have the same probability. Figure 4.16 shows the results for

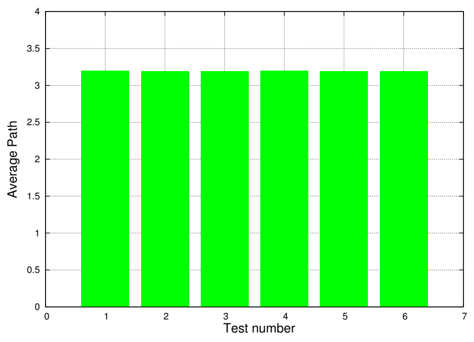


Figure 4.15: Scenario A with random use of equiplets

scenario B. Here we see a decrease of the average path length. There is not much difference between the different approaches. In Figure 4.17 the results

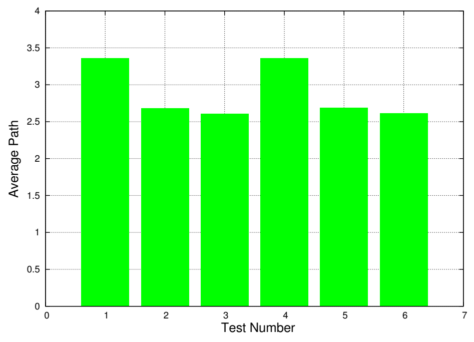


Figure 4.16: Scenario B with increasing use of equiplets

are shown for scenario C. Again a decrease of average path length. The best result is test number 5 where grid pattern 1 is used in combination with the number of inter-equiplet hops. The results of a pure batch scenario is shown in Figure 4.18. Normally in a batch the production machines are in-line separated by one single hop. This possibility is discovered by test 5, using grid pattern 1 in combination with the number of inter-equiplet hops. When we look at the results based on the usage of equiplets, there is no gain at all. This has to do with the fact that all equiplets are equally used, so sorting does not make any difference. In Figure 4.19 the results for scenario E are

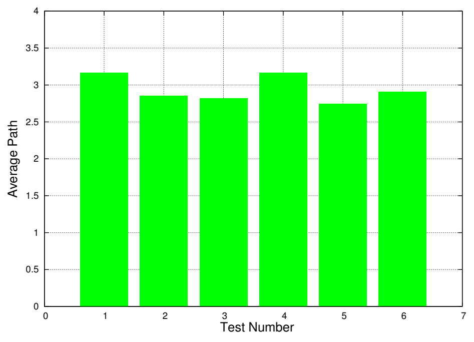


Figure 4.17: Scenario C with two overlapping sets of equiplets

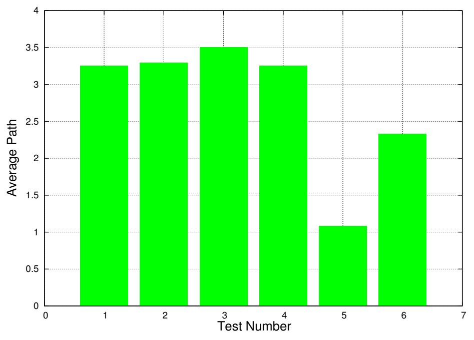


Figure 4.18: Scenario D with a single batch

shown. Here we also see a gain and in this case test 6, using grid pattern 2 in combination the number of inter-equiplet hops is the best solution. The

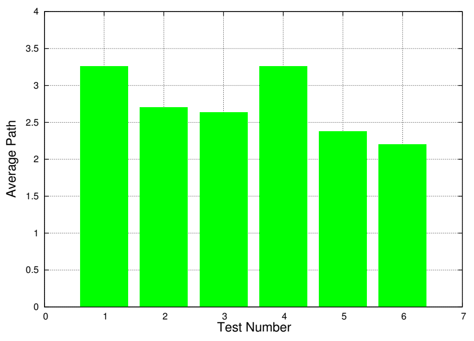


Figure 4.19: Scenario E with repeated tuples of equiplets

final histogram of Figure 4.20 shown the results for test 10. Here the gain is minimal but still available in three of the experiments.

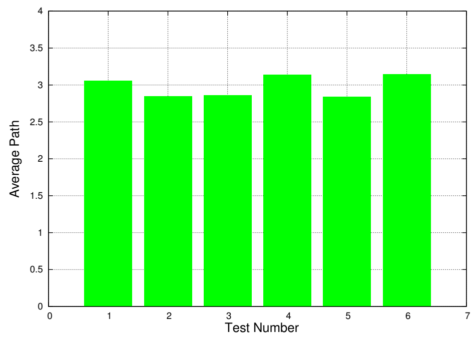


Figure 4.20: Scenario with 10 different batches of similar products

4.5 Discussion and future work

In Table 4.1, the percentage of reduction in hops is calculated for all scenarios and heuristics by taking the average of 3.2 and comparing it with the actual results shown in the graphs of the previous section. The highest profit is printed in bold typeface. It turns out that test 5 gives the best results, but not for all scenarios, having test 3 as a winner for scenario B and test 6 for scenario E. The approach presented here can be integrated with the

Table 4.1: Reduction of hops in %

Test	A	B	C	D	E	F
1, 4	0	0	0	0	0	0
2	0	16.3	10.9	-3	15.6	10.9
3	0	**18.5**	12	-9	17.8	10.6
5	0	16.3	**14.4**	**66.3**	25.6	**11.6**
6	0	18.4	9.4	28.2	**31.2**	2

grid software architecture. In the architecture, provisions have been made to implement a monitoring system. This system can produce the usage of the equiplets and the inter-equiplet transport in the past and also by looking at the planning blackboard the use and transport in the near future. This information can be used for optimising the grid. This way the grid control software can adapt to the production situation. In future research other grid patterns should be investigated and specially the scenarios in a real agile production environment should be studied to get an understanding of what might be adequate grid scenarios.

4.6 Streams within the grid

In the production grid there is at least the stream of products to be made. Another stream might be the stream of raw material, components or half products used as components. We will refer to this stream as the stream of components. These components could be stored inside the equiplets, but in that case there is still a stream of supply needed in case the locally stored components run short. In Figure 4.21 this situation is shown for four supply streams. This increases the logistic complexity of the grid model. In the

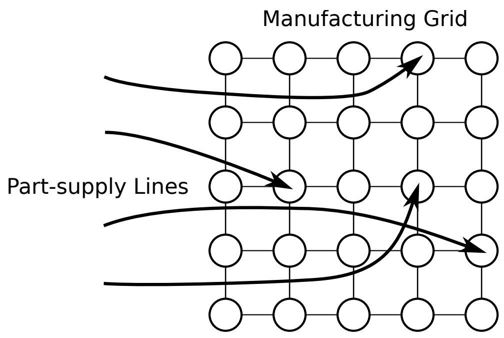


Figure 4.21: Streams of supply

next subsection models will be introduced that alleviates the complexity by combining the stream of products with the stream of components within the grid and keeping the supply stream of components outside the grid.

4.6.1 Buiding box model

In the building box model, a tray is loaded with all the components to create the product. To maintain agility, this set of components can be different for every single product. Before entering the grid, the tray is filled by passing through a pipeline with devices providing the components. In this phase a building box is created that will be used by the grid to assemble the product. The equiplets in the grid are only used for assembling purposes. Figure 4.22 shows the setup.

4.6.2 Solution of problems with building box model

A problem with the previous setup is the fact that more complex products should be built by combining subparts that should be constructed first. In the previously presented setup all parts for the subparts should be collected in the building box, making the assembling process more complicated. Another

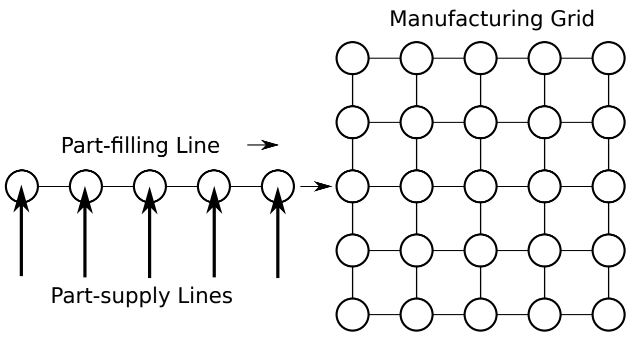


Figure 4.22: Production system with supply filling pipeline

disadvantage of putting all components for all subparts together in a building box is that this slows down the production time, because normally subparts can be made in parallel. A solution for these problems is shown in the setup of Figure 4.23. In Figure 4.23 a feedback of subparts to the supply line is

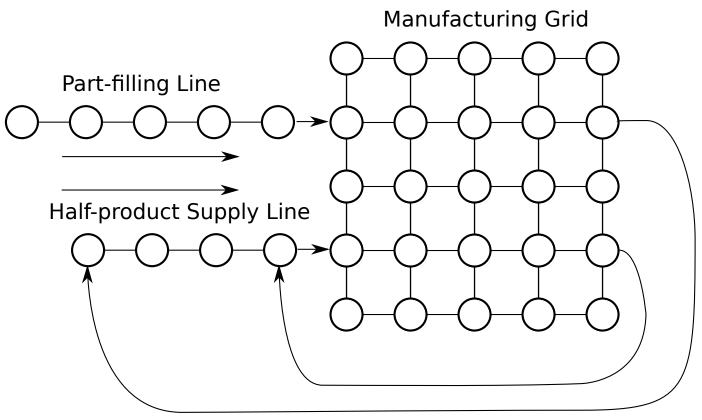


Figure 4.23: Production system with loops

possible and subparts can be made in parallel each having its own building box.

4.6.3 Including product inspection and retrying

The next refinement of the system is presented in Figure 4.24. Here a set of special test nodes has been added to the system. These nodes are actually also equiplets, but these equiplets have a front-end that makes them suited for testing and inspecting final products as well as subparts that should be used for more complicated products. A test can also result in a reject and this will also inform the product agent about the failure. If the product agent

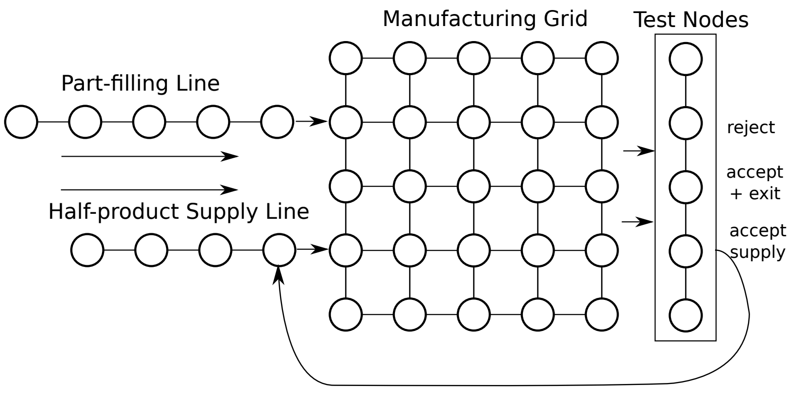


Figure 4.24: Production system with test and loops

is a child agent constructing a subpart, it should consult the parent agent if a retry should be done. In case the failure happens to the product agent for the final product, it should ask its maker what to do.

4.7 Possible transport systems

In this section transport possibilities are discussed. In batch processing, all products follow the same path. In grid processing, paths are different. Because cheap mass production used to be batched-based, most transport systems in production fit well in the batch approach, however for random walks in the grid different solutions are needed.

4.7.1 Conveyor belt-based systems

A conveyor belt is a common device to transport material. Several types are in use in the industry. Without going into detail, some kind of classification will be presented here:

- Belts for continuous transport in one direction.
- Belts with stepwise transport from station to station. These types of belts can be used in batch environments, where every step takes the same amount of time and the object should be at rest when a product step is executed.
- Belts with transport is two directions. This can also be realised by using two one direction belts, working in opposite direction.

In (Bussmann et al., 2004), an agent-based production system is built using transport belts in two directions where a switch mechanism can move a product from one belt to another (see Figure 4.25). A special switch-agent is controlling the switches and thus controlling the flow of a product along the production machines. This concept fits well in the system developed by

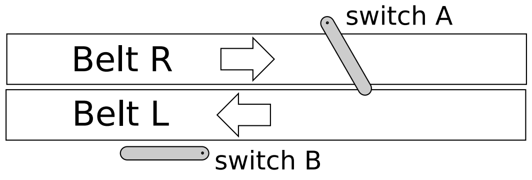


Figure 4.25: Bidirectional conveyor system used by Bussmann

Bussmann, because the system is actually a batch-oriented system. In a grid the use of conveyor belts might be considered, but for agile transport several problems arise, giving rise to complicated solutions:

- Should the direction in the grid consist of one-way paths or should be chosen for bidirectional transport?
- A product should be removed from the moving belt during the execution of a product step. A stepwise transport is inadequate, because of the fact that production steps can have different execution times in our agile model. This removal could be done by a switch mechanism as used by Bussmann, but every equiplet should also have it own switch to move the product back to the belt.
- Because the grid does not have a line structure for reasons explained in the first part of this chapter, a lot of crossings should be implemented. These crossing can also be realised with conveyor belt techniques, but it will make the transport system as a whole expensive and perhaps error-prone (Salvendy, 2001). In Figure 4.26 such a crossing is shown. It consists of two sets of small wheels rotating perpendicular. By raising one set of wheels the product can be moved according to the direction of the rotating wheels. At the cross-section itself the other set can be raised while the original set will be lowered, moving the product in a perpendicular direction.

4.7.2 Autonomous transport

An alternative for conveyor belts is the use of automatic guided vehicles (AGV). An AGV is a mobile robot that follows certain given routes on the

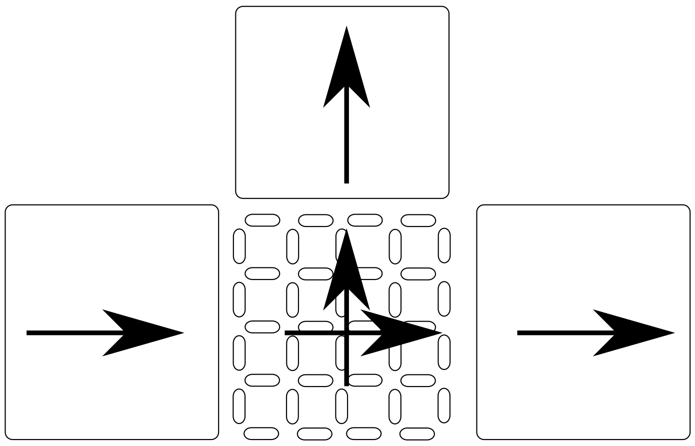


Figure 4.26: A crossection in a conveyor belt system

floor or uses vision, ultrasonic sonar or lasers to navigate. These AGVs are already used in the industry mostly for transport, but they are also used as moving assembly platforms. This last application is just what is needed in the agile manufacturing grid. The AGV solution used to be expensive compared to conveyor belts but some remarks should be made about that:

- These AGV offer a very flexible way for transport that fits better in non-pipeline situations.
- Low cost AGV platforms are now available.
- From the product agent view, an AGV is like an equiplet, offering the possibility to move from A to B. This makes the implementation fit seamlessly in our production model.
- a conveyor-belt solution that fits the requirements needed in grid productions will turn out to be a complicated and expensive system.

In the grid a set of these AGVs will transport the product between equiplets and will be directed to the next destination by product agents.

AGV system components

An AGV itself is a driverless mobile robot platform or vehicle. This AGV is mostly a battery-powered system. To use an AGV, a travel path should be available. When more than one AGV is used on the travel path, a control system should manage the traffic and prevent collisions between the AGVs or prevent deadlock situations. The control system can be centralised or decentralised.

AGV navigation

There are plenty ways in which navigation of AGVs has been implemented. The first division in techniques can be made, based on the fact if the travel path itself is specially prepared to be used for AGV. This can be done by:

- Putting wires in the path the AGV can sense and follow.
- Using magnetic tape to guide the AGV.
- Using coloured paths, by using adhesive tape on the path to direct the AGV.
- Using transponders, so the AGV can localise itself.

An example of the pattern of a guiding path with wires, magnetic tape or coloured tape is given in Figure 4.27. The second type of AGV does not

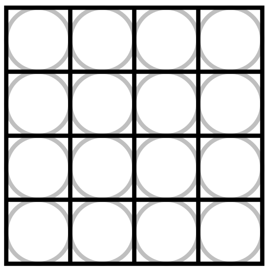


Figure 4.27: Path guiding pattern in a 4x4 grid

require a special prepared path. In that case navigation is done by using:

- Laser rangefinders.
- Ultrasonic distance sensors.
- Vision systems.

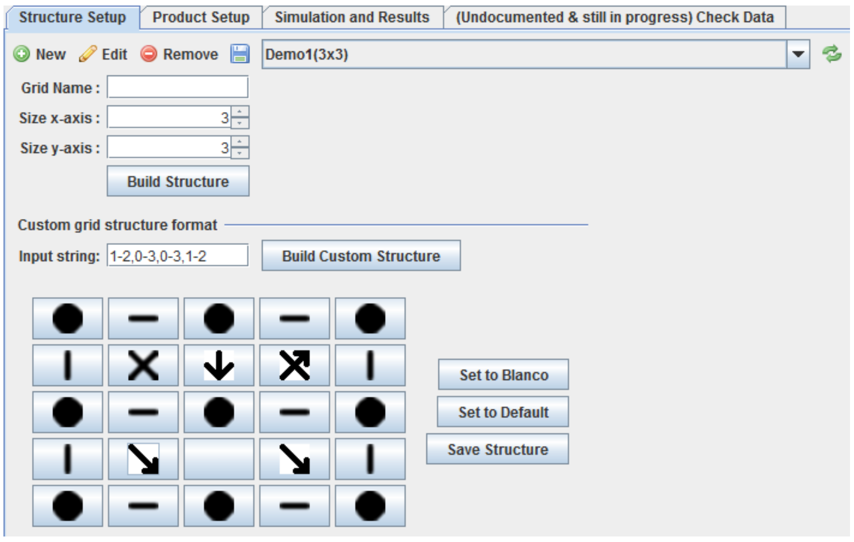
In the grid the equiplets are not far apart, so the length of a path to a neighbour-equiplet is less than one meter.

4.8 Path planning

A path planning tool has been built, to calculate a path a certain product has to follow along the equiplets. The Dijkstra path algorithm (Dijkstra, 1959) has been used. We start in Section 4.8.1 with a description of the tool followed by Section 4.8.2.

4.8.1 The path planning simulation tool

The tool can work on different grid transport patterns. In Figure 4.28 the GUI of the simulator is shown. Using this interface a grid can be specified as an $N \times M$ grid. Properties of the interconnection between nodes can also be chosen. In the figure bidirectional and unidirectional paths are displayed and also diagonal connections with all the directional possibilities can be chosen. This selection can be done by clicking on the boxes that connect the nodes. A selection of the type of connection can thus be made. When the box is blank, this means that there is no transport possible. Using this tool



Demo1(3x3)

Grid Name :

Size x-axis : 3

Size y-axis : 3

Input string: 1-2,0-3,0-3,1-2

Figure 4.28: GUI of the simulation tool

it is possible to analyse the paths chosen by product agents along the nodes (representing equiplets). A possible result is shown in Figure 4.29, where several paths are plotted for different arrangements of grids.

4.8.2 Results

To investigate the average pathlength in the grid for different paths, several structures have been investigated.

- A fully connected grid. where all paths are bidirectional paths as in Figure 4.30.
- A grid where all paths are bidirectional, but this design has removed the crossings as in Figure 4.31. This structure could be implemented by conveyor belts in combination with switches.

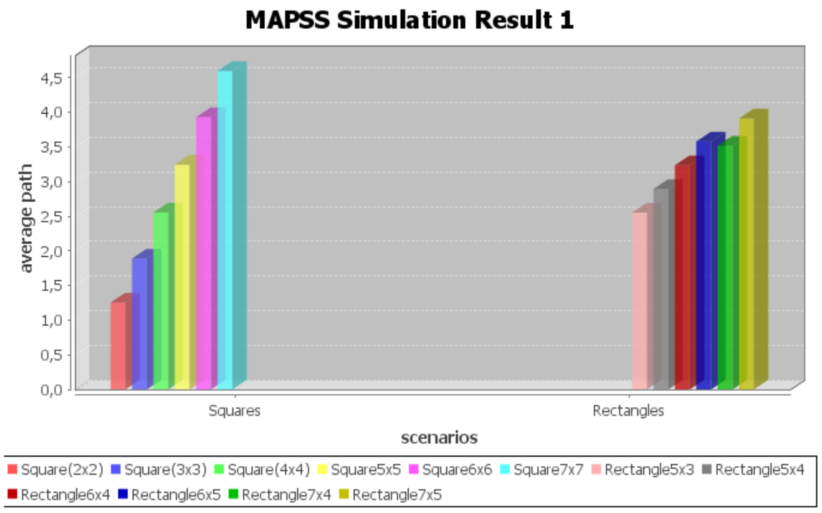


Figure 4.29: Example of a simulation result

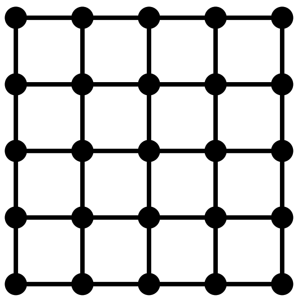


Figure 4.30: Fully connected grid

- A structure with five unidirectional paths and two bidirectional paths as in Figure 4.31. This structure is also a possible implementation with conveyor belts.
- A structure with bidirectional paths combined in a single backbone as in Figure 4.33.
- A structure with five bidirectional paths and two unidirectional paths as in Figure 4.34.
- A fully connected grid, but now with half of the paths unidirectional as in Figure 4.35.

For all these structures the average path is the result from a simulation of 1000 product agents, all having a random walk within the grid. Each agent also has a random set of equiplets it has to visit ranging from 2 to 50

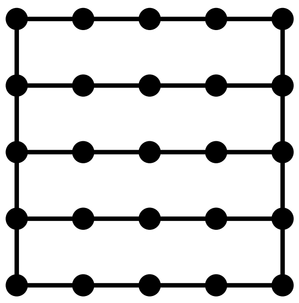


Figure 4.31: Grid with crossings removed

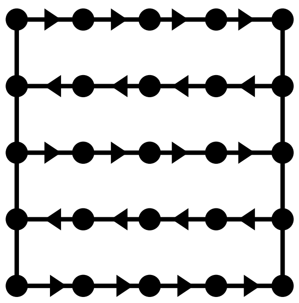


Figure 4.32: Grid with unidirectional paths and crossings removed

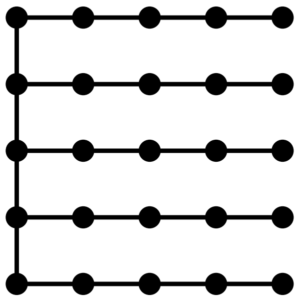


Figure 4.33: E-shaped grid

equiplets per agent. If the paths have no crossings, a conveyor belt might be used, because crossing belts will result in a more complex system. All structures can also be implemented with AGVs.

The results of the simulation are given in a table and are plotted in a histogram 4.36. In Table 4.2 a second outcome from the simulation is also shown. This is the percentage of agents that could find an alternative path of the same length. This result is of interest when in a traffic control implementation, alternative paths become important.

As could be expected, the best result is achieved in the fully connected grid with bidirectional paths. Changing the grid to structure 6 with unidi-

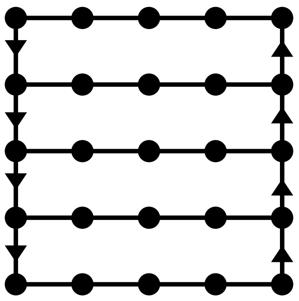


Figure 4.34: Grid with bidirectional paths, unidirectional backbones and crossings removed

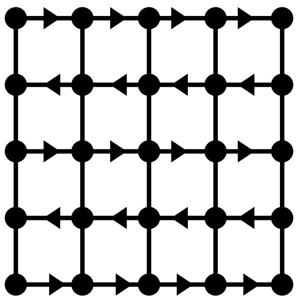


Figure 4.35: Grid with unidirectional horizontal paths

Table 4.2: Results of the simulatiom

Structure	1	2	3	4	5	6
Average path	3.2	3.9	6.4	5.1	6.0	3.6
% Alternatives	60	16.7	8.4	0	0	27

rectional paths, results in only a small penalty. This structure could also be useful in a AGV-based transport system, reducing the collision problem because of the one-way paths used. Both structures also offer a relatively high percentage of alternative paths, that could also be useful in an AGV-based system. The structures that fit a conveyor belt solution show a pathlength that is considerably higher. For the agile and agent-based manufacturing the buildingbox as well as the AGV-based system offers advantages:

- By using a building box, the transport of parts to the assembling machines (equiplets) is combined with the transport of the product to be made. It will not happen that a part is not available during manufacturing.

- Because the product as well as it parts use one particular AGV dur-

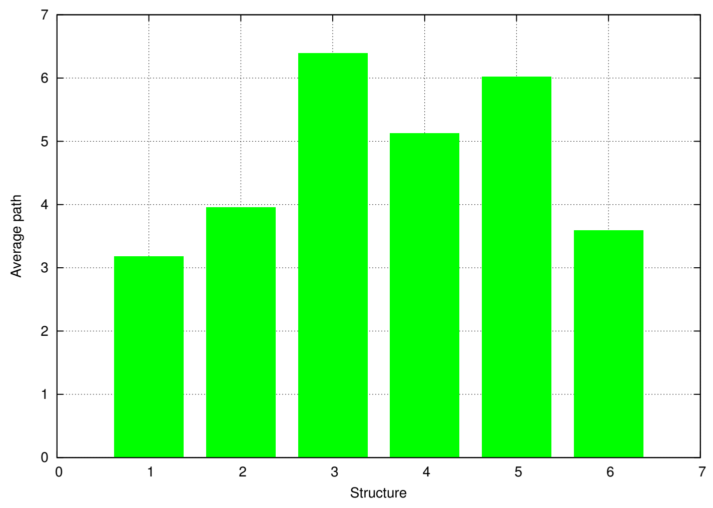


Figure 4.36: Simulation Results

ing the production, there is never a competition for AGVs during the manufacturing process.

- An AGV can use the full possibility and advantage of the grid-based system being a compact design resulting in short average paths.

- The product agent knows which equiplets it should visit and thus can use the AGV in the same way as an equiplet. The product agent can instruct the AGV agent to bring it to the next equiplet in the same way as it can instruct an equiplet agent to perform a production step.

- As a movable assembly platform, the AGV can move the product under construction in any direction in the x-y plane, while it can rotate the product around the Z-axis thus delevering extra degrees of freedom, or expanding the working area of a (pick and place) robot.

- If an AGV fails during production the problem can be isolated and other AGVs can continue to work. In a conveyor belt system a failing conveyor might block the whole production process.

There are also some disadvantages.

- There should be a provision for charging the battery of the AGV. This provision could be attached to every equiplet, because an AGV will visit equiplets regularly.

- Simulations show that the number of agents in the grid shows a strong increase in a grid that is loaded over 80%. This is shown in figure 3.30. To prevent the situation of too many products (each having its own AGV) in the grid, the load should be kept under 80%. In standard industrial production, the usage of the production machines should be high, because of the high cost of these machines. However, equiplets are low cost production devices and an average load below 80% should be no problem. Compare this with the situation in the printing industry. Printing devices used to be expensive and should be used as much as possible. Printing a single document was costly and the price would drop if multiple copies of the same document were produced. Today, printing devices with comparable quality are low cost devices. In many offices these printers spend most of there time in standby mode.

- Only products that fit within the buildingbox manufacturing model can be made. However, this is not a disadvantage that is specific for AVGs, but it is actually a grid-based production property that had been chosen.

4.9 Related work

In (Koren et al., 1999) the concepts of reconfigurable manufacturing systems are introduced and explained. A more recent article about this subject can be found in (Bensmaine et al., 2013). In this work, to take full advantage of the reconfigurability of RMSs, a new approach is proposed using genetic algorithms and a simulation-based optimization for process planning for a single product type. The proposed approach copes with market uncertainty and demands fluctuation in order to satisfy demands within their deadlines and with a minimum total cost. Our work is agent-based and is not limited to a single product type. The transport system within production environments is intensively studied, however these studies focus on technical solutions within batch processing environments. In (Bussmann et al., 2004) the transport system in combination with an agent-based infrastructure is introduced. In this work the conveyor belt system is adapted to a more agile transport system. However it is actually still a batch production system.

4.10 Conclusion

For the transport within the grid the use of cheap AGVs seems to be an adequate solution. An AGV fits in the agent model because for the product agent the AGV is similar to an equiplet. Instead of a production step, an AGV can deliver a transport step. Another advantage is the fact that one failing AGV does not disrupt the production grid as a whole. The building box model fits well in the concept of agile manufacturing, reducing the amount of transport within the grid itself.

4.11 Summary

In this chapter the focus was on the grid infrastructure. The aspect of reconfigurability has been investigated as well as the way the production grid might be used. In Chapter 3 the path was adapted to the grid, here the adapting of the grid itself has been investigated. The question of how the streams within the grid could be controlled, resulted in the use of the building box model in combination with an AGV-based transport system.

Chapter 5

Product agents beyond manufacturing

So far we have seen the role of the product agents during manufacturing. In this chapter the focus will be on the role of the product agent in other phases of the life cycle of a product. During manufacturing, the product agent collected valuable information. In this chapter we will see how this information can be used in other phases of the life-cycle of a product and other roles of the product agent. Parts of this chapter have been published in the proceeding of the International Conference on Computer-aided Manufacturing and Design (CMD 2010) (Moergestel et al., 2010a), the International Symposium on Autonomous Distributed Systems (ISADS2013) (Moergestel et al., 2013e), the International Conference on Agents and Artificial Intelligence (ICAART 2013) (Moergestel et al., 2013c) and the Workshop on Agent Technology in Ambient Intelligence held at the International Conference on Control Systems and Computer Science (CSCS 2013) (Moergestel et al., 2013a).

5.1 Introduction

Agent technology for agile manufacturing was the starting point of this research. In this research the concept of a product agent was introduced. Every product to be made starts as a software entity or agent that is programmed to meet its goal: the production of a single product. To be able to reach its goal this agent knows what should be done to create the product. This entity is called a product agent and it guides the product along the production cells to be used for manufacturing and it will collect all kinds of important manufacturing data during the production process. When the product is finished, this agent has all the manufacturing details and this agent is still available for further use containing valuable information about the product. The next step in this approach is to investigate and study the roles of this product agent in the other phases of the life cycle of the product. First some general concepts of agents in products are discussed. Three special case studies will be presented next. In the first case study the situation of adding a product agent to an existing device will be explained. This situation occurs when a product has been made without using the manufacturing concept presented in the previous chapters. This agent has a monitoring function. Monitoring of computer networks and complex technical systems like aeroplanes is common practice. In this case the use of a monitoring agent in an arbitrary product is discussed. The product itself could be any product with sufficient hardware capabilities. The focus is on the product enhancement by adding an embedded agent. This product agent can represent the product in the Internet of Things and it can also be a member of a multiagent system. In this case we study the implementation of a product agent that has not been used to create the product itself, but this agent is created for a specific phase in the life cycle of the product. The case study for our product agent in the use phase is based on a discovery robot. This robot is also introduced and globally described. After this description of the product, the embedding of the product agent is discussed and some results of the implementation of the product agent in this complex system are shown.

The next case study describes an application of the product agent in the recycle phase of a product. This is also related to the Internet of Things paradigm and helps to recycle useful material and subsystems. An agent that knows the sequence of steps that were used to build a product can be helpful in the situation a product must be taken apart. Exchanging parts between products is also a topic of the case study.

The third case study is dedicated to agents applied in an infrastructure for home automation. The case has been added to study the situation where a product agent not only has monitoring as a role. In this case product agents have a more prominent role because they are the vital parts of the system described. The product agent will make the system work as a whole based on interagent communication and sharing information. The components used in this case could be manufactured in a grid-based manufacturing system as described in the previous chapters, however, in the case presented here this is not yet realised. All cases discuss related work and end with a conclusion.

5.2 Agents supporting products

This section will focus on the possibilities for agents to support products. First an overview of the roles in different phases of the life cycle is given. Then a discussion of the Internet of Things (IoT) will be presented because this concept plays an important role in this chapter. Next, a review is given for some possibilities for embedding agents in products.

5.2.1 Role of agents in the life cycle of a product

In Figure 5.1 the life cycle of an arbitrary product is shown. After the design, the product is manufactured in the production phase, next the product is distributed. A very important phase is the use of the product and finally the product should be recycled. In all these phases, the product agent can play a role that will be globally described in the next sections. We remark

Design
Manufacturing
Distribution
Use
Recycling

Figure 5.1: Life cycle of a product

that in the literature the term product life cycle (PLC) is commonly used to denote a concept that is different from what we call the life cycle of a product. The product life cycle can be defined as the process wherein a product is introduced to a market, grows in popularity, and is then removed as demand drops gradually to zero (Lilien et al., 2003).

Design and Production

As stated earlier, the design of a product will be greatly influenced by the individual end-user requirements. Cost-effective small scale manufacturing will become more and more important. The manufacturing system based on a grid of cheap and versatile production units called equiplets is already described in the previous chapters. Important is the fact that the product agent is responsible for the manufacturing of the product as well as for collecting relevant production information of this product. This concept is the basis for the roles of the product agent in later phases of the life cycle. The product agent carries the product design as well as the production data and can be viewed as the software entity that represents the product possibly in cyber space.

Distribution

Product agents can negotiate with logistic systems to reach their final destination. Logistic applications based on multiagent systems already exist (Burmeister et al., 1997). Information of product handling and external conditions, like temperature, shocks etcetera can be measured by cheap wireless sensors and collected by the product agent in its role of guidance agent during the transport or after arrival at the destination. The handling and external conditions during transport can be important during product use, especially for product quality, maintenance and repair.

Use

The role of the product agent during the use of the product could focus on several topics. The first question one should ask is: who will benefit from these agents, that is who are the stakeholders. In a win-win situation both the end-user as well as the manufacturer could benefit from the information. If a product is a potential hazard (in case of misuse) for the environment, the environment could also be a winner if the agent is capable of minimizing the effects of misuse or even prevent it.

In the next subsections several topics of usage of the product agent are proposed.

Collecting information A product agent can log information about the use of the product as well as the use of the subsystems of the product. Testing the health of the product and its subsystems can also be done by the agent. These actions should be transparent for the end-user. If a product needs resources like fuel or electric power, the agent can advise about this. An agent can suggest a product to wait for operation until the cost of electric power is low i.e. during the night. It depends of course on the type of device if this should be implemented.

Maintenance and repair Based on the logging information about the product use and the use of the subsystems, an agent can suggest maintenance and repair or replacement of parts. Repairing a product is easier if information about its construction is available. Also the use of a product or the information about transport circumstances during distribution can give a clue for repair. An agent can also identify a broken or malfunctioning part or subsystem. This could be achieved by continuous monitoring, monitoring at certain intervals or a power-on self test (POST) (Arbaugh et al., 1997).

An important aspect of complex modern products is the issue of updates or callbacks in case of a lately discovered manufacturing problem or flaw. In the worst situation, a product should be revised at a service center or the manufacturing site. Information about updates or callbacks can be sent to the product agent that can alert the end-user in case it discovers that it fits the callback or update criteria. This is a better solution for a callback than globally advertising the problem and alert all users of a certain product when only a subgroup is involved.

Miscellaneous Use of product agents could result in transparency of the status of a product after maintenance by a third party. The agent can report to the end-user what happened during repair so there is a possibility to check claimed repairs. Of course the agent should be isolated from the system during repair to prevent tampering with it. Recovery, tracking and tracing in case of theft or loss are also possible by using this technique. When the end-user wants to replace a certain device by a new one, the product agent can give advice about the properties the replacing device should have, based on what the product agent has learned during the use phase.

Recycling

Complex products will have a lot of working subsystems at the moment the end-user decides it has come to the end of its life cycle. This is normally the case when a certain part or subsystem is broken. The other remaining parts or subsystems of the product are still functional, because in a lot of complex products the mean times between failure (MTBF) (Gnedenko et al., 1999) of the subsystems are quite different. The product agent is aware of these subsystems or components and depending on the economical value and the remaining expected lifetime these components can be reused. This could be an important aspect of 'green manufacturing'. An important issue here is that designers should also take in account the phase of destruction or recycling. Disassembly and reuse of subsystems should be a feature of a product for this approach to be successful.

The product agent can reveal where rare or expensive material is situated in the product so this material can be recovered and recycled. This way the product agent can contribute to the concept of zero waste. *Zero waste is just what it sounds like - producing, consuming, and recycling products without throwing anything away* (Gunther, 2007).

Another advance of having a product agent at hand in case of recycling is the fact that the product agent has the information how a product is constructed. This is helpful when a product must be taken apart. For certain steps a kind of undo-steps should be carried out to dismantle a product.

5.2.2 Internet Of Things

Being the software entity representing the product, the product agent can play an important role in the concept of the Internet of Things (IoT). In this section, IoT will be discussed to clarify why the product agent fits in this model. A possible very short definition of the Internet of Things is: *a network of communicating objects*. This means that it is not necessary to have the devices connected to the Internet. Sensor networks (wireless or wired) using protocols like ZigBee, 802.15.4 (Baronti et al., 2007) and WirelessHART (Song et al., 2008) fit in this concept. The connection to the Internet as we know it, is in these cases mostly done by a gateway or another intermediate device.

Requirements

Each device should have a way to communicate. For this communication to take place identification of devices that it needs to communicate with is necessary. Of course the devices should communicate with compatible protocols. Depending on the device the communication at the physical layer could be wired but mobile devices should use wireless communication technology.

Another requirement is the capability for computing. In some cases computationally complex tasks should be done. It is possible to alleviate the local computer by using remote powerful systems to do these complex tasks. For some devices real-time operation, including real-time computing and real-time communication is required, putting requirements on the local computing power and speed.

To make computing possible, the devices will need power. Some devices could be powered by the net or by the wired communication system (Power over Ethernet), but mobile devices are mostly powered by batteries that need to be charged at regular intervals. For some application solar cells could be used, but most mobile devices should be connected at regular intervals to a charger system to reload the battery. In some cases it may be possible to reload batteries by so called power harvesting methods. A medical device that is attached to a human can get energy from movements of the body or a low power device can harvest energy from its surrounding electromagnetic field generated by transmitters or special electrical equipment. The issue of power is very important when devices are used for critical applications. In the situation of critical applications other issues are also important like:

- How to handle the failure of some components?
- How to handle loss of communication?
- How to ensure secure operation, including secure storage of information on each device?

If devices are introduced to work in an environment where other not so smart devices are operating, the smart devices belonging to the IoT should deal with legacy devices in a proper way. For example, in a smart traffic system a smart car that is designed to cooperate with other smart cars should also handle and coexist with legacy cars that are not equipped with IoT components. A smart car should also operate on roads that are not equipped with appropriate IoT assisting infrastructures.

To implement an IoT system one should think of a distributed framework that can be used to build dependable, optimal and useful IoT systems. Such a framework should provide administrative and control services, various types of useful resources, and device-based capabilities. This includes:

- Resources to support communication, computing and storage requirements of each application.
- Capabilities to minimize power usage on basic devices and also to recharge various devices.
- Services to keep track of the position of each device as needed.
- Mechanisms to achieve high availability, reliability, security, and other dependability requirements for each application.

Classes of IoT Applications and systems

(Bastani, 2013) proposes 5 different classes of IoT systems.

1. Fully static systems
   - All the devices in the system are pre-configured to communicate and interact with each other.
   - The devices can interact via wired or wireless communication networks.
   - The application code on each device is pre-coded to interact with the other devices that it needs to interact with. This also specifies the type of information (sensor readings, actuator commands, etc.) that are sent by each device.

- Application examples:
  - Body area sensor networks.
  - Some industrial sensor networks like WirelessHART (Song et al., 2008).
  - Various technologies for monitoring patients and alerting doctors in case of emergencies.

2. Centrally administered systems with a single administrative unit.
   - The devices are pre-configured to interact with a central unit - This is the registration step.
   - Each device can query the central unit to identify other devices that it needs to interact with.
   - Devices can communicate with each other directly or they can communicate via the central unit.
   - Devices can be fixed units as well as mobile units.
   - Application examples:
     - Smart homes with capabilities for coordinating power usage, adapting to environmental conditions, ensuring security of the home, etcetera.
3. Centrally administered systems with multiple administrative units where the administrative units are centrally coordinated.
   - These systems are typically widely distributed, hence several administrative units are used at different geographical locations.
   - Devices are pre-configured to locate/interact with nearby administrative units.
   - The administrative units keep track of all the devices and ensure that each device is aware of the other devices that it may need to interact with.
   - Devices can inter-communicate directly or via the administrative units.
   - Devices can be fixed units as well as mobile units.
   - Application examples:
     - Smart vehicle systems (AGVs) that can coordinate with other vehicles to prevent accidents.
       - ∗ Roadside units are deployed to keep track of each vehicle

        * These units provide various services, including communication, safety alerts, etc.
    - Infrastructure monitoring systems.
    - Agriculture systems, etc.

4. Centrally administered systems with multiple administrative units where the administrative units form a decentralized system.

    - The administrative units are mobile systems.
    - These systems cooperate with each other to identify and track all the devices.
    - Devices are made aware of the other devices they may need to interact with.
    - Devices can communicate with each other directly or they can communicate via the administrative units.
    - Devices are typically mobile units.
    - Application examples:
        - Smart business systems that keep track of inventories, etc.
            * Systems that keep track of the quantity and quality of each unit.
            * Systems that automate customer checkouts.
            * Systems that automate supply chain management processes.
        - Livestock and pet monitoring systems.
        - Keeping track of children and old people to ensure their safety.

5. Fully autonomous decentralized systems:

    - Each device is an independent unit that has capabilities to identify other units and coordinate with them.
    - Devices communicate with each other directly.
    - Devices can be fixed or mobile units - The autonomy is used to minimize the interaction time while ensuring requisite coordination and cooperation with other units.
    - Application examples:
        - Avionic systems: Each aircraft can autonomously identify nearby aircraft and coordinate its motion in order to prevent accidents.

  - Ocean systems: Each vessel can autonomously identify nearby ships and coordinate its motion in order to prevent accidents.
  - Wind farm systems: Each turbine can directly interact with neighbouring turbines to achieve optimal overall harvesting of wind energy based on the current conditions while ensuring that there will not be any damages to any of the turbines.

IoT and agent technology

IoT has a great potential to significantly enhance the quality of life, society and the environment. Open standards should be used to make the concept widely accepted and interoperable. For communication purposes, standard TCP/IP can be used. Especially the use of IPv6 is obligatory, because of the huge number of IP addresses needed for the implementation of this technology. By using TCP/IP a device becomes a part of the internet as we know it and is not using its own possibly proprietary or obscure communication protocol hidden behind a gateway. Though interoperability is achieved by adhering to TCP/IP, the security become an issue. When all devices are accessible, abuse and tampering by hostile users or even hostile devices becomes a possibility that can have a severe impact on the adoption of the concept of IoT.

Agent technology can play an important role in de realisation of the IoT, especially in the situation where multiple devices will act as a multiagent system. Agents are the software representatives of the devices. There are several ways to tie an agent to a device as shall be discussed in the next section.

5.2.3 Product types

This approach of having an agent for a product could be used on different kind of products, but one should investigate if the final product has intelligence and hardware to communicate with the agent. Some products have this by nature (computers, cell-phones); for other products (cars, machinery, domestic appliances) it should be a small investment. An important aspect will be the possibility to connect to certain subsystems for monitoring important events. If temperature is an important item for the product agent, connection to a temperature sensor or at least a place where this temperature data is available is a must. If this connection is not available, a temperature measurement system should be added to the agent.

Where do these agents reside?

A product agent should stay alive or at least the information the agent has collected and the knowledge the agent has learned should be available under all circumstances. To accomplish this, two solutions are available. The agent can be a mobile agent moving from platform x to platform y as depicted in Figure 5.2a. The other solution requires moving data (beliefs of the agent) from one agent to a newly created agent as shown in Figure 5.2b. In our case both agents should be product agents. The second solution is much easier to

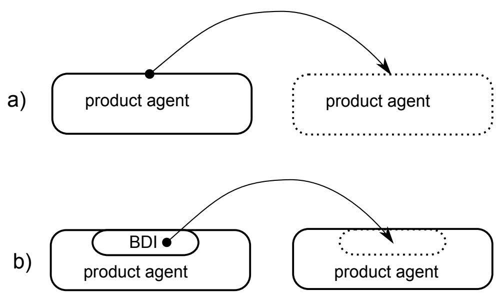


Figure 5.2: Mobile agent versus moving data

implement because of the fact that only transport of data is required, while in the case of moving agents, the whole executable should be adapted to the new situation. Another advantage of the second approach is that a product agent can be added in any phase of the life cycle. This is also what has been done for the specific research presented in the first case. A product agent was added to a system in the use phase. The biggest challenge for implementing the approach of a product agent or guidance agent will be in the use phase. This is where the product is under control of the end-user and not as during the production under control of the manufacturer. In the latter case an agent-based infrastructure can be implemented for the production system or production line. The same is true for transport and even disassembly of the product. In case of the use phase, the agents should reside in a system that is connected to the product, but should be available at the moment the product itself is broken. This is comparable to the case of the black box in aeroplanes. There are several possibilities, depending on the type of product:

- The agent runs on its own separate hardware that is closely tied to the product.
- The agent runs on the hardware of the product but stores information on a special place on the product itself. This information can be recovered after breakdown.

- The agent runs on the hardware of the product but stores information on a remote system.
- The agent runs on a remote system that has a continuous connection.
- The agent runs remote on a system using a 'connect when necessary' approach.

The last two options require a stub or entry point for the remote agent to make contact with the product system. The connection with the environment could be established by wired or wireless sensors or sensor networks as well as computer subsystems in the product. Interaction with humans in the environment could be established by a messaging system or human computer interface (HCI). This interface could be implemented by standard web technology. The advent of HTML-5 can help to develop powerful web interfaces.

Embedded agent-based realization

There are a lot of possibilities for embedding the product agent in its role during the life-cycle of the product. In this section, an overview of possible realisations will be given. This overview is far from complete, but gives an impression of how the possibilities of the agent can increase in the situation of higher complexity. In Figure 5.3 the inclusion of an RFID-chip (Finkenzeller, 1999) in a device is depicted. This will lead to a situation where the product can be tracked by RFID-scanners. RFID implementations are becoming more powerful but the main use is product tracking and not yet embedding smart agents. For RFIP no battery is required, because the RFID-chip is powered by the reader hardware. In Figure 5.4 a more powerful solution

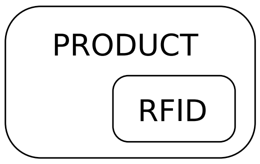


Figure 5.3: Remote-monitored product

is presented. Here the product agent can live inside the product, using the available processor and storage capacity. In this situation the agent is still lacking continuous communication or communication initiated by the agent. The system has to wait until a connection is made to the outside world. To keep the agent alive, a power source like a battery is needed. The communi-

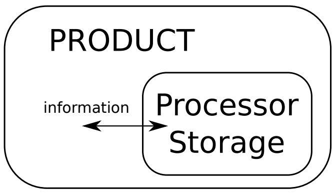


Figure 5.4: Product with local agent environment

cation capability is added in Figure 5.5. Here the agent can communicate at will with the external world. The communication can be continuous or on an at hoc base. In the final situation, depicted in Figure 5.6 the product agent

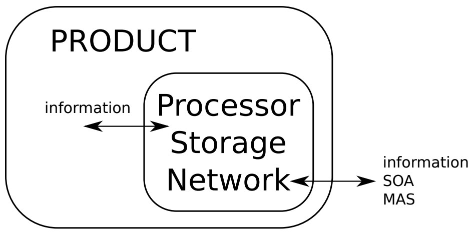


Figure 5.5: Networked agent

can use external services as helpers to add possibilities the on-board system cannot deliver. Storage in the cloud as well as services that can play a role for the agent can be used using the network connectivity. This connectivity can also update the system, in the same way as is used in standard computers. Some heavy weighted applications, like complex reasoning, can also be used as a service. The acronym SOA stands for Service Oriented Architecture and is widely used in distributed business architectures (Papazoglou, 2003).

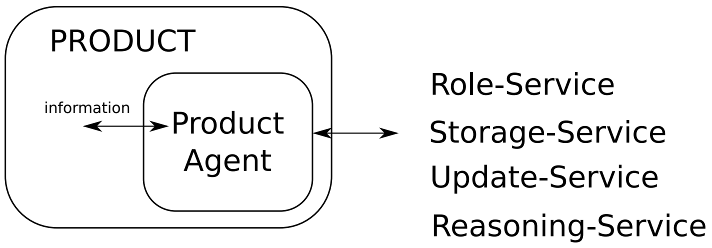


Figure 5.6: SOA-based product agent

5.3 Case study: Discovery robot

This section gives details of the a discovery robot that was built by our research group. To investigate the implementation of the product agent during the use phase, the product agent was embedded in this complex technical system. To understand the details of the product agent implementation, it is important to have a global understanding of the construction and working of the discovery robot. In this section we present a short overview of the robot capabilities, the architecture, the software and an example of a result produced by the robot system.

Robot capabilities

The robot that will be used as a platform for the product agent is capable of mapping a room with objects by using a laser scanner. The robot can move by itself using the map that has been created by the laserscan. It is possible to direct the robot to a certain point in its map. The robot is also capable to avoid newly introduced obstacles and other moving objects. This robot is used as a system that will be enhanced by a product agent.

Architecture

Figure 5.7 shows a picture of the hardware of the robot. Two motors are connected to two wheels. Two swivelling wheels are added to keep the platform in balance. Attached to the platform is the laser scanner, printed circuit boards, a WiFi transceiver, a camera and a set of ultrasonic sensors placed in a circle at the edges of the platform. These ultrasonic sensors are not yet used at this stage. A block diagram of the robot is depicted in Figure 5.8.

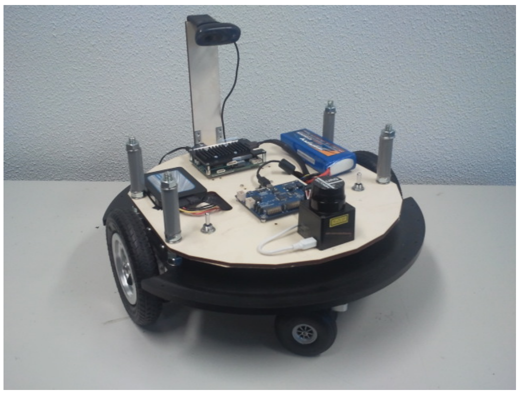


Figure 5.7: Discovery robot

An important aspect is shown in this figure. An external computer is part of the system. This computer is used to do the heavy calculation to generate the map information, to display the map in real-time and to plan the path the robot has to follow. A wireless Ethernet connection (WiFi) connects the robot with this external system.

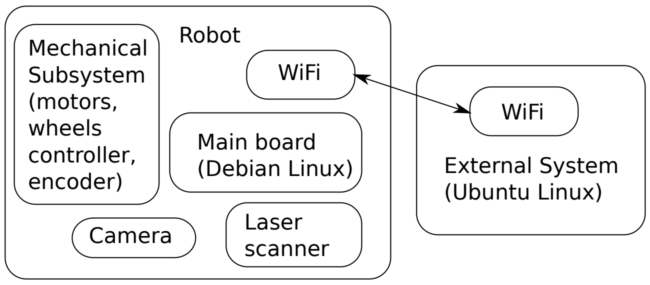

Robot
Mechanical Subsystem (motors, wheels controller, encoder)
WiFi
Main board (Debian Linux)
Camera
Laser scanner
WiFi
External System (Ubuntu Linux)

Figure 5.8: Block diagram of the robot

Software

The software for this robot is based on Linux, ROS and SLAM. Linux is a Unix-like open source operating system that is popular for technical applications, embedded systems, network servers and standard computers.

ROS ROS is an acronym for Robot Operating System (Quigley et al., 2009). ROS is not really an operating system but it is middle-ware specially designed for robot control and it runs on Linux. In ROS a process is called a node. These nodes can communicate by a publish and subscribe mechanism. In ROS this communication mechanism is called a topic. Figure 5.9 shows the relation between two nodes and one topic.

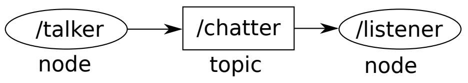


Figure 5.9: Two nodes connected by a topic

A node that produces data can publish this in one or more topics. Other nodes interested in these data can subscribe to one or more topics. TCP/IP is used to actually carry out the communication. This type of communication is asynchronous, meaning that after publishing to a topic, the node that did publish will continue to do others tasks. A synchronous type of communication is also available in ROS. This type of communication is called a service.

A service is called by a node to another specific node and the calling node will wait until the service is completed.

The ROS platform has been chosen for the following reasons:

- Open source, so easy to adapt, compliant with a lot of open source tools.
- Wide support by an active community.
- Huge number of modules already available.
- Nodes that are parts of ROS can live on several different platforms, assumed that a TCP/IP connection is available.

SLAM The mapping is done using SLAM. SLAM stands for Simultaneous Localisation And Mapping (Durrant-Whyte and Bailey, 2006). This module was already available in ROS and fitted well to the on board laser scanner. Slam is more than just a mapping system. It can also be used in mobile robot system as its name suggests for localisation. The idea is that a robot will be positioned in an unknown area at an unknown position. From there the robot will start to build step by step a model of the environment and its position within that environment. To build the map of the environment, the robot needs information about the environment. This information can be achieved by using sensors, like cameras or as in our model laser rangefinders. When a sensor is capable to detect distances and/or depths it will be a good feature to implement and combine it with slam. In our system the robot will start at position (0,0) and from there it will build the map. this will only be a partial map at first, but when the robot starts to move it will use information from the wheel encoders to keep track of the path, the robot has followed. This technique is named odometrics. New information will be added to the map, because the robot scans its environment from different positions. Meanwhile the location of the robot within the map will also be calculated.

Results

The results of a mapping in progress are displayed in Figure 5.10. Here the robot mapped the corridors in a rather big building with three wings. The corridors are plotted as a light grey shape. The length of the longest corridor in this map is about 50 meters. At this stage, the robot is not yet autonomous, but is controlled by a human operator that uses the external system and the on-board camera to guide the robot during the mapping.

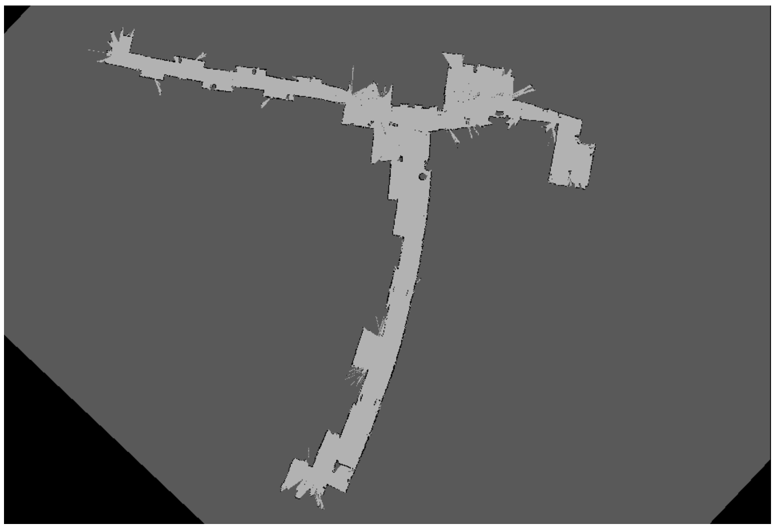


Figure 5.10: 2D mapping of a building

When the map is completed, the robot is capable to navigate autonomously to a given point in the map, even if new or moving obstacles are introduced in the mapped environment. Now the robot system has been introduced, the focus in the next sections will be on the embedded product agent.

5.3.1 Embedded product agent

This section describes the product agent and also shows some results of its functioning.

Functional requirements

The product agent that is added to the robot has the following requirements:

- Monitoring status of the system or subsystems.
- Monitoring health of the system or subsystems. The difference between health and status will be explained in the next subsection.
- React only in case of emergency.
- The robot should operate without the agent.
- Making useful data available to the outside world, like construction details, materials used and its localisation in the robot.

Implementation

The first step in implementing this product agent in the robot is to make an overview of information available in the system. Different types of information are considered:

1. Status: is data available and of interest to the product agent and/or the end-user.
2. Health: has to do with the condition of components that have mechanical parts or deteriorate during use.
3. Alarm: an internal condition that could result in a troublesome situation or disaster.
4. Additional information: this is the information that was conceived in earlier phases of the life-cycle.

Because the ROS environment is already available, it seems a natural choice to use this environment to implement the agent. The agent consists of ROS-nodes, ROS topics and some other subsystems. In Figure 5.11 the internal modules of the agent are shown. All parts surrounded by an ellipse are ROS nodes. The rectangles represent topics. For human interfacing a small web server is included. This server is capable to serve static pages, containing technical data about the robot as well as dynamic pages containing data collected during use. Figure 5.11 shows the the internal parts of the product

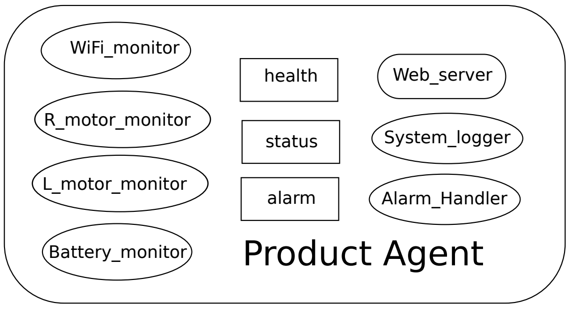


Figure 5.11: Architecture of the product agent

agent and Figure 5.12 shows the product agent in its the environment. The product agent interacts with its environment. The agent gets its information from the robot and its operating system. The agent will log this information and can also display information on a web browser (web client) by using the aforementioned webserver. A shutdown can be performed in case of a certain alarm condition.

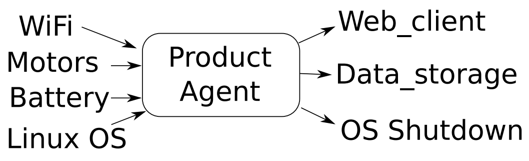


Figure 5.12: The product agent and its environment

Monitoring status

The monitoring function is an important aspect of the product agent. In our prototype a selection of possibilities was made. A node will monitor the use of the motors and this will be available to subscribers of the health topic. The status topic is comparable to the health topic, but here information is made available that is not a result of the wear and tear of for example mechanical parts or of the de-charging of the battery, but is a result of measurements of interesting data like the strength of the WiFi signal. There is one topic that can trigger a node that will issue a system shut-down. This topic is called the alarm topic. Apart from these nodes, the agent can also retrieve its information directly from the Linux environment. Commands are available to get the CPU-load and memory usage. The pseudo filesystem `/proc` offers also a wealth of technical information that can be useful for the product agent.

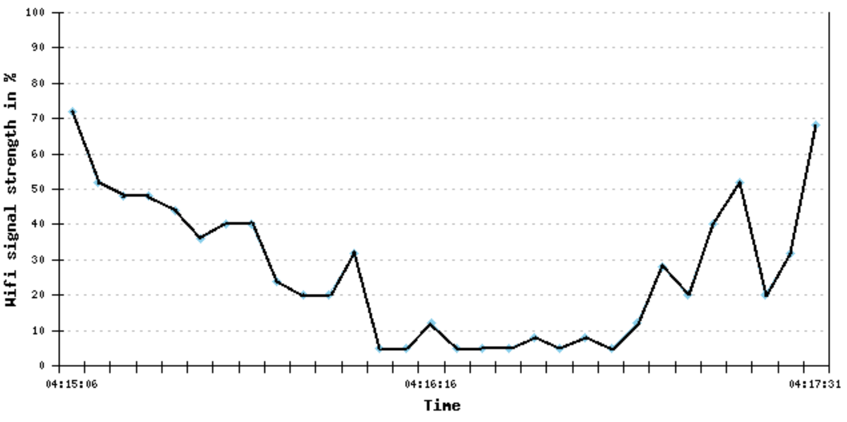


Figure 5.13: Strength of the WiFi signal

Examples of what can be retrieved from the product agent are plots displayed in the following two figures. Figure 5.13 shows a picture of the strength of the WiFi signal. The robot first moved away from the WiFi access point and then returned towards the WiFi access point again. The plotted data show a global decreasing and again an increasing trend but

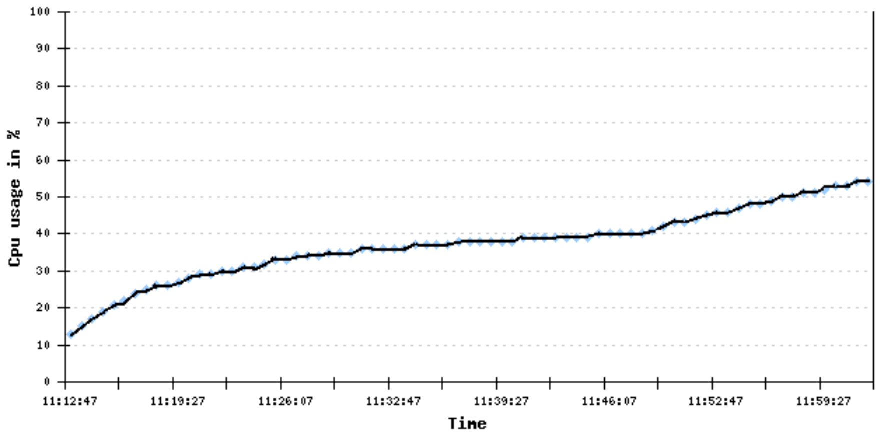


Figure 5.14: CPU load

there are also strong fluctuations. These fluctuations are normal and due to all kinds of reflections and interference that occur in an indoor environment. In Figure 5.14 the load of the processor of the onboard Linux-based controller board is plotted. This curve is quite smooth and shows that the available processor power is adequate to operate the robot platform.

Monitoring health

In the robot there are two candidates for monitoring the health. The motors and the battery. The battery should be monitored because of the fact that, like almost all rechargeable batteries, it can be recharged and decharged a finite number of times and information of its remaining charge is valuable information to the end-user that operates the robot. In Figure 5.15 the status of the battery is plotted during 90 minutes of operation of the robot. A steady decrease is shown as might be expected.

Alarm conditions

In this section an alarm condition will be described. The fact that the type of battery that is used in the robot should never be completely discharged gives rise to such an alarm condition. When the charge capacity drops below 10% a system shut-down action should be triggered. By shutting down the system, the discharge of the battery will stop, thus preventing the loss of a rather expensive component. To implement this feature an Analog to Digital Converter (ADC) should be available to check the status of the battery.

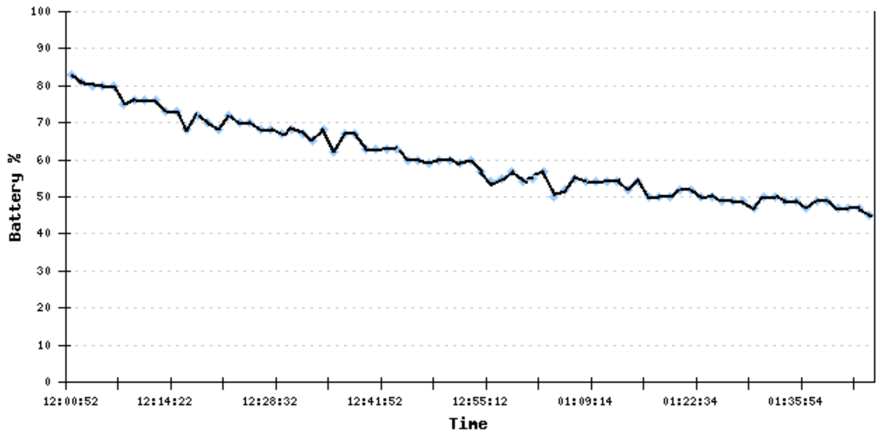


Figure 5.15: Charge status of the battery

Extra functionality

The extra functionality that is offered by our implementation is embedded documentation and a mapping of materials and components that are of interest during the recycle phase. The information is offered using the same web-interface as was used in the monitor section previously discussed. The documentation is comparable to printed documentation that could be bundled with any device. This includes a user manual, a technical manual and a maintenance manual including a trouble shooting section.

In Figure 5.16 a webpage is displayed showing the subsystems of the robot. This allows the user to select a subsystem to get more dertailed information about that specific subsystem. Important information for the recycle phase is also offered using the web interface. Two different approaches are implemented. Using the web interface, one could point at any part of the robot and receive information about the 'ingredients'. An example is given in Figure 5.17. Here the wheels are selected and as a response the information about the material available in these parts is displayed The materials as well as other relevant information is displayed.

Another approach is presented in Figure 5.18. A list of interesting materials is presented and by clicking on an item, the subsystems containing this material are highlighted as shown in Figure 5.18, where the subsystems containing gold are shown. These examples only show the interface designed for human users. The information is also available in a machine readable form using XML.

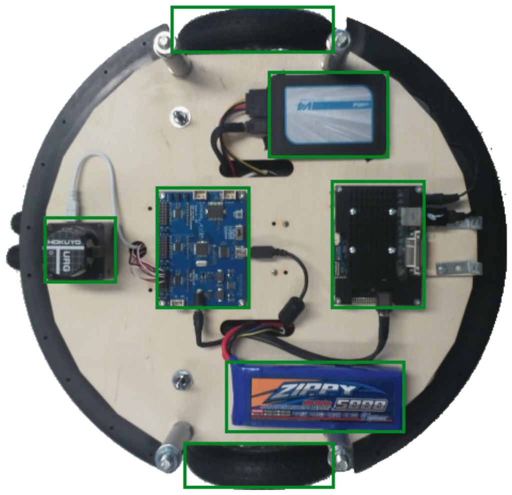
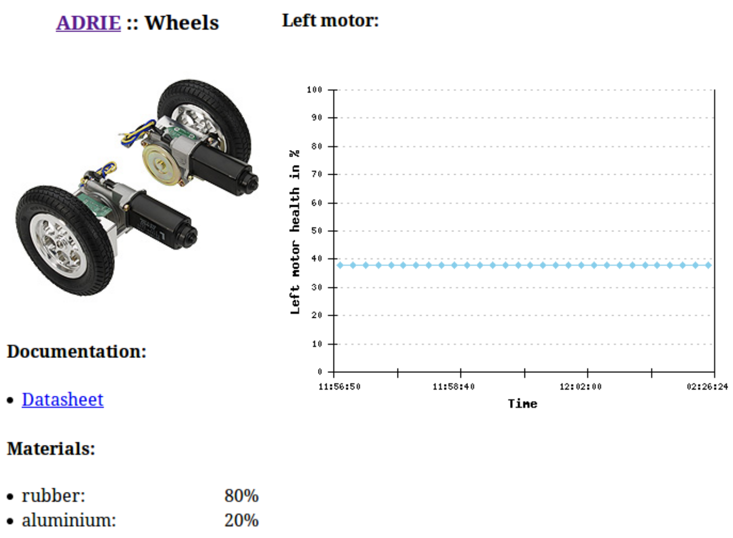


Figure 5.16: Discovery robot



ADRIE :: Wheels

Left motor:

Documentation:

- Datasheet

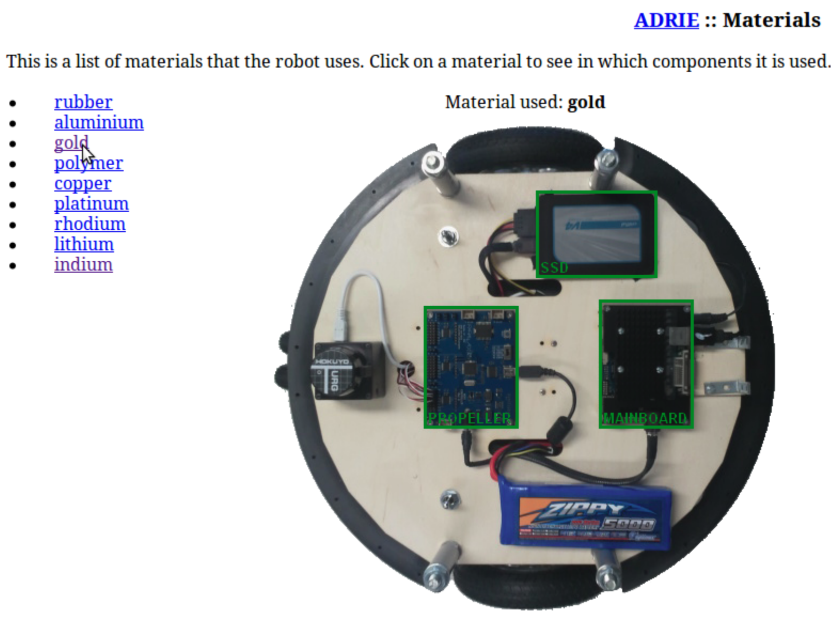
Materials:

• rubber:	80%
• aluminium:	20%

Figure 5.17: Motor and wheel subsystem

5.3.2 Related work

The work on ROS played a very important role in this research. By using ROS we had a stable and well developed platform for our robot. The use of proven modules prevented reinventing solutions to already solved problems. The work on discovery robots is huge, (Wnuk et al., 2006) and (Blazovics et al., 2011) show some developments focussing on multiagent and swarm solutions. Agents for distribution, logistic applications and product manufacturing already exist (Paolucci and Sacile, 2005). In most situations agents



ADRIE :: Materials

This is a list of materials that the robot uses. Click on a material to see in which components it is used.

- rubber
- aluminium
- gold
- polymer
- copper
- platinum
- rhodium
- lithium
- indium

Material used: **gold**

Figure 5.18: Webinterface to show the location of materials

represent human operators or negotiators. Jennings and Bussmann introduce the concept of workpiece agents that operate during the production. These agents do not however perform individual product logging. The use of a product is also studied by observing and/or interviewing end-users (Nielsen and Levy, 1994) (Nielsen and Mack, 1994). Some software applications do connect with their originating company to report the use by end-users.

Several proposals and implementations of including monitoring and documentation within the product itself are made and implemented. Burgess (Burgess, 1998) (Burgess et al., 2002) describes Cfengine that uses agent technology in monitoring computer systems and ICT network infrastructure. In Cfengine, agents will monitor the status and health of software parts of a complex network infrastructure. These agents are developed and introduced in the use phase of this infrastructure and focus on the condition of the software subsystems. In our approach this monitoring function for hardware and software is the role of the product agent but that role has been played already by an agent during the manufacturing phase where valuable information that can be useful to the end-user has been collected. Actually this product agent in the use phase is not necessarily the same software entity that played the role of product agent during production, but the belief base of the product agent is kept intact and handed over to a new incarnation of the product agent.

In (Hamilton et al., 2007) an integrated diagnostic architecture for autonomous underwater vehicles is described. In this work the focus is on an intelligent system for system diagnostics. The architecture uses a variety of domain dependent diagnostic tools (rulebase, model-based methods) and domain independent tools (correlator, topology analyzer, watcher) to first detect and then diagnose the location of faults. This work could be used and combined with the model present in the current chapter, because the artificial intelligence-based techniques can applied in the product agent. Our work expands the idea of diagnosis and related data to the whole life cycle of a product. By using this same agent again in the final phase of the life-cycle, component reuse and smart disassembly is a very important aspect when it comes to recycling of rare or expensive building material. The status of the quality of used sub-parts is available from and presented by the product agent.

In (Ashton, 2009) the concept of the 'Internet of Things' is explained by the first user of the term 'Internet of Things'. The main idea of this concept is that the content of Internet is not only built and used by humans and therefore largely depending on humans, but the content will also be built by things connected to the Internet that are programmed to do so. The work presented in the current case in this thesis shows a possible technique to implement this concept of the 'Internet of Things'.

5.3.3 Discussion

In this case the focus was on implementing a product agent in a complex product. This product was a discovery robot but could have been any other technical system. For every system the requirements for a product agent should be specified. However some global specifications are applicable for every system. The choices for monitoring subsystems made in this research were limited only to a few due to the fact that a proof of concept was the goal of this research. The actions the agent can perform are in this case displaying and storing system status and system health status as well as system design and technical data. The agent is not influencing the robot itself. However, one alarm condition is implemented resulting in a system shut-down. It is not a difficult task to expand the capabilities of the agent. The robot itself will be further developed. This robot technology could be the basis of the autonomous flexible transport of products between equiplets as described in Chapter 4. For the product agent a wide variety of future enhancements is possible, especially when product agents of a certain type of product are united in a multiagent system:

- A model that builds a failure overview of subsystems. This way an accurate insight in the reliability of subsystems and components can be obtained. This model only works if a huge amount of product agents are participating.

- On behalf of the end-user, a product agent can report component failure and suggest or order replacement parts.
- An interesting model to implement the previous feature could be a marketplace in cyberspace where product agents can negotiate with other product agents about exchanging parts. This will be discussed in the next case of this chapter.

In all these enhancements special attention should be paid to security and the protection of privacy of the end-users of product agent enhanced systems. An important aspect is the fact that the agent should store its information at a safe place in case the robot hardware will fail. In our case this is the remote system where the agent has the possibility to store important data.

5.3.4 Conclusion

Product agents can play an important role in every part of the life cycle of a product. An important property of these agents might be that they should have no direct impact on the product or system they are living in. However useful information should be collected and in case of disaster, these agents should keep a log of the events leading to the disaster.

Product agents can be a virtual digital equivalent of a product and this concept will be an enabling technology in implementing the internet of things.

The concept presented here is a natural evolution of the concept of using agents during production. However in case of products made by production technology not based on agent technology, a product agent can be added afterwards, as described in this case study. The information that could have been collected during design and production is added afterwards and will play a role in the recycle phase or maintenance during use phase.

The ROS platform proved to be a very good platform to implement the product agent. This is because of the fact that the data-communication infrastructure between nodes is already implemented in a way that helps a lot in both the design and the implementation of the product agent.

5.4 Case: Resource depletion

A problem humanity encounters is the depletion of natural resources. This can be seen by the sometimes enormous increase of price of some of these

Table 5.1: Depletion of elements

Name	Symbol	Years available
Silver	Ag	29
Indium	In	13
Antimony	Sb	30
Hafnium	Hf	10
Tantalum	Ta	116

resources. The price of lead, gold and copper increased by resp. 378 %, 308 % and 269 % from 1999 to 2009 (Bloomberg, 2009). In some cases this has to do with the use of elements that are hard to find. In other cases the demand for elements has increased because of certain newly developed applications or an increased field of applications for that element. Apart from searches for new places to mine these elements, another - for the environment perhaps better way - to come around this problem is to reuse material. Today cellphones containing these rare elements are considered a new kind of ore. To reuse the elements it would be nice if it could be located within the device. This is where the product agents come in handy. Table 5.1 shows some of these rare elements and the expected time left before recycling is the only way to get these elements (Cohen, 2007). The expected time left is based on the use as it is. If the use of a certain element increases, the time left will be shorter of course. Product agents will help us hunting for places where rare metals are concentrated enough to be worth recovering. This is because the product agent carries all the information that has been collected during the production phase. To make this concept work, a list of “ingredients” should be part of this information in the same way (or even better) as is the case in the food industry.

5.4.1 Recycling of subsystems

In this section a Monte Carlo simulation is used to show the effect of recycling subsystems of broken products. Also a marketplace model is presented that has been developed to show a possible implementation of agent-based recycling.

Extending the average lifetime

An interesting application for the product agent can be automatic recycling of subsystems during its use.

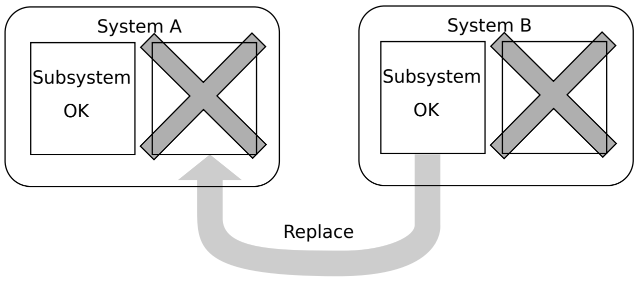


Figure 5.19: Two equal systems, both consisting of two equal subsystems

To explain this in more detail, consider a product consisting of 2 equal subsystems as depicted in Figure 5.19. This means that these subsystems have the same average lifetime and are also similar. In this section we focus only on products where two or more subsystems are all of equal type. We assume that the time before a failure occurs is a normal distribution according to formula:

$$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Where μ is the average lifetime, σ the standard deviation and σ^2 the variance. In (Gnedenko and Ushakov, 1995) is stated that the time of failure of components subject to mechanical wear is a normal distribution. An end user considers the product to be broken if one of these subsystems is broken. Normally one subsystem will be the first to fail leaving another still functioning subsystem in the broken product. To get some insight in this situation, a Monte Carlo simulation was set up. This simulation was based on 1000 products, starting with two equal subsystems. For every product we generated two failure times according to the given normal distribution. The plot of all these failure times (see Figure 5.20) turned out to be a rough approximation of the theoretical Gaussian curve.

If we plot the minimum and maximum failure time of the two equal systems per product, this results in two smaller curves as depicted in Figure 5.21.

If there is no exchange of subsystems among the products, half of the products will be broken on the average time of the curve belonging to the first subsystem to break. Taking the average of the left curve in Figure 5.21, this

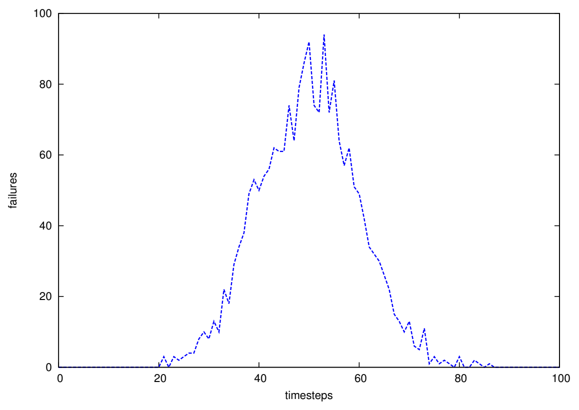


Figure 5.20: Distribution of failing subsystems

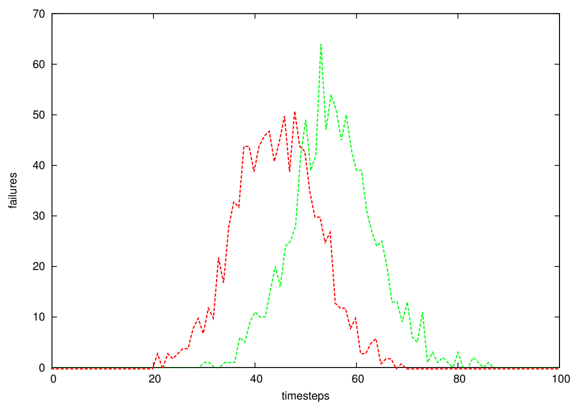


Figure 5.21: Distribution of two failing subsystems

turns out to be at time $t = 432$. If exchange of subsystems is possible, half of the products will still work at $t = 500$. The gain in lifetime will be: $\frac{500-432}{432}$ and could also be expressed as a percentage. The gain for this situation will be bigger if there are more equal subsystems in the product. In Figure 5.22 the gain in average lifetime is plotted as a function of the number n of equal subsystems. A value of 100 for σ and 500 for μ is used. Starting with $N = 1$ there is no gain at all, because on the average 50% of the products will be broken at time $\mu = 500$.

Another observation is that larger values of σ could also result in a bigger gain in lifetime if exchange of subsystems between products is possible. In this plot we assumed 8 subsystems.

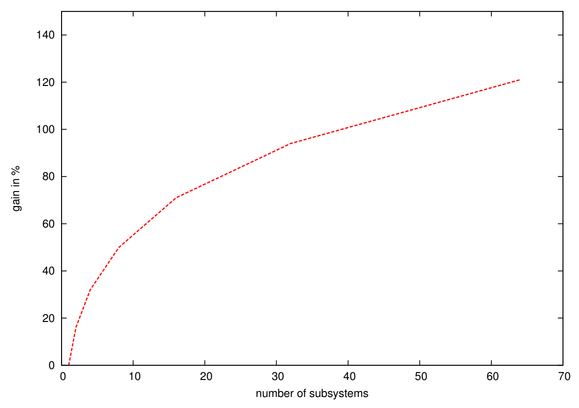


Figure 5.22: Gain as a function of the number of subsystems

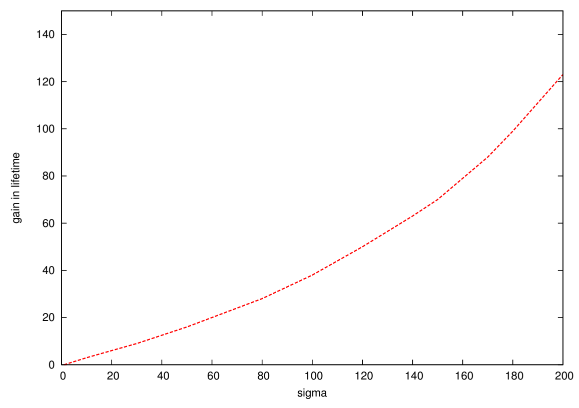


Figure 5.23: Gain as function of σ for $n = 8$

In real practice the situation is a bit more complex. Let us consider some situations:

- If these N products contain subsystems with large value for σ, meaning some parts will live long and other parts fail very quickly, the donor, acceptor approach will be very useful as shown in Figure 5.23.

- Another observation that is easy to understand is: if these N products contain one subsystem that will almost always be the first one to fail, the donor and acceptor approach will not help that much. This type of easily failing subsystems should be in stock as spare parts.

To make this system work for distributed products, a way of communication and exchange appointments should be provided. In the next section a marketplace model is discussed.

Negotiating between product agents

For the implementation of this exchange of parts concept, a Jade multiagent-based system has been developed. The reason for using Jade are the low threshold to develop a multiagent system especially for Java programmers. The FIPA-based communication is already available in Jade and connection to the outside world is easy. Agents can migrate, terminate or new agents can appear. For prototyping a negotiating system as proposed here, Jade is a powerful platform. The product agent residing in a broken system can send the following information to a webserver:

- Information about the status of the machine. What are the broken parts and what parts are still functioning.
- If available: information from the end-user stating if the end-user prefers to be a donor, an acceptor or does not care.
- The e-mail address of the end-user to contact the end-user to confirm the negotiation outcome.
- The maximum time to wait for a successful negotiation.

The webserver contains a Java servlet. After receiving the information from the remote product agent, this servlet will spawn a Jade agent equipped with the information from the original source in a Jade container residing in the computer system that runs the webserver. This Jade agent will take over the role of the product agent. The Jade product agent of a broken product puts its ID and status on a blackboard. Using the same blackboard it will look for a situation where a complete product can be made. This will result in a set of product agents available for exchange. Now comes the question:

To be or not to be a donor The product agent should decide to be a donor or an acceptor. This decision will also be influenced by the owner of the product. The following rules were applied to decide what the product agent should do:

- In the first place, the end-user of the product should decide what to do. If this user does not care, the following rules apply.
- The product with the largest amount of working subsystems should be the acceptor. If this does not result in a decision, use the next rule:
- If a part is broken with a large value of μ (expected lifetime is long) the role of acceptor is a good one, because of the fact that there is a big chance that there exists a donor with this part available.

- If this still does not lead to a decision, a random choice is made.

When a match between two product agents is found, these agents both contact the end-user for confirmation. If both confirmations are positive, the negotiation is considered successful.

Simple implementation

Just to give an impression of how the exchange of subparts between agents could work, consider an induction cooktop with the following subparts (these parts are chosen according to a real device):

- Two small heating elements.
- Two large heating elements.
- Two identical control systems each controlling a small and a large heating element.
- Glass plate (cover).

A Jade-based marketplace has been implemented, just to show how the proposed system might work. At the start all systems are in good condition. At a random time each system has a failing part. The failing part is also randomly chosen. If there is a failure, the product agent in de Jade-based marketplace will search for a replacement part. If there is no replacement part available, the working parts are offered for sale. After some time parts are offered at the marketplace and other agents representing failed systems are capable to bid for parts. The behaviour of the multiagent system is output to a console window. Part of it looks like:

```
agent-2 lost 1 of part CONTROL_UNIT
agent-2 BuyBehaviour quering for CONTROL_UNIT
agent-5 lost 1 of part CONTROL_UNIT
agent-5 BuyBehaviour quering for CONTROL_UNIT
agent-7 lost 1 of part SMALL_HEATING_ELEMENT
agent-4 lost 1 of part CONTROL_UNIT
agent-4 lost 1 of part LARGE_HEATING_ELEMENT
agent-7 BuyBehaviour quering for SMALL_HEATING_ELEMENT
agent-4 BuyBehaviour quering for CONTROL_UNIT
agent-2 BuyBehaviour no hits, selling furnace
agent-5 BuyBehaviour no hits, selling furnace
agent-7 BuyBehaviour no hits, selling furnace
```

```
agent-4 BuyBehaviour no hits, selling furnace
agent-11 lost 1 of part CONTROL_UNIT
agent-11 lost 1 of part SMALL_HEATING_ELEMENT
agent-15 lost 1 of part CONTROL_UNIT
agent-15 lost 1 of part LARGE_HEATING_ELEMENT
agent-10 lost 1 of part CONTROL_UNIT
agent-12 lost 1 of part GLASS_COVER
agent-12 lost 1 of part SMALL_HEATING_ELEMENT
agent-0 lost 1 of part LARGE_HEATING_ELEMENT
agent-0 BuyBehaviour quering for LARGE_HEATING_ELEMENT
agent-12 BuyBehaviour quering for SMALL_HEATING_ELEMENT
agent-10 BuyBehaviour quering for CONTROL_UNIT
agent-15 BuyBehaviour quering for CONTROL_UNIT
agent-11 BuyBehaviour quering for CONTROL_UNIT
agent-0 BuyBehaviour no hits, selling furnace
agent-12 BuyBehaviour no hits, selling furnace
agent-15 BuyBehaviour lowestPrice: 30 sending bid
agent-10 BuyBehaviour lowestPrice: 30 sending bid
agent-11 BuyBehaviour lowestPrice: 30 sending bid
agent-2 SellBehaviour got Bid: Bid [partType=CONTROL_UNIT,
        price=31]
agent-2 SellBehaviour sending ACCEPT_PROPOSAL
agent-2 SellBehaviour got Bid: Bid [partType=CONTROL_UNIT,
        price=31]
agent-2 SellBehaviour sending REJECT_PROPOSAL
agent-15 HandleBidResponse got ACCEPT_PROPOSAL
agent-10 HandleBidResponse got REJECT_PROPOSAL
agent-2 SellBehaviour got Bid: Bid [partType=CONTROL_UNIT,
        price=31]
agent-2 SellBehaviour sending REJECT_PROPOSAL
agent-11 HandleBidResponse got REJECT_PROPOSAL
agent-10 BuyBehaviour quering for CONTROL_UNIT
agent-11 BuyBehaviour quering for CONTROL_UNIT
agent-10 BuyBehaviour lowestPrice: 30 sending bid
agent-0 SellBehaviour got Bid: Bid [partType=CONTROL_UNIT,
        price=31]
agent-11 BuyBehaviour lowestPrice: 30 sending bid
agent-0 SellBehaviour sending ACCEPT_PROPOSAL
```

The behaviour can also be shown by using the standard GUI in Jade. Agents can have their own idea about the price. In case of buy behaviour the agent can start with a certain price and can have a maximum for a higher bid if the bidding is rejected. The selling agent on the contrary has a minimum price and will sell to the highest bidder if its bid is above the minimum price If the bid from different buyers are the same and above the minimum price the first arrived bid will be accepted.

5.4.2 Related Work

The product agent proposed here has its role in the use phase for repair and in the recycle phase. When we consider the use of a product this is also studied by observing and/or interviewing end users (Nielsen and Levy, 1994) (Nielsen and Mack, 1994). Some software applications do connect with their originating company to report the use by end users. Several proposals and implementations of including monitoring and documentation within the product itself are made and implemented. Burgess (Burgess, 1998) (Burgess et al., 2002) describes Cfengine that uses agent technology in monitoring computer systems and ICT network infrastructure. Cfengine is already discussed in section 5.3.2.

By using this same agent again in the final phase of the life-cycle, component reuse and smart disassembly is a very important aspect when it comes to recycling of rare or expensive building material. Research in the field of recycling is overwhelming. Ellis (Ellis et al., 1994) describes industrial methods to recycle rare earth elements. This article is about metallurgy and not about using information technology. Kovacs (Kovacs and G., 2008) proposes agent technology in car-recycling. This work focusses on exchange of information between enterprises that recycle and destruct used cars. There is however not a notion of a product agent in their approach. Another difference with our approach is that it focusses only on cars. The work of Graedel et al. (2013) shows the dependency of our modern society of several elements that are becoming rare and thus expensive. Recycling from devices where these elements are used is one of his proposals. Another proposal is searching for alternatives.

5.4.3 Conclusion

Adding an agent to a product that has knowledge of the product and the way it is made can help to extend the lifetime of a product. Besides it gives an opportunity to locate and reuse rare elements. The product agents are autonomous software entities that can assist in recycling.

In the concept that has been developed, agents play an important role in the whole life cycle of a product. This concept can be an enabling technology for the internet of things. The product agent will be the representative of a product. It is a software entity that collects information for a product from the internet, shares information with other product agents and sends information to other product agents.

Interesting further research will be the insight in the reliability of subsystems. The distributed multi agent system of autonomous product agents can generate all kinds of statistical interesting data about the MTBF of these subsystems. This could help manufacturers to improve the quality of their products.

5.5 Case: Domotics

This section describes the case of an agent-based architecture for domotics. This architecture is based on requirements about expandability and hardware independence. The heart of the system is a multiagent system. This system is distributed over several platforms to open the possibility to tie the agents directly to the actuators, sensors and devices involved. This way a level of abstraction is created and all intelligence of the system as a whole is related to the agents involved. A proof of concept has been built and functions as expected. By implementing real and simulated devices and an easy to use graphical interface, all kinds of compositions can be studied using this platform.

The components used in this case are good examples of what the grid-based manufacturing system described in the previous chapters could be used for.

5.5.1 Introduction to domotics

An interesting application field for agent technology is domotics. Domotics is also called home automation and it is a field within building automation. Though building automation focuses normally on big buildings where people come together for work, education, shopping, recovering, sporting or having a meeting, domotics is specializing in the specific automation requirements of private homes. The application of automation techniques is meant for the comfort and security of its residents. Domotics applies many techniques used in building automation such as light and climate control, control of doors and window shutters, security and surveillance systems, etcetera but additional features are used in domotics. These additional functions in home automation include the control of multi-media home entertainment systems, automatic plant watering and pet feeding, and automatic scenes for dinners and parties (future.wikia.com/wiki/Domotics, 2008). Additional features are also security and adaptation of the system to the behaviour of the inhabitants.

An important difference between building automation and home automation is, however, the human interface. In home automation, the control of the system is not done by highly trained technical people as is the case in building automation. Because the control should be done by the home inhabitants the control should be easy, largely image-based and self-explanatory.

Home automation could use wireless techniques, but normally a wired infrastructure is used. A wired infrastructure is a bit more reliable and more tampering proof. When home automation is installed during construction of a new home, control wires can usually be added without much extra work. In standard automation systems these control wires run to a controller, which will then control the environment. However, in practice home automation is often added after the home has been built and even then it should be easily adaptable in the future when new opportunities and techniques become available. In automation there is a trend towards more intelligent devices and a distributed approach for the system as a whole.

In the next sections at first we focus on domotics and its characteristics. In this section also the goal of the research project is explained resulting in system requirements. Next, the design of the system is discussed. In that section hardware and software platforms are introduced. The system architecture is explained in a separate section that will be followed by the implementation, the results and a of course a discussion about related work and a comparison of our work with other research in the field of domotics.

5.5.2 Characteristics of domotics

In this section we first discuss domotics and its levels and global architectures. Next the formulation for the goals of our system will be introduced as well as the global system requirements.

Domotics

Home-automation is sometimes used as a synonym for domotics, but Harper (Harper, 2011) describes five levels of home automation and states that only level four and five apply to domotics (Harper et al., 2003). The five levels are:

1. Homes containing stand-alone intelligent objects.

2. Houses containing intelligent communicating objects. In this case a performance gain can be achieved by sharing information between the objects.

3. Homes that communicate by themselves. In this case internal and external data communication networks open the possibilities to remote control and monitoring.

4. Learning homes: activity patterns are recognized and applied to optimize the technology in house.

5. Attentive homes: the activity and location of people and objects within the homes are constantly registered, and this information is used to control technology in anticipation of the occupants needs.

Looking at these levels we observe an increase of the amount of communication, interoperability and artificial intelligence techniques going from the first level to the highest level. Thus to open the road to the highest level, from the starting point technologies should be applied that do not obstruct this path towards higher levels. In (future.wikia.com/wiki/Domotics, 2008) three possible architectures are described.

1. Centralized architecture: a centralized controller gets information from all kinds of sensors or sensor networks and controls the actuators available.

2. Distributed architecture: the sensors and actuators are intelligent by themselves and communicate to get the desired actions.

3. Mixed architecture: both sensors and actuators are intelligent but there is also a central system to coordinate the actions.

Goals for our system

For our domotics system the most important goals are to develop a system that is simple to use, easy to implement, reliable and expandable. To achieve these goals interoperability between components is an important issue. This interoperability could be possible by adhering to open standards that are widely supported like network protocols and software platforms that can be connected by applying these protocols. These platforms should support modern software applications. Nowadays powerful computing platforms are available having a small size and a low price.

Global system requirements

Considering the characteristics and the aforementioned goals, we come to the following system requirements.

1. Modularity: to be expandable a modular design must be followed.

2. Configuration of the system should be easy but the configuration system should also be expandable.

3. Maintenance and monitoring should be properties of the system to assure high reliability.

4. Adaptivity to new situations, like new devices and new rules for operation should be possible.

As a general definition we consider a device to be a sensor or actuator. The system is rule-based where rules are applied to events, measurements and preferences of the end-users. By applying these concepts the domotics system is not only a set of automation islands, but offers integration of the different parts. Smart integration, where rule-based knowledge adjusts to the needs of the users (humans) should be possible. To make this possible, end-users are represented by agents to communicate their preferences to the system. A high level of integration is achieved by putting agents in devices at the hardware level, thus introducing a kind of abstraction layer. The system as a whole is a multiagent system (MAS) and its working is based on interagent communication. The reasons for this approach will be explained in the next section.

5.5.3 Design considerations

To design the domotics system according to the requirements mentioned in the previous section, some considerations have to be made. Why is agent technology apt for the system, what communication model should be used and finally is there a relationship with other work in our research department.

Agents

A device in a domotics system is acting in an environment and its actions are influencing that same environment. The devices should have the possibility to communicate and cooperate to achieve systemwide goals. Looking at these requirements and properties of devices, they are also found in the definitions of agents for example the definition given by Wooldridge (Wooldridge, 2009): *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.* Considering these similarities agent technology seems to be a natural choice.

To mention a few tasks that should be done by the device agents:

- At start-up testing the device and registering the device in the MAS.
- Interfacing at the lowest level with the hardware for actions to be performed or measurements to be done.
- Offering an abstraction layer usable for interoperability.
- Monitoring the use and health of the device.

Communication models

When agents must be coupled with devices several approaches are possible. In Figure 5.24 the situation is shown where agents reside in an agent runtime environment that is coupled with the actual devices by several means of communication links. Interfaces will provide the actual coupling. A drawback of this situation is that most devices offer different interfaces so the communication methods depend largely on the types of devices used. The advantage is that interagent communication is simple, having the agents running in the same environment. Another possibility is shown in Figure 5.25. Here the

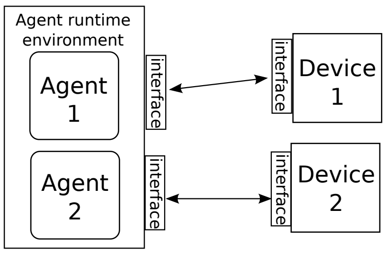


Figure 5.24: Communication model 1

agents are tightly coupled with the devices and a network, depicted as a cloud, will be used for the agents to communicate. In this model the communication is only between agents and this can be done on basis of TCP/IP using standard webtechnologies. The problem that agents are running on different platforms can be solved by using an environment that makes interoperability of agents on different platforms possible. As we shall see the Jade platform that was selected for this research couples different containers running agents over the network in a transparent way.

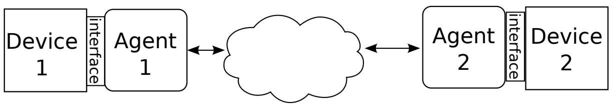


Figure 5.25: Communication model 2

Connection with related cases

In the previous two cases, the roles of agents in the life-cycle of products were investigated. From this research it became clear that adding an agent to a product or device offers all kinds of possibilities and advantages. To mention a few advantages of adding this so called product agent:

- If connected to the internet, products can communicate worldwide. The embedded agent is the enabling technology for the concept of the Internet of Things (Ashton, 2009).
- The embedded product agent can monitor the use of a product.
- The product agent can perform a power-on self test (POST) where the functionality of the product is tested every time it is switched on and the product agent can also test the subsystems of a device.
- In case of a broken subsystem the embedded agent can search for a replacement.

Considering these advantages embedding the agents in the device itself or make a tight coupling with the device as proposed is Figure 5.25 can introduce the aforementioned advantages.

It is important to emphasize that the role and responsibilities of product agents in the use-phase of a product is different from the role and responsibilities of the agents that are part of the domotics MAS. However, a product agent is closely tied to a product and thus needs a hardware platform to run on. It can share this platform and software environment with the agents that play a role in de domotics system. However, there is no reason why the product agent should not take a more active role in the device. This is what has been done in this research: giving the product agent a more prominent role. This way, it is a natural step to create a distributed multiagent system as the basis for our domotics system and the advantages as described here for having product agents in the system are also available.

Hardware set-up

To implement the distributed approach it is necessary to create an environment for agents near the devices. Devices were equipped with a small computer system for the agents to run. After some research for platforms for our system, the Raspberry-Pi seemed to fit the requirement of offering a stable and cheap hardware platform, capable and powerful enough to run a Java virtual machine to support the Jade environment, offering standard Ethernet connection and last but not least having the possibility to attach the hardware device itself to this system. Though this might seem overkill to use such a sophisticated device, it offers opportunities for the embedded agent having a huge system resource for future expansions. These resources might be needed is case of expanding the domotics system to the level of attentive homes.

Software platform

A short investigation of available platforms has been done to select a platform that fits the need for this project. For the selection of the software platform, the following considerations were used:

- Is there an active development on the platform?
- What type of licence has the software? An open source-based platform is preferred.
- Is the platform widely used?
- How good is the available documentation? This is also related to the first criterion, because an active development is many times always a reason for participators to write good documentation.
- Compatibility with operating systems. A platform that runs on several standard operating systems has an advantage over a platform that can only be used on a specific operating system.
- What is the basic language of the platform? A standard language is preferred over the use of uncommon languages.

The list of platforms that has been investigated for tis research consists of:

- ABLE.
- DIET Agents.
- FIPA OS.
- JACK intelligent agents.

- Jade.
- Spade.

All these plaforms except for Spade are Java-based, meaning that they can run on a operating system that offers a Java-virtual machine. Spade is based on Python and is also runnable on several operating systems. For the documentation Jade seems to be the best one, having several tutorials, an active community publishing documentation and examples. There is also a book available describing the architecture, installation and use of Jade (Bellifemine et al., 2007). In the software developers group working on this project, there was already some experience using Jade, combined with the advantages over alternatives mentioned here the choice for Jade as a software platform was made.

The Jade runtime environment implements message-based communication between agents running on different platforms connected by a network. In Figure 5.26 the Jade platform environment is depicted.

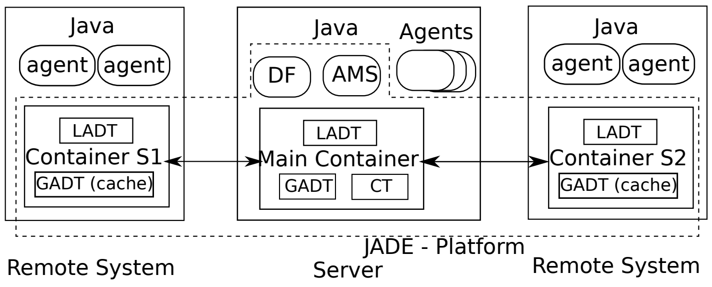


Figure 5.26: The Jade platform

The Jade platform itself is in this figure surrounded by a dashed line. Figure 5.26 is slightly different from an earlier Jade platform picture in this thesis. In that earlier picture the position of the equiplet agents and product agents was shown. Figure 5.26 is a more generic picture. It consists of the following components:

- A main container with connections to remote containers.
- A container table (CT) residing in the main container, which is the registry of the object references and transport addresses of all container nodes comprising the platform.
- A global agent descriptor table (GADT), which is the registry of all agents present in the platform, including their status and location. This table resides in the main container and there are cached entries in the other containers.

- All containers have a local agent descriptor table (LADT), describing the local agents in the container.

- The main container also hosts two special agents AMS and DF, that provide the agent management and the yellow page service (Directory Facilitator) where agents can register their services or search for available services.

Agents running on this platform are also visible in Figure 5.26. These agents can be implemented in Java by extending the agent-class offered by Jade. Every container can run its set of agents and these agents can communicate with each other.

5.5.4 Software architecture

In this section the architecture of the system is presented. First the global architecture. In the next subsection the roles and responsibilities of the agents involved are discussed as well as the global architecture of two design models of the device agent. Finally the interagent communication and message types are presented.

Global system architecture

In Figure 5.27 the global architecture of the domotics system is shown. A GUI subsystem is provided for configuration, control and monitoring. The blackboard system in the middle is the place where all relevant information that could be shared among the participating agents is collected. This blackboard system supports a publish and subscribe system that is used for interagent communication. At the bottom we see the actual device agents.

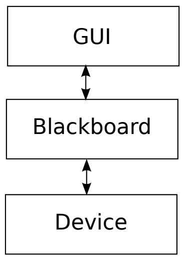


Figure 5.27: Global system architecture

Agent roles and responsibilities

In agent-oriented software engineering (AOSE) (Bordini et al., 2006), the roles and responsibilities of the agents form the basis of the agent software model. Our MAS contains four types of agents.

1. The device agent, closely coupled to the hardware devices in the domestic environment.
2. The human agent representing the human inhabitants of a home.
3. The blackboard agent controlling the interagent communication and the storage of important data.
4. The GUI-agent serves as a middleman between the GUI and the MAS.

Looking at roles and responsibilities results in the following observations: The device agent will directly control a device. The actual control depends on the device being a sensor or actuator, so actually a device agent can be an actuator agent or sensor agent. It will receive information that it has subscribed to and it will publish information on the blackboard. Depending on the information received and its rule-base it will control the device. In an earlier section 5.5.3, the concept of a product agent was introduced. This product agent could be the representation of a product in the Internet of Things and has the responsibilities mentioned in section 5.5.3. Being tightly coupled to a device (actually a product) and capable of communicating, the device agent could possibly also play the role of a product agent, this means; monitor the device, perform a power-on-self-test to check the health of the device and collect information about usage of the device. However, the product agent can also be implemented as a separate agent running on the same hardware platform in the same software environment as the device agent.

The human agent will present a human in the system. This agent is implemented as a sensor agent and shows the location of a human inhabitant along with its preferences and physical situation. The blackboard agent will control the agent network, storing information and giving information to other agents. It will keep track of subscriptions done by the other agents and will inform these subscribed agents when requested or when an update is done by another agent. The GUI-agent: is a part of the GUI subsystem. It has been implemented to make communication with the blackboard agent and other agents in the MAS at an interagent communication level possible.

In Figure 5.28 the inheritance model of a device agent for a lamp device is shown. First a general device agent will be responsible for communication with the outside world. This agent is expanded to a lamp agent at the application layer and finally this results in a real hardware lamp agent or a simulated lamp agent. This way it is easy in our implementation to introduce simulated devices. These simulated devices are visible in a GUI environment and can be used to build a system for testing purposes or in situations were the actual hardware is not yet available. The actual working of these non-existent devices can be observed using the GUI.

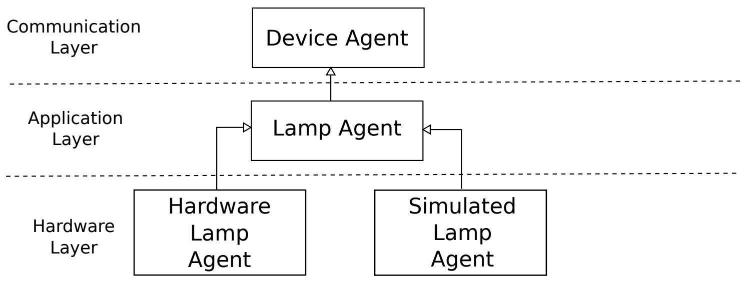


Figure 5.28: Inheritance model for a lamp device

In Figure 5.29 another type of device agent is shown: a light sensor. This agent is also derived from the device agent. However at the application layer an extra functionality typical for sensors is added. This extra block performs polling of the sensor to get new data values.

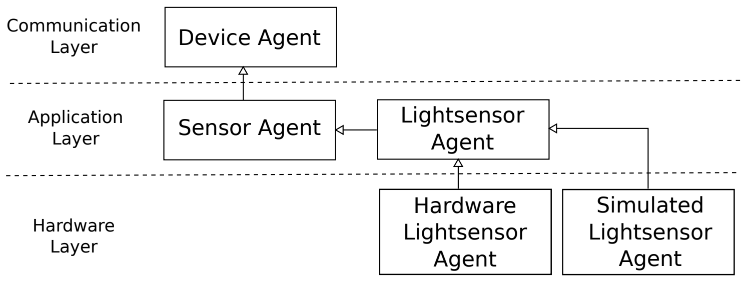


Figure 5.29: Inheritance model for a light sensor

Interagent communication

All communication for the system as a whole is done at the agent level. The device agent have their own specific and perhaps dedicated communication interface with sensors and actuators. The Jade environment supports the FIPA-standard for interagent communication (FIPA stands for Foundation for Intelligent Physical Agents). So the standard FIPA possibilities are already implemented and supported. Within FIPA a message format must be chosen. Four possibilities were investigated:

- Design of a new specific format.
- CSV (Comma Separated Values).
- JSON (JavaScipt Object Notation).
- XML (eXtensibel Markup Language).

The first two options were rejected. A new format means a lot of extra software tools to be developed. CSV is too primitive, for example nesting is not supported. This leaves JSON and XML as a choice. For both choices validator tools and libraries are available, however XML is more mature and this was the reason that at that time XML has been chosen.

A message contains the following items:

- A value.
- A key, if a device can send different kinds of values.
- The topic the value is related to (i.e. light etc.).
- An agent/device identifier.

Using this information a device agent can send different values by submitting key-value pairs, where key identifies a specific type for the value. By sending these kind of messages to the blackboard agent, the information is stored in a database and the blackboard agents will direct this information to other agents that have subscribed to this information.

Several types of messages are possible: to mention a few:

- Subscribe: subscribe to information for a certain topic. If the information of a topic changes, automatic information update is sent to a subscriber.
- Unsubscribe: used to stop a subscription.
- Request Value: get information from the blackboard about an agent.
- Publish: this can be done by a device agent. This way it will put information on the blackboard.

5.5.5 Implementation

In the next three figures some details of the internal structure of the agents and gui system is shown. The communication layer is part of all agents and the GUI system. In Figure 5.30 the internals of a device agent is depicted. These agents interact with the actual hardware using the hardware layer though it is also possible to simulate the hardware. The embedded GUI subsystem in this agent is available to monitor the actual devices or in case of a simulation the simulated device. Artificial intelligence software runs in the application layer in combination with a message processor for interpreting the messages received from other agents (using the blackboard) or to construct messages meant for other agents. The blackboard agent takes care of data

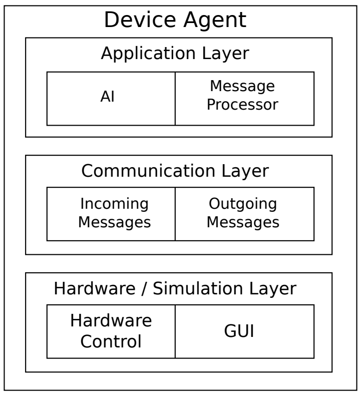


Figure 5.30: Internal structure of a device agent

storage in the application layer and the message processor implements the publish and subscribe mechanism. The communication layer serves the same goal as in the device agent: supporting communication with other agents. The GUI system is not an agent, but has an embedded agent to make it

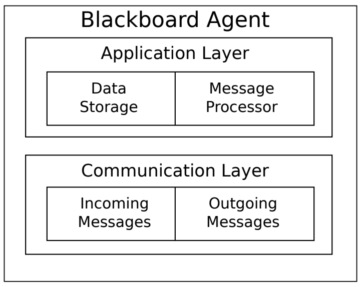


Figure 5.31: Internal structure of the blackboard agent

part of the multi agent system. This agent has only a communication layer

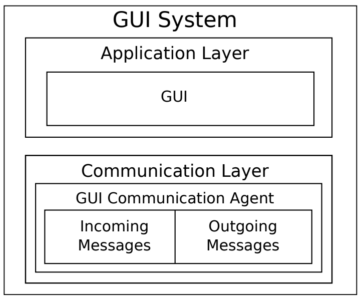


Figure 5.32: Internal structure of the GUI system

to support the communication with the MAS. The actual implementation was made on a standard desktop system connected by ethernet interfaces to several raspberry-Pi systems. Ethernet over powerlines was used to minimise additional cabling requirements. Because many devices are actually connected to powerlines this approach seems to be the most natural. In the future devices could embed an agent environment system based on embedded technology and by attaching these devices to the power, the communication infrastructure is immediately established. On top of ethernet, TCP/IP is used as the carrier of interagent communications. Devices having special interfaces are connected to the agent platforms close to these devices. This way it is not necessary to support all kinds of exotic or non-standard cabling systems.

Using the GUI drawing tool, a map of the home was easy to draw and in this map the position of several devices could be drawn. The map is stored as an XML file so other XML aware applications can easily get the actual information data about the map. Figure 5.33 gives an impression of how this part of the GUI looks like. This tool is based on graphic standards that can also be used in other applications. This way a portable and open system is also applied at the GUI level. Using this map, several types of devices can be added to the system by the end-user of the system. A device-related agent will be created as well. In Figure 5.34 a dialog window for creating an new device is shown. Using this GUI a domotics system for a home could easily be built by the end-user because of the intuitive and simple user interface. As shown in Figure 5.34 a device can also be removed from the system. This removal includes the device-related agent.

5.5.6 Results

What has been created is a domotics implementation based on agent technology. By using agent technology the devices involved were closely tied to

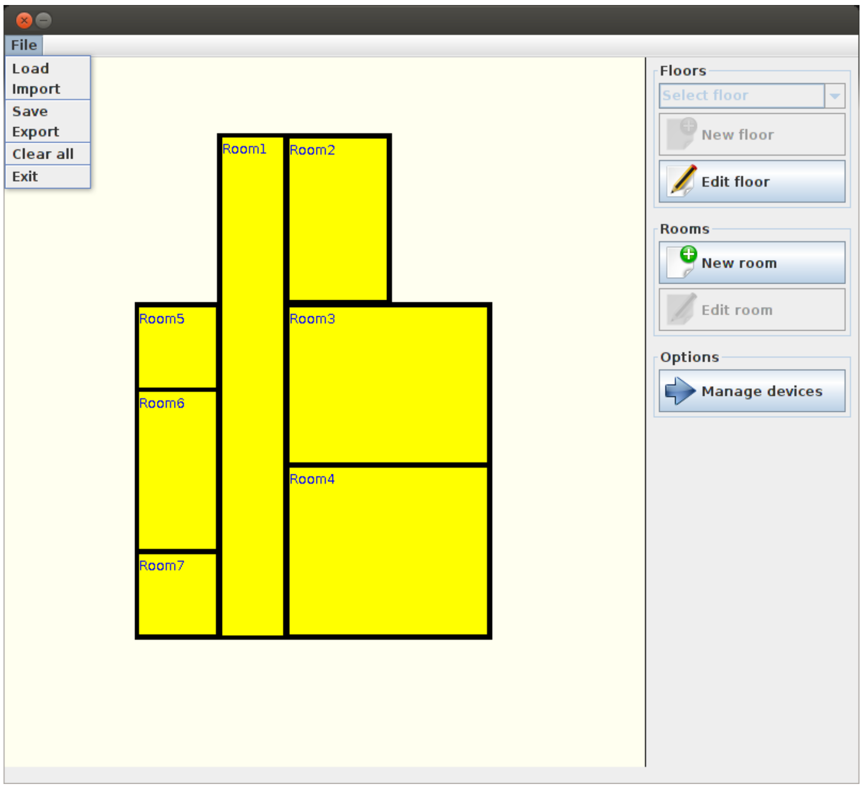
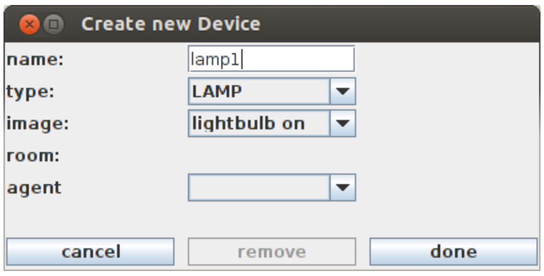


Figure 5.33: The design part of the GUI subsystem

agents, making these devices versatile as well as intelligent. For the system as a whole these devices could be considered as entities capable to operate at a high abstraction level. These combined device-agent system can be seen as nodes operating in the internet of things. By using a blackboard and a publish and subscribe system these agents could interact and exchange information. This fits the basic requirements of the system.

Figure 5.35 shows the actual implementation of the lamp-device. By just plugging in this device into the power outlet the device is powered, the Raspberry-Pi is activated and the communication over the power-line is set up. The lamp is coupled with the raspberry-Pi by a simple interface on the breadboard shown in the picture. This interface is connected to a relay for switching the lamp on and off. The light sensor device is shown in Figure 5.36. Other devices like controllers have been implemented as simulated devices. The same is true for the agents representing the human inhabitants. These simulated agents can be accessed and controlled by the GUI to check the working of the domotics system as a whole. In the domotics system, both communication and the software implementation were based


Create new Device

name: lamp1

type: LAMP

image: lightbulb on

room:

agent

Figure 5.34: Adding a new device to the system

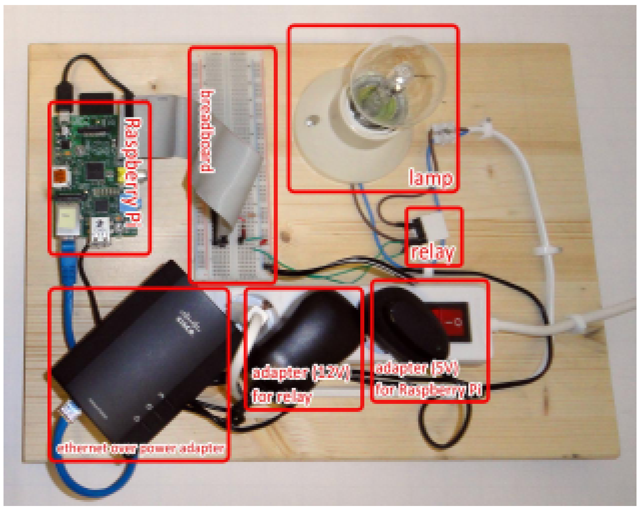


Figure 5.35: Prototype of the lamp device hardware

on open and widely available standards. This makes it easy to expand the system and combine it with other open standard-based techniques. This was an important goal of this project.

5.5.7 Discussion

First the focus is on the network infrastructure used in our solution. In domotics among others KNX is a widely supported solution (www.knx.org, 2012). Most KNX-based systems use a centralized control and configuration system. KNX claims to have the following advantages:

- Interoperability. KNX devices from different manufacturers will opperate together.
- International standard. KNX is an international standard, adopted among a wide range of manufacturers.

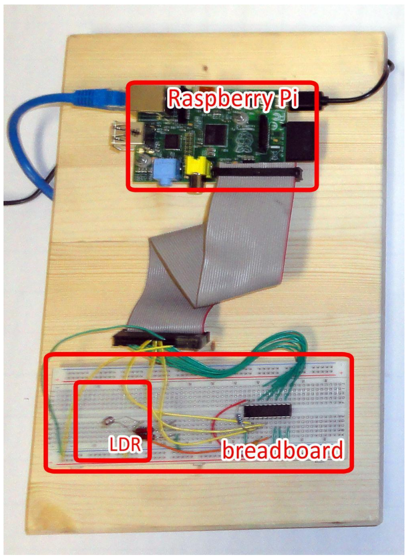


Figure 5.36: Prototype of the light sensor

- High product quality. Products conforming the KNX requirements should have a high quality.
- Manufacturer-independent tooling. Tools for supporting KNX are available from different manufacturers.
- All home and building control. KNX claims to support all control requirements within home and building automation.
- Different kind of building. Simple and complex systems can use the same standard.
- Support for different configuration mode. Simple and complex configuration modes are possible.
- Different communication media, like wireless, twisted pair or using the wiring of the power net.
- Connection to other systems is possible.
- Independent from any hard- or software technology.

Almost all of these advantages except perhaps the high product quality also apply to a TCP/IP-based system as described in the current case in this thesis. TCP/IP does not guarantee high quality of a product, only adhering to the TCP/IP standard. A further advantage is that in most cases existing network infrastructure is already based on TCP/IP thus introducing new devices is easy to accomplish. In industry the last decades have shown a trend towards the use of standard networking techniques like ethernet and TCP/IP replacing so called field-buses except for special situations where extreme conditions occurring in the chemical industry require special solutions or when extremely hard real-time requirements play a role. For most situations however standard networking solutions, often with enhanced reliability like industrial Ethernet, offer the same reliability as field-bus-based solutions and are also very cost-effective. The advent of the next version of the Internet Protocol (IPv6) will also ease the adoption of standard internet in new fields, because the new features like an enormous number of node-addresses, support for different types of network traffic (Flow labelling and priority) and security (IPsec is standard included in IPv6) (Tanenbaum and Wetherall, 2010). Integration with smartphones becomes easy because these systems can operate in dual mode. This means that they can communicate using the cellphone communication infrastructure as well as WiFi to connect to a wireless LAN. By using the cellphone network they are capable to cooperate with the home network from all over the world. If the smartphone is within the reach of the local wireless network it can use WiFi and become a node in the network without introducing extra costs.

As a second aspect we will now focus on the properties of the agent-based solution proposed in this section of the thesis. What has been done was bringing a powerful yet cheap and reliable platform close to the device. This approach also fits with the product agent concept introduced earlier in 5.5.3. This agent is monitoring, performing a power-on self test and other tasks mentioned in section 5.5.3. This results in two levels of error detection. At one level it might be that the whole node is unavailable and therefore this node will be excluded from the domotics system and an error message can be generated on the GUI. At another error level within the node itself a part or subsystem of the device is not operating. In this case the domotics system can pinpoint the problem to be solved.

Using a MAS where humans are represented by agents themselves is an enabling technology to integrate the humans in a MAS-controlled home automation system. A human is represented by an agent in the MAS and is capable to influence the system each using their own preferences.

5.5.8 Related work

The implemention of domotics systems based on agent technology has been done by several authors. Some publications like (Muñoz et al., 2006) and

(Ruta et al., 2006) focus on systems for disabled people. This can be considered as a specialisation within domotics where for example multimodal interfaces (gestures, text, voice and haptic devices) are used to interact with the domotics system. In (Brink et al., 2008) a Jade-based system is presented that has been developed to support elderly people. In that publication the authors also promote the use of open standards and open systems to make interoperability possible. However, their system has been designed for a special target group, while our system is meant for all kinds of implementations of domotics. Other papers describe systems where agents are used to save energy like (Ruta et al., 2012) and (Abras et al., 2006). In this work agents focus on coordination of devices to minimise peak loads or to shift to moments of low energy prices.

The work of Bolzabi and Netto (Bolzani and Netto, March 2009) describe the Home Sapiens, a smart home framework. They developed their own framework where three types of agents cooperate. User agents, representing users, Micro agents, representing devices and system agents to glue these components together. Our approach is different from this: by using Jade and agents that are tied closely to the devices, a clean abstraction layer is presented. All kinds of interaction between these agents is in principle possible without the need for extra system agents.

DomoBuilder presented in (Addis and G. Armano, September 2010) is an agent-based system using the Jade platform. The agent model is used to achieve an abstraction level. So far for the similarities with our model. The difference with our model is that DomoBuilder is using a central controlling system called Kernel. This Kernel has some of the functionalities realised by the blackboard-agent in our model, but it has much more power to actually control the system using timers and event handlers. This results in a more centralized model while in our model the control power is delegated to the agents in the devices, resulting in a distributed model. In case of a communication problem, a distributed model has the advantage that the agents tied to the devices can control these devices themselves using rules how to operate in case of missing information, while in a centralized model the control of the devices is lost.

The work of R. Nunes as presented in (Nunes and da Silva, 2004) and (Nunes, 2003) has its focus on the intelligence of home automation systems as well as smart energy management. In our work we focus on an architecture that enables the use of artificial intelligence techniques.

In (Conte et al., 2009) a Labview simulation is presented by Conte e.a. to control resource management in Home automation systems. In this paper the human user is also considered to be an agent. Though this simulation is based on agent technology, it does not take advantage of an agent-aware platform, as Labview is a programming environment based on graphical building blocks to build all kinds of experimental and technical systems. Being commercial software Labview is also tied to licence costs. Except for DomoBuilder, most systems are based on a central system where the MAS is implemented and agents keep contact with sensors and actuators by a specific data communication infrastructure and interfaces.

5.5.9 Conclusion

In this research we implemented a powerful system. In our proof of concept only a few real devices were implemented. However the concepts that are used makes it easy to expand this approach to a much more complicated system without the need for redesigning the system as a whole. We also implemented simulated devices to show this possibility. An important aspect is the inclusion of a user interface for the end-user for configuring the system for a certain environment. This interface can also be used to adapt the system to a newly created situation. In future research smarter and learning agents will be used. Also the interaction with real human inhabitants should be implemented. This could be done by using smartphone devices but other possibilities should also be investigated. Having this platform available is a good start for this new research.

5.6 Summary

This chapter introduced the product agent as the software representative of a product during its whole life cycle. In three cases the enhancement of a system by adding a product agent was shown. A product agent could be seen as an add-on as in the first two cases, but it can also be a powerful component of a device to make it operate in a network of devices as shown in case three.

Chapter 6

Putting things together

In the previous chapters, aspects of the agile agent-based manufacturing system have been described as well as the use of the product agent in the life cycle of the product. This chapter will be devoted on describing the prototypes of the manufacturing system that has been built so far. Two types of realisation are described: first the equiplet-based manufacturing system and in the second part a manufacturing system that is based on the same architecture but uses human workers instead of equiplets to realise the production.

Parts of this chapter have been published in the proceedings of the IFAC Modeling in Manufacturing (MIM-2013) (Moergestel et al., 2013d), the proceedings of the International Conference on Intelligent Agent Technology (IAT-2013) (Moergestel et al., 2013f) and the European MultiAgent Systems workshop (EUMAS 2013) (Moergestel et al., 2013b).

6.1 Equiplet-based manufacturing

The equiplet-based manufacturing decsription will have its focus on the MAS where the equiplet agent is the representative of the equiplet. The details of the implementation of the equiplet are not the subject of this thesis. Work on the equiplet can be found in (Telgen et al., 2013). The product agent has several roles:

- Path planning of the production.
- Scheduling of the production.
- Guiding the product during manufacturing: The product agent will guide the product along the equiplets. At every equiplet it will instruct

the equiplet agent what step or steps to perform. It will log the results of a production step and also update a globally shared knowledge base that can be consulted by other product agent to check the reliability of a certain equiplet for a certain step with certain parameters.

- Error recovery: Having the responsibility for the manufacturing of a product, the product agent is also the entity that should recover from errors during manufacturing. If there is a failure on a certain equiplet, depending on the type of failure (recoverable or severe) the product agent will try to plan the required step on an alternative equiplet for the same reason as why one would not prefer to hire a plumber who previously made mistakes resulting in a flood. By putting the information about the failure (step type and parameters) in a shared knowledge base, the product agents will learn as a group about the reliability of the equiplets for certain steps. A more detailed discussion about error recovery is in section 6.1.2.

- Role in other parts of the life cycle of a product: As presented in chapter 5 being a software entity that knows a lot about the product and the actual production, there are a lot of possible roles and new goals this product agent can have during the life cycle of the product. To achieve these roles, the agent could be embedded in the product itself. This embedding will also be described in this chapter.

6.1.1 System architecture

In this section a description of the system architecture as well as the software will be presented.

In figure 6.1 the layered software architecture is given. Only one product agent and one equiplet agent is depicted and the modules in the lower layer of the equiplet depend on the front-end that has been connected to the equiplet. In this case an equiplet with the pick and place capabilities and vision modules is assumed.

For the MAS layer Jade (Bordini et al., 2005) was used as a platform. The reasons for choosing Jade have been given in chapter 2.

The software for the equiplet is based on ROS. ROS is an acronym for Robot Operating System (Quigley et al., 2009). ROS is not really an operating system but it is middle-ware specially designed for robot control and it runs on Linux. In ROS a process is called a node. These nodes can communicate by a publish and subscribe mechanism. In ROS this communication

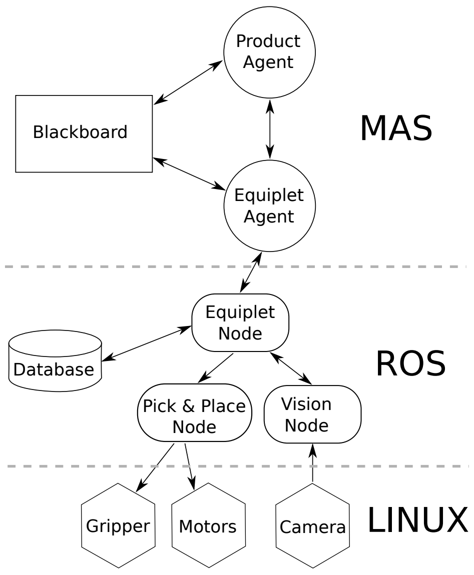


Figure 6.1: Layered architecture

mechanism is called a topic. In section 5.3 of chapter 5 the ROS properties are already introduced. Just as in the case of the discovery robot from chapter 5, this platform has been chosen for almost the same reasons:

- Open source, so easy to adapt, compliant with a lot of open source tools.
- Wide support by an active community.
- Huge amount of modules already available.
- Nodes that are parts of ROS can live on several different platforms, assumed that a TCP/IP connection is available.

At the lowest layer in figure 6.1 is a Linux platform running modules that communicate with the underlying hardware. Linux is a stable, portable and versatile platform. In the next section we will take a closer look at the implementation of this architecture.

6.1.2 Implementation

Before discussing the software for the production grid, a description of the way that products are made in the grid as we designed it is presented.

Production constraints

Our production model is based on trays that will carry the product to be built. These trays are transparent boxes, so equiplets with a camera can inspect both from the top and the bottom. In the latter case the workplace of an equiplet should also be transparent, which is the case for the equiplets built so far. The trays are marked with an unique QR-code. During the first production steps the trays are filled with all the components required to make the product as described in chapter 4. This way a kind the construction box as presented in chapter 4 is generated. This means that for all steps to come, the components are available. This is a big advantage over a situation where logistic streams of components within the grid should be taken care of. The disadvantage is that parallel production of sub-parts in complex production paths is not possible. However for the proof of concept this is not a big problem and solutions can be found where the sub-parts are first manufactured in parallel and added to the construction box as described in chapter 4. Of course within our conceptual model other production models could be used, but the examples given here are based on this model.

Webinterface

To test the production grid, a webserver has been added to allow end-users to construct products to be made by the grid. This is why it can not happen that a product is requested that does not fit within the capabilities of the production grid, because the grid itself is offering the webinterface for designing the product. No norms are needed for the product agents acting within the grid. If a product can be made using the webinterface, the grid will be capable to make it.

The addition of a webinterface is not only for testing purposes, but this idea fits neatly in the concept of agile manufacturing, where the end-user plays a prominent role in the production itself. The end-user specifies the product that will be tailor-made to his or her requirements. This pull-driven type of manufacturing will not lead to overproduction and waste of material.

The architecture of the software of the manufacturing system is depicted in figure 6.2. A web server publishes a website where a customer can design his product. By pushing a submit button, a server-side program will create and activate a product agent. This agent will start to plan the production path and communicate with the available equiplet agents to create the product. A more technical picture showing the distributed nature of the system is given in figure 6.3. The numbered components in figure 6.3 are:

1. The client PC as used by end-user. The end user can use any HTML-5

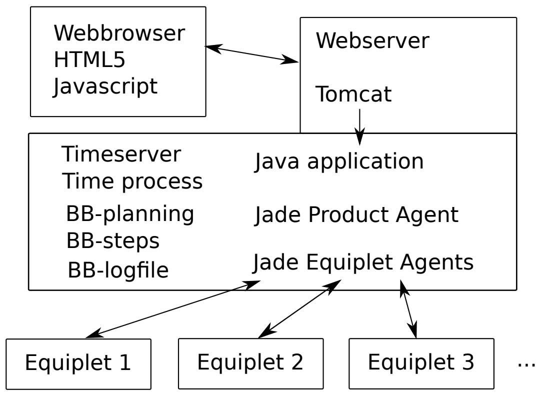


Figure 6.2: Combination with webinterface

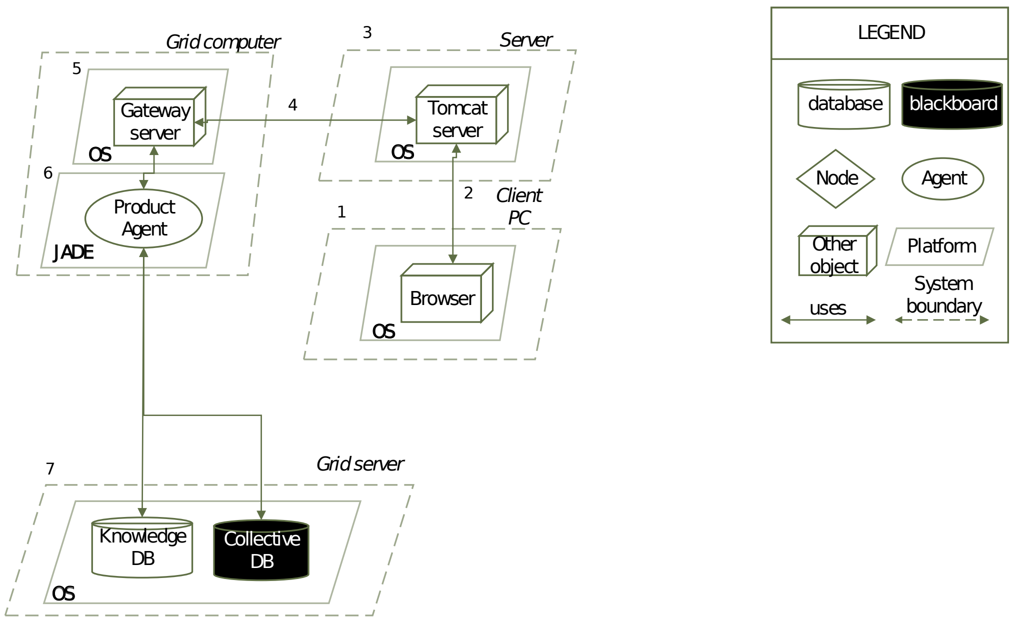


Figure 6.3: Different platforms and their relations

enabled browser.

2. Connection to the Tomcat server is established via a web socket.

3. The Tomcat server on which the website is hosted. The server can be placed on the grid server, but it can also be located somewhere else.

4. A connection between the gateway server and the Tomcat server is made through a (Java) socket.

5. The gateway server is responsible for spawning a product agent in the

jade container. The gateway server acts as a *gateway* to the outside world, implemented to be able to spawn agents.

6. The Jade container of the grid contains all agents. Agents can communicate with the Tomcat server.

7. The grid server is where the databases and blackboards reside. These are the systems where shared and individual knowledge will be stored.

8. Agents have to be able to report back to the user. In order to do so, an algorithm was implemented to allow them to send information over a socket. In order to keep the connection alive, a heart-beat system has been developed. This system will be described in the next section.

Communications with the web interface/Tomcat server

Once a product agent is created through the web interface, the agent will create a socket behaviour. This socket behaviour is the way for a product agent to communicate with the server and thus to the web interface. To check whether or not the server is still alive and reachable a *heart* message is sent. If this message is not answered with a *beat* message it is assumed that the server is down. This is how the socket behaviour is used and implemented: The socket behaviour is used for the communication with the web interface. This web interface will be called WIMP as an acronym for Web Interface Managing Production. The socket behaviour extends the Jade Waker behaviour which means it will become active after a certain amount of time. At the time of writing the wake up period for the socket behaviour is set at 5 seconds. This means that every 5 seconds the socket behaviour will become active and check if it is connected to the WIMP server. If it is connected it will check if there are data in the buffer, if any it will process the data. If the buffer is empty or if all data is processed the socket behaviour will go idle and will become active once the Waker behaviour is fired again (which at the time of writing means it will become active after 5 seconds). The socket behaviour can also be used to write messages to the WIMP server even if the socket behaviour is not active, this is because it will be executed within the action method of another behaviour.

The heartbeat behaviour was created to eliminate a problem which we were having with the socket behaviour. The problem encountered was that the socket behaviour is unable to see if the socket connection is still alive, if it is not closed properly. The socket behaviour will only know if the connection is closed when either the client closed it properly or when the socket behaviour is trying to write on the socket when it is closed. Because we can receive commands from the WIMP server, we need to be sure the connection is active. If the connection is closed, but the socket behaviour is not aware of this, that would mean that the socket behaviour simply cannot receive messages from the WIMP server. And since the socket behaviour does not know the socket is closed, it will not try to reconnect. This was the reason why the heartbeat behaviour was created. The heartbeat behaviour sends a *heart* message every 5 seconds and sets a timeout timer for 15 seconds. After sending a heart message the heartbeat behaviour expects a response within 15 seconds from the WIMP server. The response should be a *beat*. If it does not receive a response message within 15 seconds it will report to the socket behaviour that the connection is no longer active and will tell the socket behaviour to reconnect. If it is not possible to reconnect immediately, the socket behaviour will try to reconnect every time it becomes active (every 5 seconds).

WIMP

At the client side a web-browser receives a web-page in HTML5 format with embedded JavaScript will display a graphical environment where a product can be designed. This is what has been called the WIMP. At this moment 4 typical product design web interfaces are constructed in WIMP:

1. Pick and place: 2D ball in cradle placement.
2. Paint pixels: pixel-based picture.
3. Pick, place and stack: simple 3D design.
4. Inspection of 3D printing object in STL-format.

A simple example of the pick and place interface is shown in a screen-shot in figure 6.4. A case with compartments of a certain dimension specified by

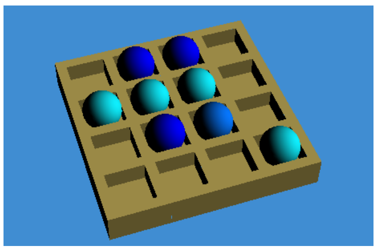


Figure 6.4: Case with coloured balls in the webbrowser

the user is to be filled with coloured balls. The end-user selects a ball of a certain colour and moves the ball to an empty compartment.

An example of a sceenshot of the paint design interface is given in figure 6.5. On a canvas, a pixel-based painting using a combination of several colours can be made.

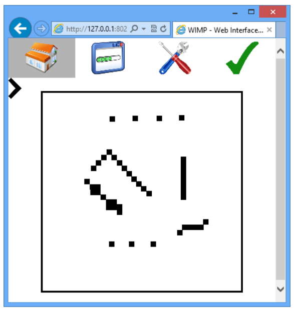


Figure 6.5: A simple paint example

The WIMP software is also capable to build three dimensional structures. It has some built-in intelligence. For example if a user wants to add a part at a place where adhesive is needed to keep it in place, it will warn the user if he did not select the adhesive option for the placement of this part. This

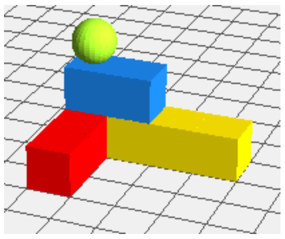


Figure 6.6: A 3D structure

part of WIMP is only a basic implementation and in future development all kinds of special provisions should be added. For example when gluing two objects together several points of special interest arise. First of all the location of the objects you want to glue is very important. If the object is glued onto an existing structure it is possible that the existing structure will tip over. The structure must be stable enough and strong enough to support the new object. To determine if those conditions are met you have to know the material of the current structure, how much it weighs, and several other factors. Another important aspect of gluing objects is the type of adhesive. Not all materials can be glued together and not all types of adhesive can be used in combination with all materials. During manufacturing the objects that will be glued must be held together. This must be done until the adhesive is dry. Some adhesive types need heat to function properly, other types can be hardened by using UV-light (Lafeber et al., 2012).

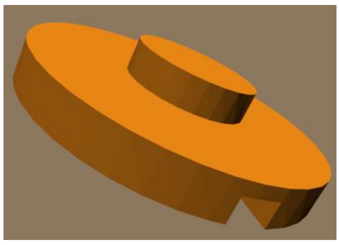


Figure 6.7: View of an STL-image

At the client side a product is described by JSON. JSON, or JavaScript Simple Object Notation is a popular alternative to XML. XML was almost the de-facto standard before the existence of JSON. Until HTML 5, you needed to include libraries to encode and decode JSON objects. Now, the JavaScript engine that comes with HTML 5 has built-in support for encoding/decoding JSON objects. For every part placed on the design grid in the webbrowser, the parttype (ball, or block), colour (red, blue, green, yellow) and position (coordinates on the design-grid) is entered in this JSON information. It is also possible to choose whether or not to use adhesive. By clicking the submit button, the JSON information is transferred to the webserver. In figure 6.8 the internal structure of a product step information block is given. A unique ID is followed by a capability. This is the step action required and will be tied to an equiplet capable to perform this step. The parameters give extra information about the object the action has to work on. For example in a pick and place action, the parameters will specify the coordinates of the final positions and the object that has to move to that position.

ID	Capability	Parameters

Figure 6.8: Components of a step object

Webserver and Tomcat-driven Java application

The web page presented to the client is presented by a Tomcat web server. Tomcat is designed to support Java Servlets. This means that Tomcat is capable to start a Java program at the server the moment the client sends a request for a product. This Java program is capable of spawning a product agent in the Jade environment. To do this a Gateway is used in the Jade environment to achieve this functionality. This newly spawned agent will also receive the JSON information about the product to be made. From this information, the needed product steps are generated by the product agent. An overview of the connection sockets is shown in figure 6.9.

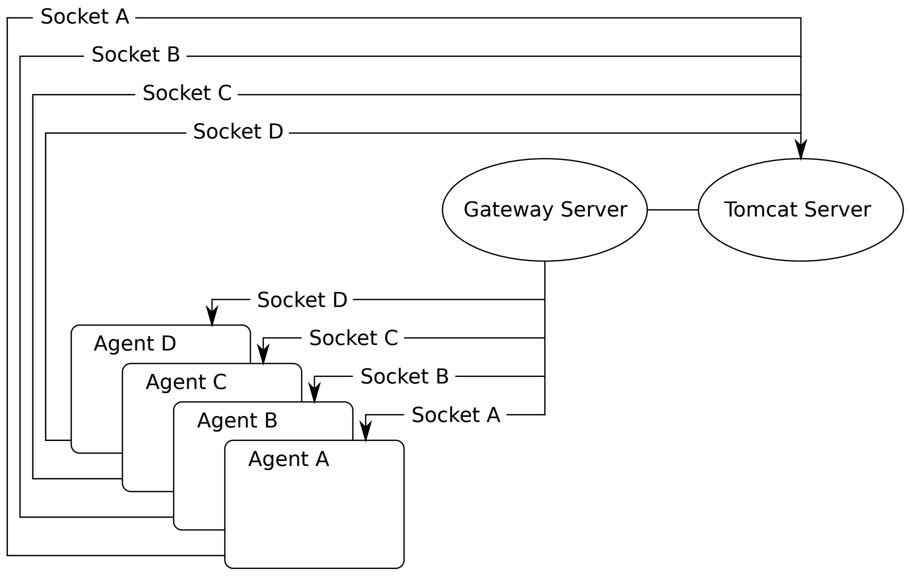


Figure 6.9: Socket connections between product agents and the user interface Every product agent is capable to receive information from the Tomcat server using the Gateway Server. Every product agent can also directly send information to the Tomcat Server. This will create the possibility to inform the end-user in realtime about the progress of the production.

Product agent

The product agent is created and its goal is to produce the product. Therefore it has to fulfil its sub-goals. The first sub-goal is planning the production path. This means: selecting the equiplets involved, inquire if the steps are feasible and finally scheduling the production. The next sub-goal is to guide the product along the production path and to inform the equiplet about the step or steps to perform. For every step, data aquisition of the production data is possible and should be carried out by the product agent. It depends on the equiplet agent what information will be made available.

Blackboard and timing

The blackboard system as described in the architecture was implemented as actually three separate blackboards (see figure 6.2). This has to do with the fact that the performance of the system could be better and also the read and write access permissions become more clear. The BB-steps blackboard is used by the equiplet agents to announce its production steps. This information is under normal circumstances read-only for the product agents. The BB-planning blackboard is read and written by the product agents and a timing process. The information on this blackboard is the planning of timeslots or time steps for every equiplet, and a load of every equiplet.

The implementation is based on a circular buffer of time steps for every single equiplet. A time-pointer is pointing at the current time step slot. This pointer is updated by the timing process, that will also clear the time step slot that has been passed. It will also update the load of the equiplet when a filled time step has passed by and thus has been cleared by the timing process. Other time steps slots contain the unique product agent id that belongs to the product agent that has reserved that slot. The selection of

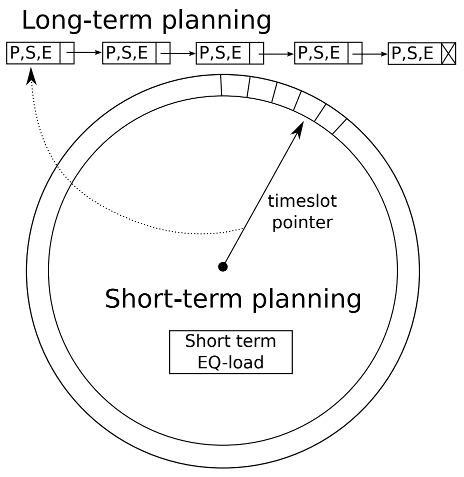


Figure 6.10: Long-term and short-term planning

the time step slot duration was based on the consideration that a product will arrive by an AGV. The product has to be at rest during the production step and the AGV has to move away from the working place to make place for the next product to arrive. This whole process takes at least a few seconds. A choice for one second has been made and given a buffersize of 10000 slots this means that the short-term planning is for around 3 hours. By increasing the buffersize or enlarging the timeslot this can be adjusted to the properties of the production devices in the grid. As a consequence it should be possible to combine several production steps having a short duration in one timeslot if these steps are to be performed by the same equiplet.

To synchronise all agents, a timeserver has been added to the system. The scheduling is done by the product agents. Every newly arrived product agent tries to schedule itself in a way that it will not exceed its deadline. If it fails, it will ask other product agents with a later deadline to temporally give up their scheduling. Next it will try to generate new schedules for all involved agents. If successful, the new schedule will be adopted. If the scheduling fails the old schedules are restored and the new agent reports a scheduling failure. This solution has been described in chapter 3.

The third blackboard in figure 6.2 (BB-logfile) is used to build a knowledge base about the performance of the individual equiplets and is shared among the product agents. Successful and unsuccessful steps are reported in this blackboard by products agents. This blackboard serves as an extra check when the product agent is planning the set of equiplets to be used for a certain product. The higher the failure rate of a certain equiplet, the more it will be avoided by the product agents. This failure rate can be reset after repair or adjustment of an equiplet.

Equiplet agent

The equiplet agent is also implemented as a Jade agent and it is the interface to the underlying software and hardware. It depends on the front-end of the equiplet what modules are available. The equiplet agent is also the interface to the product agent. Both types of agents live in Jade containers and can communicate with each other. The communication between the product agents and the equiplet agents as well as other product agents is FIPA-based. The Jade platform is FIPA-compliant. For the implementation of the blackboard, Open BBS has been chosen. This Java-based blackboard was easy to integrate in the Jade environment; it was open-source and tests proved that it performed well enough for our grid.

Interaction between the agents

The webinterface will deliver a JSON-object describing the product. This is translated to product steps by the product agent. The product agent will parse the JSON information and generate a step for every part of the JSON object that actually changes the product. The webinterface takes care of deletion and retries, so the JSON object itself should not be adapted by the product agent.

For example consider a situation where three balls are placed in a crate at certain compartments. This will result in three similar production steps (pick-and-place) from the point of view of the product agent. Now the product agent will interact with the available equiplets agents. When a candidate is found that is capable to perform the pick and place step, the feasibility of this step with given parameters will be checked. In figure 6.11 the

FIPA interaction the product agent and equiplet agents is shown. For every

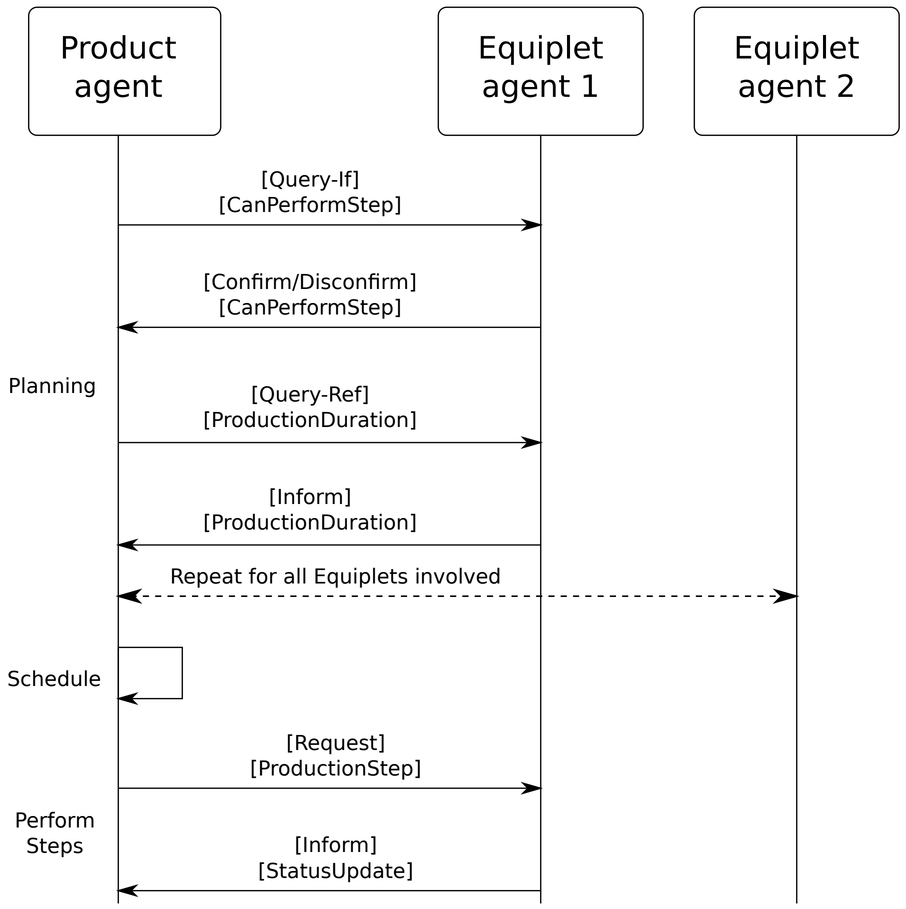

Product agent
Equiplet agent 1
Equiplet agent 2
[Query-If] [CanPerformStep]
[Confirm/Disconfirm] [CanPerformStep]
Planning
[Query-Ref] [ProductionDuration]
[Inform] [ProductionDuration]
Repeat for all Equiplets involved
Schedule
[Request] [ProductionStep]
Perform Steps
[Inform] [StatusUpdate]

Figure 6.11: Interaction between product agent and equiplet agents

interaction, a watchdog timer will take care of possible time-outs. This is necessary in an environment such as Jade where this asynchronous type of communication is applied. In case of a time-out a recovery will be done by repeating the part that was timed-out and if a failure is detected the error recovery mechanism will be triggered.

Even though this chapter focuses on the product agent, it might be a good idea to explain some details about the equiplet agent. More information about the equiplet internal architecture can be found in (Telgen et al., 2013). The equiplet agent will translate the production steps in front-end-specific sub-steps. A pick-and-place action is composed of movements and control of a vacuum pincer to pick the objects involved. The movements and commands are sent to the ROS-layer that will control the hardware and the commands are actually carried out by the connected hardware.

Error recovery

In this section the error recovery is discussed. Three types of errors are anticipated:

1. Equiplet crash: this is an unexpected hardware or software failure of an equiplet.
2. Equiplet shut-down: the equiplet is brought down or in a state where it is not capable to perform production steps.
3. Production step error: this means that the production step has been tried by the equiplet but it failed to produce the right result.

In the first situation, the product agent that has this equiplet in its production path, will discover that the equiplet agent is not responding or responding negative. As a result the product agent will clear the possible production steps on the equiplet blackboard BB-steps. This is the only situation where an product agent will adjust the information on this blackboard. This will prevent the scheduling of this equiplet by newly arriving product agents. However other product agents that already planned to use this equiplet will discover the same problem and react the same way. In the second situation, the equiplet agents itself will clear the equiplet blackboard and also mark the future planned steps on BB-planning as cancelled. It will also inform the product agents involved about this cancelling. These agents will reschedule their production. To prevent a burst of atomic rescheduling actions to occur at the same time step, two solutions are proposed:

1. This rescheduling is postponed to one step before the actual planning of the cancelled step. Normally the scheduling takes only a small part of a time step. This way the rescheduling action is smeared over the available time steps.
2. Another solution might be initiated by the equiplet agent itself. This agent will inform the product agents involved and also give a sequence number. The agent with the lowest sequence number will reschedule in the next coming timeslot, the agent with the next lowest number will skip one slot and use the next one and so on for all product agents involved. The advantage of this approach is that the rescheduling is done as soon as possible giving a bigger chance for success.

For the last situation, only two types of production step errors have been implemented yet. The first type is the recoverable error. In this situation the product has not been changed by the failing production step and the step could be retried, preferably at another equiplet. In the second case the product has been changed, but not according to the specified production step. This is considered to be an exception, the product will be removed from the grid (possibly for human inspection) and the product agent maker will get an error report. In both cases the error is also stored in a knowledge base, so other product agents with similar parameters could decide to avoid the error-prone equiplet in the future.

Reliability For almost every production system downtime is expensive. So now we should focus on the reliability of our model. In chapter 1 a list of possible hardware and sofwtare failures was already presented:

- Equiplet failure.
- Equiplet agent failure.
- Product agent failure.
- Supporting systems failure.
- Network failure.

Here some solutions are presented in more detail. Equiplet failure and equiplet agent failure can be overcome by equiplet replication. We have a grid of equiplets and in this grid we can allocate a spare equiplet to take over the task of the failing equiplet. The failing equiplet can be restarted and can again become a working member of the production grid. When an equiplet system fails during a production step, the product agent must recover from the failing step. This can be done by special recover equiplets that can be human-operated and complete a halfway-broken production step. This has been explained in the previous section.

Failure of a product agent can be taken care of by using a software replication of the product agent on the central server. This agent is updated after every production step and triggered by a watchdog timer in case it takes too long for the actual product agent to finish a production step. The replicated product agent will try to contact the actual product agent to see if it is still alive. If not, it will take over the product and try to recover the production state by using the afore mentioned recover equiplet.

The supporting systems can use standard failover techniques that are also used in other kind of critical server systems. Virtualization and replication play an important role in modern high availability systems and are nowadays a well understood and mature technique to assure uptime.

A network failure can be discovered by grid techniques like heartbeat and again watchdog timers. It depends on the actual situation how we should recover from such a failure. The communication used is at the network and transport layer based on TCP/IP resulting is a best effort network. When used on the same switch with probably the same VLAN, this type of communication has been proven to be quite reliable. VLAN is an acronym for Virtual Local Area Network. Modern switches are capable to configure subsets of their network connections to act as a LAN. Such a software configured LAN is called a VLAN (Tanenbaum and Wetherall, 2010).

6.1.3 Results and future work

The research done so far for this agent-based production system had several milestones. The first milestone was the proof of concept given by a simulation of the multiagent system as described in chapter 2. In that system the product agents planned their production path along equiplet agents that used timing delays to mimic the production steps. The equiplet agent was not combined with the equiplet hardware. The next milestone was the implementation of a reliable and fast scheduling algorithm as described in chapter 3. The latest milestone is described in this chapter where we made the two final steps. First the MAS with the ROS-based equiplet were integrated in the system, so the integration with real equiplet hardware has been accomplished. As a second step a web front-end has been built to specify the product to be produced. At this moment the given 2D examples can be executed on the three available equiplets. So the total chain from design to production is working. In figure 6.12 a design in the paint application of WIMP is made. In figure 6.13 the result of this product is shown. Though this example might look a bit disappointing because of its simplicity, it shows that the multiagent system is working to our expectations. The 3D example is already implemented at the MAS level and ROS level. The equiplet front-end to perform these steps is under development as a glue dispenser and an extra degree of freedom (rotation capability around the z-axis) of the pick and place robot is needed. However using a dummy equiplet (as in the earlier developed simulation) shows that the software is working to our expectations. This also includes the error recovery system.

6.1.4 Conclusions

In this first part of the current chapter an agile agent-based production system is presented. Before this system was built, a simulation system was developed and tested. A multiagent-based production scheduling has been

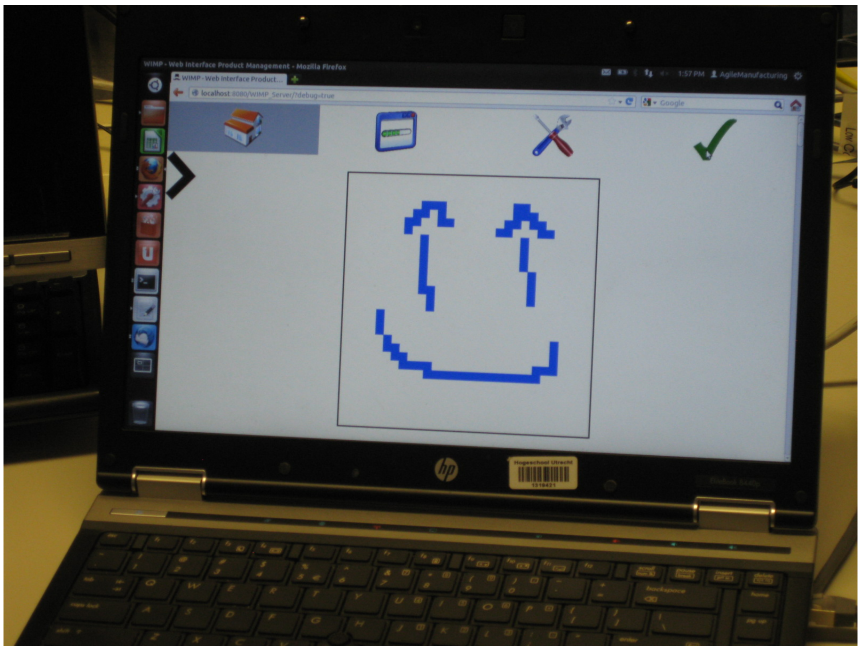


Figure 6.12: WIMP paint design

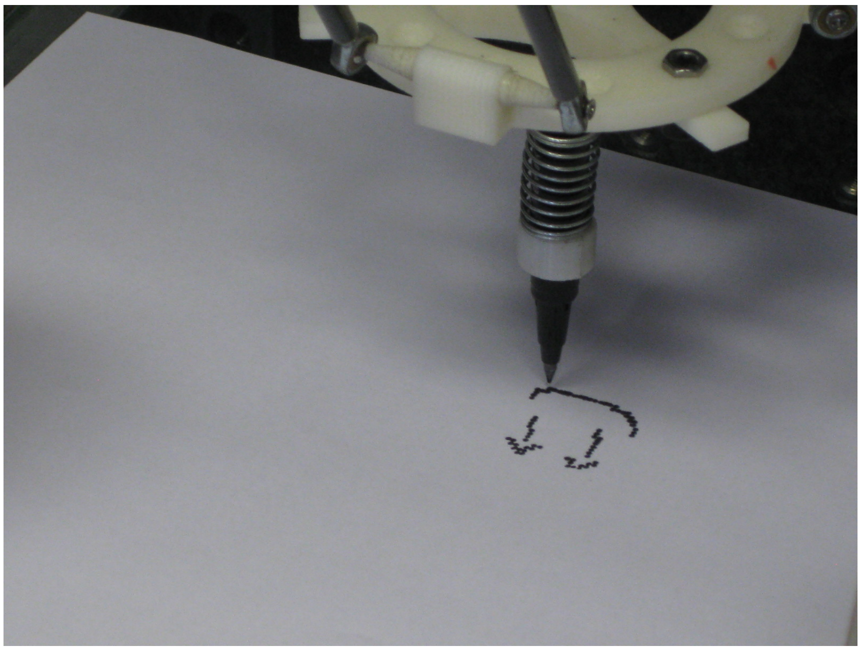


Figure 6.13: Resulting product on the equiplet

developed and tested. In this section we described a real production system that has been built as a proof of concept. All software used is based on open standards. Further research on the production of products with a higher complexity must be done, however the basic techniques for the implementation proved to work.

The grid is capable to produce several different products in parallel and every product has its own unique production log generated by the product agent. This product agent can play an important role in the other parts of the life-cycle of the product. When a product will be disassembled the product agent carries important information about the sub-parts of the product. This can be useful for recycling and reuse of sub-parts.

The production approach described here is also applicable to a hybrid system containing human actors as parts of the production system. The production steps for a certain product should be translated to human-readable instructions and humans replace the equiplet systems. In that model the equiplet agents carries out this translation so the MAS layer is still intact. This approach is useful in the situation where the production tasks are too complicated for an equiplet to be performed, but it can also help in the situation where a new equiplet front-end has to be developed. This approach will be discussed in the next part of this chapter.

6.2 Multiagent-based Agile Work Distribution

This section describes an agent-based and web-based system to instruct workers to build a product or to perform certain actions. The making of a product or work to be done for a task is conducted by a so-called product agent that knows what actions should be done. The work is collected and divided in smaller tasks to be done by workers. The workers advertise their capabilities and a product agent selects workers for a task and distributes the work among the workers. Both workers and tasks to be performed are represented by agents that interact in a multiagent system. An example implementation is also presented as a proof of concept.

6.2.1 Introduction

Today new ways of working are explored like working at home, distributed production of small quantities of products or manufacturing of user-specified products. This section describes an agent-based infrastructure that helps to support these new ways of working. Two types of agents play a major role in this system: product agents and worker agents. The work to be done or the product to be made is presented by the product agent to the worker agents that will instruct the workers according to their capabilities and preferences. Workers can publish in a global accessible data storage their possible actions and other preferences by using the aforementioned worker-agent. All communication is web-based so the only infrastructure needed is an internet connection. The product agent is guiding the production process or the work to be done and collects valuable information. In some cases this product agent could finally embed itself in the product to serve as a representative for the product in the Internet of Things.

The concept that will be explained in the next section resulted from previous research on agile production in a grid of production machines called equiplets (Puik and Moergestel, 2010). This approach turned out to be a good solution for certain situations. In the current case the same agent-based model for the infrastructure is used while the grid of production machines is replaced by a group of human workers.

In the next section the concept will be explained as well why an agent-based approach is appropriate and the advantages of this concept. The next section discusses models where this concept can be applied. The architecture and agent multiagent design is discussed next and as a proof of concept an implementation of a production model for an internet radio is presented. Future work, related work and a conclusion finalise the section.

6.2.2 Description of the concept

In the next parts of this section we will refer to the making of a product or work to be done as a task. A task consists of subtasks that will be referred to as steps. At one end of the system, an end-user or principal will enter a task to the system. This task will describe in detail what the end-user expects. It will be a list of steps possibly provided with parameters.

When the description is completed, the product agent will be created that will start to search for workers capable to perform the steps that are required to complete the task. This is done by a blackboard where the steps that can be performed by the workers are announced. Actually the worker agent will announce these steps on behalf of the human workers.

Agent-based system

As already mentioned, the system as proposed consists among other parts of two types of agents.

1. Product agents that describe what should be done and control the actual execution of the task, that might consists of subtasks or steps.

2. Worker agents, representing the humans that actually carry out the requested work.

There are several reason why this system is based on agent technology. In the first place it is a good representation of the common situation where people work together.

Generating a task consisting of steps and looking for work to be done are asynchronous events. That means that at any time a worker can look for work to be done while there may be no work available at that time. A worker agent can stay alert for newly arriving work to be done and inform the real worker when there is work to be done. This solves the problem of being available all the time, because the agents are there to alert a worker. The same is true the other way around for the situation where there is a task generated. The human that generated the task can quietly wait for the task to be done, without interfering with the system, because the product agent has taken over the responsibility to delegate the steps, that are needed for a certain task, to the workers. Both tasks and workers may be unique. Tasks are dynamic, workers can also vary in time. Because every task might be unique, it is a good idea to create a software entity for every task instead of trying to generate a general task software entity and adapt that to every single new task. Another advantage of using agents is the fact that a worker agent that is always representing the same worker, can adapt to special features or capabilities of this human worker. The infrastructure and software remains the same, but because the worker agents added more capabilities the system will become more powerful. Adding more workers means adding more worker agents without adapting the system as a whole. Agents could migrate to another plant or branch organisation that uses the same agent-based infrastructure. The specific features of the worker agent are carried along. The product agent can be a representative of a product in later phases of its life cycle.

Advantages of the proposed system

There are reasons why for some specific situations this approach might be very useful.

- It integrates agile task handling software with a standard human work-intensive environment.

- It scales well: a new worker introduces a new worker agent without the need to change other parts of the software.

- It also fits well in a distributed system approach where worker and people announcing the tasks are not at the same place.
- When a worker increases his set of production capabilities, the corresponding worker agent will announce this to the multiagent system. In this way the approach is agile in regards to the capabilities of the participating workers. A worker that learns a new skill can quickly apply this in the production environment.
- A log of production done by an individual worker is created and maintained by the worker agent, so the worker can be paid by the amount of work done and not by the amount of time spent.
- This approach fits well in a distributed environment where workers at home are supported by and interacting with the multiagent system.
- It is easy to enhance this system with individual help to workers. The worker agent can be offered the possibility to show an instruction film to explain details of a certain production step. This is also a nice feature for workers that do their work remotely (i.e. at home).
- An individual training facility is also easy to implement. This way a worker can expand his set of production steps, making himself more competent and thus useful for the production environment. This will adjust the production system as a whole to the capabilities and the aspirations of the individual workers.
- Workers can make individual adjustments to the work time planning. This will also be useful in case of remote workers.

6.2.3 Application models

The model presented here is not applicable to all possible situations, but there are application areas where this concept can be applied successfully. Some examples are explained in the next subsections.

Small batches or single products

The system proposed here has already been introduced in a so called production grid. In this grid the workers are represented by machines that are instructed to perform certain production steps. This production system was designed for affordable small-scale production. Instead of machines also human workers can be instructed, having the possibility to construct at a low volume products of high complexity, depending on the skills of the human workers involved.

Step by step guidance

Because of the fact that for every step an instruction is generated that can be illustrated by a picture or a video, this method is useful in situations where workers are still in the situation of being trained or maybe unsure about how to perform a certain step. Instead of every time calling for assistance, the worker agent itself can assist the worker. In this situation a knowledge base of frequently asked questions and solutions can enhance the supporting infrastructure system on the fly.

Products according to user specification

User requirements for products are difficult to combine with mass production. The system described in the current section opens the possibilities to produce tailor-made products according to user requirements. This can be done by letting select the end-user a set of properties of the product to be made. During production, these requirements are presented at the appropriate time to the worker. In section 6.2.6 where the case study implementation is discussed, this model is taken as an example.

Office-like work done by home-workers

This situation is already in use all over the world, though the infrastructure is not agent-based as described in the current section. The reason why this situation is popular is because of the fact that the logistics for transferring physical components is not needed. Having a connection over the internet is a sufficient base for this model. The proposed agent-based solutions offers the advantages already mentioned in previous sections. Possible applications of the concept are: software development, web design, translation work etcetera.

Educational system

The model can also be applied to educational situations. In this case the instructor tries to train the students by sending tasks to be completed. It can also be applied to test the competences of a student by letting him perform a task.

6.2.4 Architecture

In this section a description of the system architecture as well as the software that has been used, will be presented. Figure 6.14 shown the global system

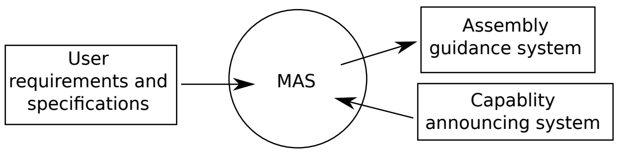

User requirements and specifications
MAS
Assembly guidance system
Capablity announcing system

Figure 6.14: Basis system setup

architecture. On both sides of figure 6.14 a human or humans are situated. The user on the left side creates the tasks, users on the right side will do the work. To get to the functional specification of the system, the services that should be offered by the system must be specified. The infrastructure offers several services:

- It brings together the party asking for certain services with the party offering certain services.
- It will check if the work to be done and the tasks to be completed are feasible.
- It will do all the planning and scheduling.
- It will inform both sides of the humans involved about the status of the process involved.
- It will keep track of the preferences of the humans involved.
- It will collect information during the time the work is done. Using these data a knowledge base of the task being performed is built.
- It will log the amount of work done by the human worker represented by the worker agent.

These functional requirements will actually be carried out by the agents involved. A more detailed view on the heart of the system is presented in figure 6.15.

In this figure the MAS contains a product agent and a worker agent. When a task is requested, a product agent is created that is responsible for the that single task to be completed. As explained before, a task might be the creation of a single product, a small batch of products or just work to be done. A task contains subtasks or steps. A step is an action plus zero,

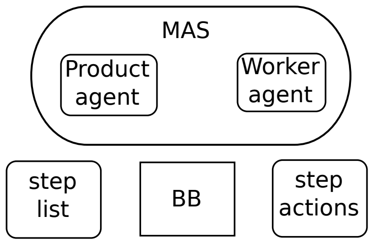


Figure 6.15: More details of the MAS-related system

one or more objects. User specification results in a list of steps to be done to complete a task. This step list belongs to the knowledge base of the product agent. The worker agent on the other hand controls the worker capabilities resulting is a description of step actions. This list of actions is generated in cooperation with the worker. The system maintains a list of needed actions for tasks to be performed in a certain context and inquires a worker about his possibility to perform that action. For example if the system is used in an environment where documents should be translated to other languages, a worker could tell the system, by using his web interface, that he can translate English to Dutch. In the action list the line: *translate English to Dutch* is entered. In our implementation, XML has been used to construct the action list. Of course this could be enhanced by extra information like, type of document (fiction, technical, ...) and level of expertise.

To bring the agents together, blackboards are used. There are two blackboards involved: one blackboard for publishing the step actions that can be performed by the worker agents and a blackboard for planning and reservation.

In figure 6.16 the layered software architecture and interaction between the components is given. Only one product agent and one worker agent is depicted.

The goal of the product agent is the completion of a task. To achieve this goal it will first plan and schedule the task in its role as planner. In this role it will try to plan a schedule based on the information on the blackboard put there by the worker agents. If the scheduling is successful it will adjust the reservation information on the planning blackboard and change its role to task controller. In case the scheduling fails it will report this to its maker and cease to exist.

For the MAS layer Jade (Bordini et al., 2005) was used as a platform. The reasons for choosing Jade are already mentioned in other parts of this thesis.

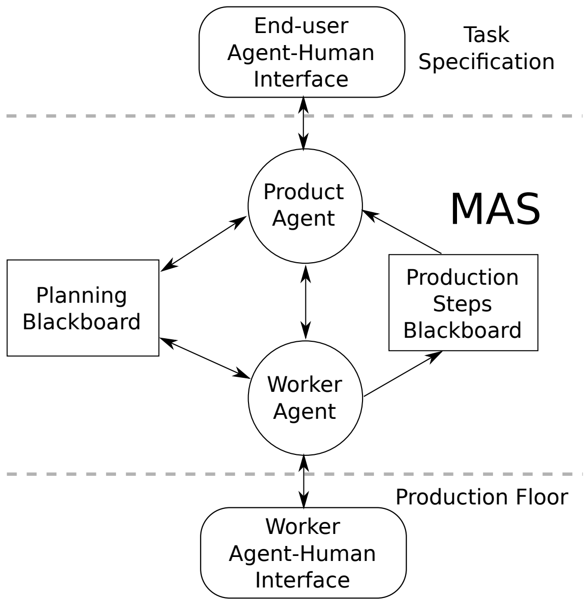

End-user Agent-Human Interface
Task Specification
Product Agent
MAS
Planning Blackboard
Production Steps Blackboard
Worker Agent
Production Floor
Worker Agent-Human Interface

Figure 6.16: Layered architecture

6.2.5 Agents roles and responsibilities

In this section the roles and responsibilities of the two types of agents are discussed in more detail.

Worker agent roles

The role of the worker agent is being the software representative of a human worker. The worker agent is unique for every worker and will be created when a worker is introduced to the system for the first time. It will be connected to that worker as long as this worker is a member that participates. A worker agent can be pro-active if the worker is not available at a certain moment. This is because of the fact that the worker informed the worker agent about its availability, preferences and capabilities. So the worker agent is still an active participant of the multiagent system. The responsibilities of the worker agent are:

- In the first phase after creation, the worker agent collects information about the worker involved. It will be tied to this worker as long as this worker stays available for tasks to be performed.
- Announcing the capabilities, by publishing the possible production steps.
- Announcing the availability.

- Updating capabilities as requested by the worker.
- Updating availability as needed.
- Confirming or denying feasibility of steps when inquired.
- Logging the work done by the worker.

Product agent roles

This agent is responsible for the completion of a task. Its goal is a completed task. To reach that goal, the product agent has several roles.

Planning and scheduling The task can only be completed by workers so a product agent will make a selection of worker agents based on the production step or steps that must be performed to complete a task. The selection of workers is done in four steps or phases:

1. Select workers for a certain production step.
2. Ask worker agent if the production step with given parameters are feasible. When a negative response is received by the product agent it will discard the worker agent.
3. Depending on the situation or model, optimize task execution.
4. Schedule production.

A product agent chooses a worker based on the set of production steps published by the worker.

Task controller When the planning and scheduling is completed, the product agent changes its role to task controller. This means:

- Collecting logging information received from the worker agent.
- Informing the human who ordered the task about the progress.
- Reporting errors.

Roles after completion When the task is finished, the product agent could cease to exist because it has reached its goal. However in case the task is the production of a product, there are a lot of possible roles and new goals the product agent can have during the life cycle of the product as explained in chapter 5.

6.2.6 Case: Internet radio

To implement the concepts discussed so far, a system has been built to construct a device for playing audio streams from the internet: a so-called internet radio. The hardware of the radio is based on a small and cheap Linux-based computer system, the Raspberry-Pi. Several casings are available and also the audio system has several options. In this example the possibilities are still limited, but the possibilities can be easily expanded without changing the concept. The whole assembly is considered to be one single sequence of steps. These steps are performed by one worker.

The scheduling and planning is already described and implemented is a similar environment (Moergestel et al., 2012), where instead of workers, production machines were used. This time the production machine is replaced by a human worker that interacts with this system by a web interface. In this case only one worker will be selected. The corresponding worker agent will receive the user preferences from the product agent.

Description

The user selects among other features, the casing of the radio by material (wood, metal, plastic), colour (the palette of options depends on the choice of the material) and the audio-subsystem (speakers, earphone, high-end audio).

Web interface for user requirements

The web interface is actually Java-based to make the connection with the Java-based agent system easier. The browser at the user-side should support HTML 5 and also JavaScript to make the page interactive and dynamic. The user selects the options by clicking and finally submits his request to assemble the radio to the system. At that moment a product agent is created. This agent also gets an XML-file from the web server that incorporates the user selections. XML has been used, because both JSON and XML were candidates for this implementation and in the previous case JSON has already been used. This way a proof might be given that both standards can be applied to this situation. Part of this file looks like:

```
<parts>
<part id="" name="colour">
<property id="value">
<value>black</value>
</property>
```

```
</part>
<part id="" name="material">
<property id="value">
<value>wood</value>
</property>
</part>
...
</parts>
```

In this XML-file the user preferences for material and colour are shown. After submitting his preferences, the user will get an unique ID that can be used to connect to the product agent in a later stadium. This ID can also be used to connect to the product when the product is made and in its use-phase.

Converting steps to required actions

The worker agents should command the human worker to actually carry out the requested steps. The worker agent received the user preferences from the product agent. This is slightly different from the situation where a worker executes only one step. In that situation the worker agent receives the parameters for that step. In this case however, the task is performed by only one worker and for efficiency reasons the whole set of preferences are handed over in one transaction. This is done by a user selection file, containing the user preferences in XML-format.

Another file that is also based on XML, describes what actions to perform for a given step. This is the parts/actions XML-file. This information is combined with a standard template XML-file that should be used for all internet radios to be built. Combining the three aforementioned XML-files results in the builder XML-file containing the actions, part positions in the warehouse and possibly training and demo information for the worker to view. Figure 6.17 shows this combination of the XML-files. A simplified version of this builder XML-file looks like

```
<builder>
<parts>
<part>
</part>

</parts>
<actions>
<action order="0">
  Take the [type] part from [location]
```

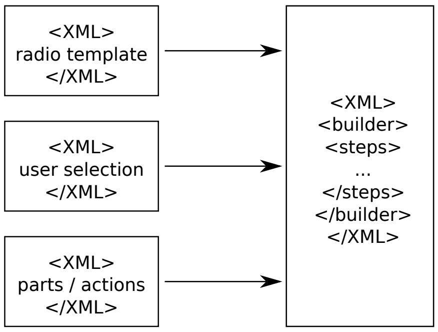


Figure 6.17: Basis system setup

```
  in the warehouse
</action>
<action order="1">
  Place the [type] part in the [casing]
  casing
</action>
<action order="2">
  Paint the [casing] [color]
</action>
</actions>
</builder>
```

When using this file on the webserver side of the worker, the worker finally gets instructions in his browser like:

```
Take the controller-board part from
shelve 2A in the warehouse
Place the controller-board part in the
wooden case
Paint the wooden case black
```

Web interface for worker

The worker is instructed by the worker agent using the web interface. After logging in, the worker will be connected to its own representative, the worker agent associated with the worker. This agent will inform the worker of the work to be done. In several ways the web interface can be used to instruct the worker. When a step is finished the worker can add additional comments that will be collected by the worker agent and next communicated to the product agent as logging information for the making of a product.

The same web interface is also used by the worker to announce his preferences. In this case it will announce his availability for work.

Embedding the product agent

Before the way of embedding the product agent for this particular case is explained, first an overview of possibilities for embedding are presented.

Pull techniques A product can pull its agent to the onboard environment at the moment is is completed and tested or at the moment it is switched on by the user. In both situations there should be a communication channel available. Another problem to be solved is: how does the device know its particular product agent? In case of an ethernet attached device, during construction the product agent might be aware of the unique ethernet address of the device and at the moment the connection is made, the product agent knows from the MAC-address that its device in requesting the agent or the agent information. This method can only be used if the device is connected to the LAN segment that is shared with the hardware the agent is running on. If the device is operating at a remote internet node, it needs a special identifier to get its particular agent. To prevent some security issues, encryption techniques should be used to prevent illegally pulling the agent or its information. Software installation programs can also be used to install the product agent on the device.

Push techniques In case of a push technique, the agent is sent to the device by the production system itself. This can be done in an active or a passive way. In the passive way, the agent is embedded in the software that is to be downloaded into the device. By installing the firmware the agent will hitch hike with it into the device. An active push techniques requires the device to be switched on and have a connection with the production environment. The agent can migrate from the production environment to the device if the appropriate provisions are made. By this is meant, that there should be an environment where the product agent can live.

Example applied to the case Returning to the case presented here, the last phase of the construction will be the transfer of the product agent into the product itself. The Jade-platform has the possibility to transfer an agent to another container, however, transferring to another platform is a different story because this means a new environment. To accomplish this step JADE Inter-platform Mobility Service (JIPMS) has been used. JIMPS makes it possible for a Jade agent to migrate to another jade environment. So in this case the product itself should have its own Jade environment. This has been done by downloading the required software to the Raspberry Pi during the construction of the internet radio. So the product agent thus far collected the assembly information from the designer web interface, transferred the information to the worker agent, monitored and collected construction data from the worker agent during assembly and now transfers itself using a push technique to the final product. In the next four screenshots, the creation and migration of agents is visualised. In picture 6.18

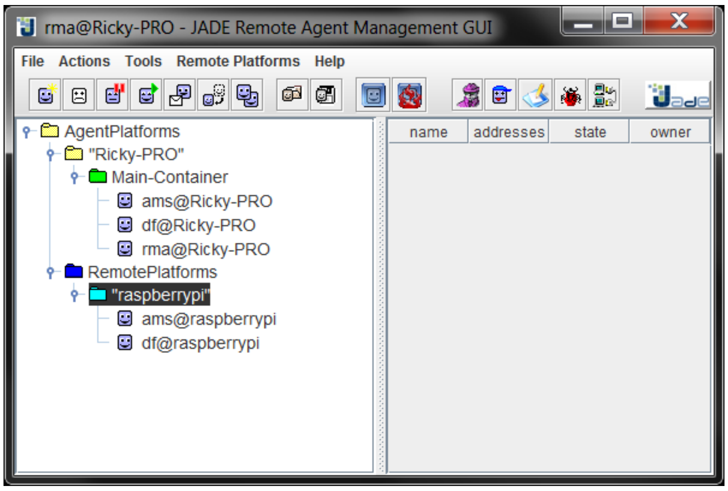


Figure 6.18: Start situation with two separte agent containers

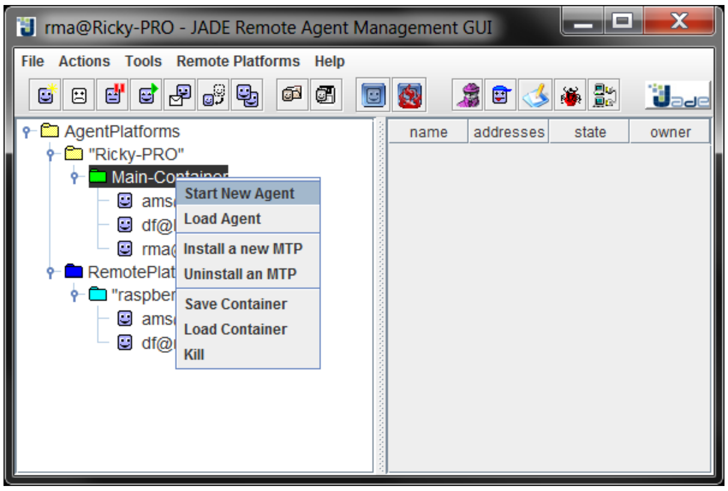
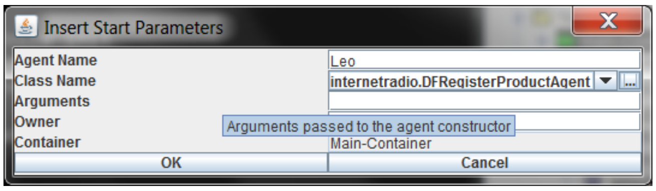


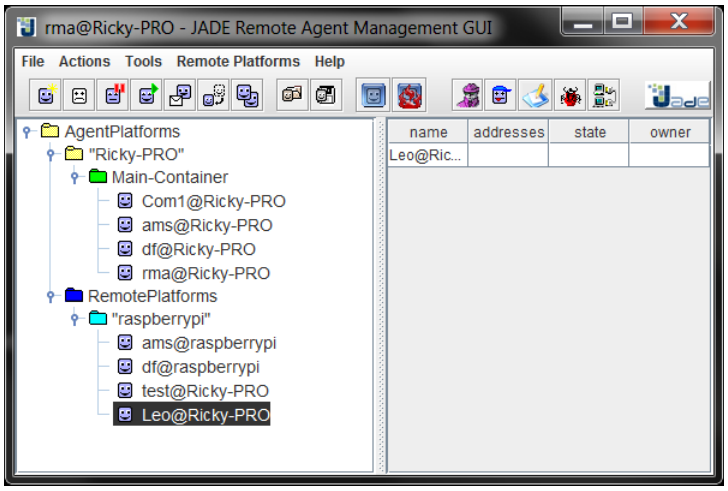
Figure 6.19: Creating an agent in the main container


Insert Start Parameters

Agent Name	Leo
Class Name	internetradio.DFRegisterProductAgent
Arguments	
Owner	Arguments passed to the agent constructor
Container	Main-Container

OK Cancel

Figure 6.20: Specifying agent properties


rma@Ricky-PRO - JADE Remote Agent Management GUI

File Actions Tools Remote Platforms Help

```
AgentPlatforms
  "Ricky-PRO"
    Main-Container
      Com1@Ricky-PRO
      ams@Ricky-PRO
      df@Ricky-PRO
      rma@Ricky-PRO
  RemotePlatforms
    "raspberrypi"
      ams@raspberrypi
      df@raspberrypi
      test@Ricky-PRO
      Leo@Ricky-PRO
```

name	addresses	state	owner
Leo@Ric...			

Figure 6.21: Final result of two migrated agents

Figure 6.22 shows the situation where the product agent has embedded itself in the product and can offer a user manual to the end-user as well as other information. The same web interface can also be used to let the device communicate with other devices and servers so the device participates in the Internet of Things (IoT).

6.2.7 Results and future work

With the case study we implemented a production system that has the proposed multiagent-based core at its basis. The end-user can specify an internet radio with specific features and a worker will be instructed to actually build the system as required. Finally the product agent will embed itself in the product to continue its life in the remaining parts of the life-cycle of the product. This agent-based work distribution or task execution approach uses a similar model as the equiplet-based agile manufacturing system described in the first part of this section. In the case study we opted for the production of a device with a lot of user specified features. So the total chain from design to production is shown. Future research will focus on enhancing the web

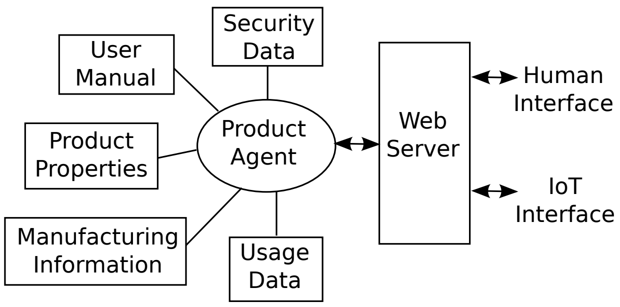


Figure 6.22: More detailed system

interface to the worker side with multimedia support and implementing the model in other environments.

6.2.8 Conclusions

The production approach described is indeed applicable to a hybrid system containing human actors as parts of the production system. The production steps for a certain product should be translated to human readable instructions and humans replace the equiplet systems presented in the first part of this chapter. In that model the equiplet agents carries out this translation so the MAS layer is still intact. This approach is useful in the situation where the production tasks are too complicated for an equiplet to be performed, but it can also help in the situation where a new equiplet front-end has to be developed.

Both approaches presented in this chapter are implementations of what in terms of cloud computing could be characterised as *Manufacturing as a Service* (MaaS).

6.3 Related work

This section discusses related work for the equiplet-based manufacturing system and the the work distribution system. The section about the work distribution system has relationships with both agent human interaction as well as agent-based production systems. An important aspect in that field is the agent-human interaction. In research papers, this aspect is commonly referred to with the term *mixed-initiative*. An overview of early research in this field can be found in [(Hearst, 1999)]. Another important paper in this field is (Shneiderman and Maes, 1997) . More resent research includes interfacing agents and humans using natural language. In (Vergunst, 2011) a system for robust task-oriented dialogues is presented. This work focusses on human-machine dialogues. In our work the interaction is still simple, but the results of the research done by Vergunst could be an interesting future enhancement.

A nice overview of agent-controlled manufacturing is presented by (Shen et al., 2006). In this paper all kinds of aspects in manufacturing where agents could possibly play a role are discussed among with references to important publications about that specific aspect. The work of Monostori and others [(Monostori et al., 2006)] is also an overview of using agents in manufacturing. This work explains the agent paradigm and translates this to several fields in the production industry. The proposed solutions are mostly apt to an existing manufacturing infrastructure and are based on quite a number of different agents, each having a specific role. Our concept is at an abstract level based on the difference between the *what to do* and *how to do* concept that is the basis of every production process.

Bussman and Jennings [(Bussmann et al., 2004)] used an approach that compares to our approach regarding the fact that the execution of tasks is controlled by a small set of agent-types. The system they describe introduced three types of agents, a workpiece agent, a machine agent and a switch agent. There are however important differences to our approach:

- The production system is a production line with redundant production machinery and focuses on production availability and a minimum of downtime in the production process.

- The roles of the agents in this approach are different from our approach. The workpiece agent sends an invitation to bid for it current task to all machine agents. The machine agents issue bids to the workpiece agent. The workpiece agent chooses the best bid or tries again. In our system the negotiating is between the product agents.

- They use a special infrastructure for the logistic subsystem, controlled by so called switch agents.

- This design does not include a hybrid system where human workers are involved.

Other authors focus on using agent technology as a solution to a specific problem in a production environment. The work of Xiang and Lee [(Xiang and Lee, 2008)] presents a scheduling multiagent-based solution using swarm intelligence. This work uses negotiating between job-agents and machine-agents for equal distribution of tasks among machines. In our approach the negotiating is between product agents and load balancing is possible by encouraging product agents to use workers or equiplets with a low load or workers that are eager to work. We have developed a paradigm equiplet-based agile manufacturing and for work distribution among workers. Both solutions are based on agent technology. Both models are based on two types of agents and focus on agile and new ways of manufacturing or working. The product agent involved can also play an important role in the life-cycle of the product.

6.4 Summary

This chapter showed two implementations of the agent-based manufacturing system. The architecture of both implementations is similar, but the setting is quite different. This shows that the architecture and the concept of the MAS-based system is a generic approach and can be used in many situations. Also the benefits of the concept of the product agent are applicable in both situations described here. For the production description at the agent level in the first part JSON has been used, while in the second part XML was chosen. Both approaches are suitable to the system proposed.

Chapter 7

Conclusion and future work

What has been presented in the previous chapters is work that contributes to a long term project. The aim of the project is to develop an new way of agile enduser-driven manufacturing.

7.1 Review of the work done

What has been presented is this thesis can be summarized as follows:

- The concept of product agent and equiplet agent, where the product agent is in this case the carrier of the *DNA* of a single product.
- A planning and scheduling system that showed that in our case a load of 80% of the grid is feasible. To achieve higher loads, the temporary storage is needed. The amount of interagent communication will also strongly increase at higher loads.
- The extension of the product agent to the whole life cycle, making the product agent a software representation of a product from manufacturing to recycling.
- A proposal for the production system and the internal transport within the grid.

The role of the product agent in the life cycle should also be considered from the aspect of security and privacy. As often is the case in some respects it can be very helpful, but from another point of view it be be a threat for personal privacy. Mechanisms to preserve privacy as needed should be used in the implementation.

The work on the equiplet agent and its internal architecture has been published in several publications, but was not the primary focus of this thesis. There are also many publications about grid production is general.

7.2 Answers to research questions

RQ1 How should we actually build these agents? What platform? Using an existing platform? How do the agents communicate?

The selection of the platform and the communication has been the topic of chapter 2. A simulation of a multiagent implementation answered the questions of what platform to use, how the agents communicate and how the agents could be implemented. Summarized: the Jade platform is used for the MAS. FIPA-based communication is provided by this platform. An existing platform has been chosen, because of the fact that it fits our needs to build a system as a proof of concept. The FIPA-based communication system supports the inclusion of messages in XML format as well as other formats. XML has been used in the first implementation to specify production steps.

RQ2 What planning and scheduling system should be used? The planning and scheduling topic was discussed in chapter 3. Here a planning and scheduling system that fits the needs for the manufacturing grid is presented with results from several simulations. This planning and scheduling mechanism is also used in the final implementation discussed in chapter 6. The planning results in a limited set of possible paths along the equiplets. The best path will be chosen and then the production steps will be scheduled. This scheduling is based on realtime scheduling solutions. Simulations showed that Earliest Deadline First turned out to be a good choice. A special implementation called weak EDF is proposed where the interagent communication is only used in case the product agent cannot schedule all its steps before the deadline by itself. This weak EDF introduced much less interagent communication overhead.

RQ3 How should the system be set up? How to transport the products during manufacturing?

The system setup and considerations about the transport system was the subject of chapter 4. The effect of the position of frontends in the grid has been investigated as well as possibilities for agile transport. Only when certain patterns in the production paths exist, the position of the frontends in the grid becomes important. For Agile transport the use of AGVs has been proposed. Using conveyor belts is complicated in a real grid topology and changing to a line topology conflicts with the agile manufacturing paradigm.

RQ4 What are the possibilities and roles of the product agent when the product is finished? What are the advantages to keep it alive? How to tie it to the product? What other roles can it play?

An important question was what to do when the product is made. Most discussions about agent technology in manufacturing do not consider using agent technology in the other parts of the life cycle of a product. Several cases are discussed in chapter 5 and these concepts are a promising field for further research. Embedding a product agent that contains valuable production data, can make the product agent a good candidate to represent the product in the Internet of Things. Many possibilities are proposed in chapter 5. A nice application is component exchange between broken products, thus enlarging the average lifetime of a product. The best solution to tie an agent to a product is to embed it in the product itself.

RQ5 At what point are humans involved in this manufacturing system? Humans can be involved in the manufacturing system introduced in this thesis in two ways. First a human can design a product to be made using a webbrowser running the WIMP software. WIMP stands for Web Interface Managing Production and can be used to compose or design a product to be made and WIMP also visualises the manufacturing progress.

Another way that humans can be involved is when equiplets are replaced by human workers. This is described in chapter 6 as an agent-based work distribution system showing that the production paradigm can be used in several situations. A hybrid approach where part of the work is done by humans and part of the work is done by equiplets is also a possibility.

RQ6 How to recover from errors and guarantee a reliable system? In chapter 6 mechanisms for error recovery are discussed. These mechanisms are mostly based on replication. Fail-over techniques will make the hardware much more reliable. Product agents can also rate the performance of equiplets, making it less likely that a product agent will chose an equiplet that is badly performing for steps with certain parameters.

The main question or problem statement was: how should the agent based agile manufacturing system be implemented and what could be the benefit to the whole life cycle of a product. Chapter 6 shows a prototype of a realisation. In this chapter the paradigm is also extended to a human worker environment, where the same approach can be used to distribute work. All aspects coming from questions previously answered in this section are combined in the final implementations. Chapter 5 describes the benefits to the whole life cycle of a product.

7.3 Conclusion

When we compare our approach with the existing solutions for production automating we have these advantages:

1. Scalability. When the production capacity is inadequate, one can simply add more equiplets and to a certain limit, the software adapts automatically.

2. MES and SCADA functionality is available (at least what is needed in the proposed paradigm).

3. Agile. The system can adapt quickly to new situations and production requirements.

4. Testing the system to prove its reliability can be done in a scaled-down situation.

5. The production grid can be working on all kinds of different products in parallel. The size of a production batch is not important.

6. Robust. There is no single point of failure and temporary problems with one ore more equiplets or product agents do not have a severe impact on the production process as a whole.

7. We can use the information of the grid to analyse the production system as a whole and to discover bottlenecks in the production. We can remove these bottlenecks to streamline and optimize the production process. If the production dynamically varies we can continually adapt the grid to optimize the production process.

8. A short transit time from research and development to the production floor. When a product is under development, we propose the capabilities of the equiplets. The needed frontends are formulated and eventually built. Such a frontend can be introduced on the production floor and the product can be built. When we need a lot of products of this new type we can replicate the needed equiplets by constructing multiple frontends.

9. The grid as a whole can inform us if it can make a certain product, based on the needed steps for this product. It can also report what production steps cannot be done yet for this product to complete, thus gaining a clue to expand the production capabilities of the grid.

10. A web-based user interface has been built to enable the user to specify a product to it needs. This interface can also be used to inform the end-user of the progress of the manufacturing process. This way *Manufacturing as a Service* (MaaS) can be realised.

7.4 Future developments

At this moment a proof of concept is given, but a lot of work needs to be done to make this concept acceptable for the industry. The transport system should be realised as well as more equiplets with a wider variety of capabilities. In Figure 7.1 an impression of the AGV on its way between two equiplets is given. The case with the discovery robot was the basis for the AGV. However, this discovery robot had an expensive laser range-finder and its functionality surpasses what is actually needed in our transport system. A cheap and reliable solution is now the topic of research. When

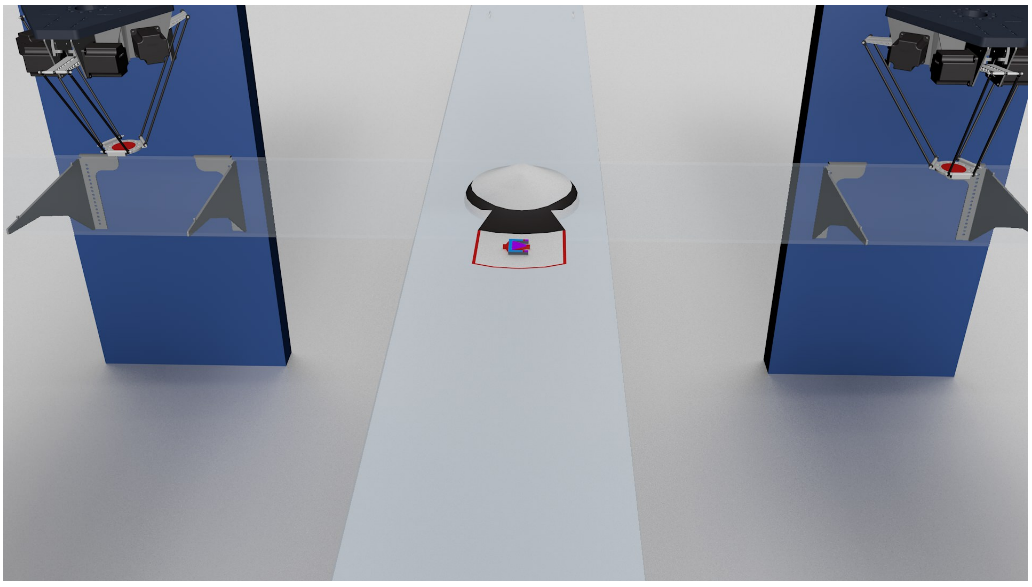


Figure 7.1: AGV and two equiplets

more equiplet capabilities become available, the web interface should also offer more possibilities to design a product.

The possibilities for reuse of parts and material should also be further developed. When the production steps are available at the time of disassembling a product, this could be helpful to decide how a product should be taken apart by investigating if and how a production step could be reversed. Standards should be developed to describe a product so parts can be identified and reused. This will help to realise zero waste

Appendix A

Monte Carlo Method

In this thesis at several places Monte Carlo techniques are used. Here some basic concepts of the method will be explained. Only some examples used in this the thesis are explained here. The Monte Carlo Method is based on using random numbers to get to a certain result. Its most well known application domain is simulations, where events with a probability distribution occur.

TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
© 2001 United Feature Syndicate, Inc.

Figure A.1: Random number generator

Three possible application of the method will be described.

1. Alternative calculation of a result.
2. Generation of data sets for testing or simulations.
3. Generating events conform a certain probability distribution.

A.1 Alternative for calculation

An example of this can be found in Moergestel (2009). Here π is calculated using random numbers. In Figure A.2 a unit square with a quarter of a circle is displayed. By generating random pairs (x, y) where $x \in [0, 1] \wedge y \in [0, 1]$ it is easy to detemine (using Pythagoras) if x, y is within the surface of the quarter circle. By using the fact that the surface of the quarter circle is $\pi/4$.

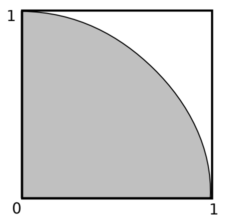


Figure A.2: Calculating π

The chance that (x, y) is within the quarter circle is also $\pi/4$, because the surface of the square is 1. Now, a huge number of random pairs is generated and for this set the amount within the quarter circle is divided by the total amount of random pairs. This results for 10000000 pairs in π being equal to 3.141550 (instead of 3.141593 for a mathematical calculation).

A.2 Generating data sets

In this thesis the planning and scheduling simulations were driven by datasets that had been generated using random generators. To check if the generated sets are according to the expectation, a plot of values can be generated. For our purposes this is sufficient. For example, a test has been generated where all products agents are choosing at random one of the 11 available equiplets. For a large number of products (42000), the distribution among the equiplets should be flat. In Figure A.3, the resulting distribution is shown.

A.3 Generating a set according to a distribution

The last application that will be described here is generating datasets according to a certain given distribution. In this case we use an approach that compares to the earlier given circle example. We surround the give distribution by a bounding box and generate random pairs (x, y) within the surface of the bounding box. Next we check if the pair is within the surface of the given distribution and if it is, the pair is used in the set, otherwise it will be rejected. The result for a set data having a Gaussian distribution is shown in figure A.4.

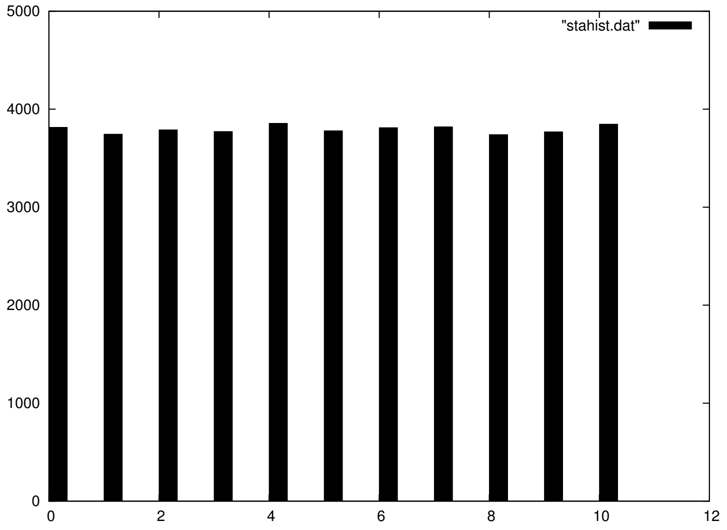


Figure A.3: Load distribution among 11 equiplets

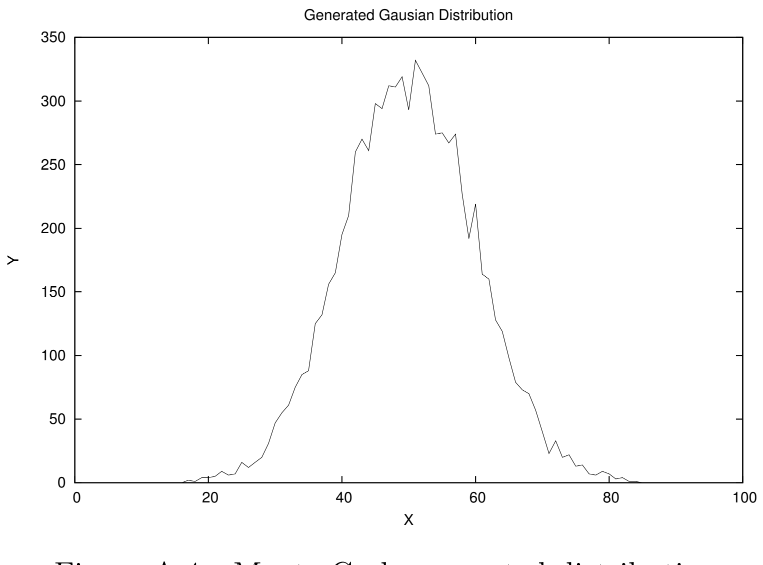


Figure A.4: Monte Carlo generated distribution

Bibliography

S. Abras, S. Ploix, S. Pesty, and M. Jacomino. A multi-agent home automation system for power management. *Proceedings of the Third International Conference in Control, Automation, and Robotics, ICINCO 2006*, pages 3–8, 2006.

A. Addis and G. G. Armano. Domobuilder: A multiagent architecture for home automation. *Proceedings of the 11th Workshop Dagli Oggetti Agli Agenti (WOA 2010)*, 621 of CEUR Workshop Proceedings, September 2010.

D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *OSRA Journal of Computing*, 3(2):149–156, 1991.

W.A. Arbaugh, D.J. Farber, and J.W. Smith. A secure and reliable bootstrap architecture. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 65–71, 1997.

K. Ashton. That 'the internet of things' thing. *RFID Journal*, (22 july), 2009.

J. Axelson. *USB Complete: Everything You Need to Develop Custom USB Peripherals*. Lakeview Research, 2nd edition, 2001.

P. Baronti, P. Pillai, V.W.C. Chook, S. Chessa, A. Gotta, and Y. Fun HU. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications*, 30(7):1655–1695, 2007.

F. Bastani. Keynote presentation at isads 2013: Internet of things. *Proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2013) Mexico City*, 2013.

F. Bellifemine, Caire G., and D. Greenwood. *Developing multi-agent systems with Jade*. John Wiley & Sons Ltd., 2007.

A. Bensmaine, M. Dahane, and L. Benyoucef. A simulation-based genetic algorithm approach for process plans selection in uncertain reconfigurable environment. *IFAC Conference on Manufacturing Modelling, Management and Control*, pages 2002–2007, 2013.

L. Blazovics, C. Varga, K. Csorba, M. Fehr, B. Forstner, and H. Charaf. Vision based area discovery with swarm robots. *Second Eastern European Regional Conference on the Engineering of Computer Based Systems, ecbs-eerc*, pages 149–150, 2011.

Bloomberg. *Increase in prices of various resources over a period of 10 years, www.bloomberg.com.* 2009.

C.A.M. Bolzani and M.L. Netto. The engineering of micro agents in smart environments. *International Journal of Knowledge Based Intelligent Engineering Systems 13, no. 1*, pages 31–38, March 2009.

N.R. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni. *Multi-Agent Programming.* Springer, 2005.

R.H. Bordini, L. Braubach, M. Dastani, A.E.F. Seghrouchni, and J.J.. Gomez-Sanz. *A survey of programming languages and platforms for multi-agent systems.* The Free Library, 2006.

M.E. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Mass, 1987.

L. Braubach, A. Pohkahr, and W. Lamersdorf. Jadex: A short overview. *Proceedings of the Main Conference Net.ObjectDays*, 2004.

M. Brink, A.J. Jessurun, F. Franchimon, and J.E.M.H. van Bronswijk. An open agent-based home automation system. *Gerontechnology 2008;7(2)*, 2008.

M. Burgess. Cfengine as a component of computer immune-systems,. *Proceedings of the Norwegian Informatics Conference*, 1998.

M. Burgess, H. Hagerud, S. Straumnes, and T. Reitan. Measuring system normality. *ACM Transactions on Computer Systems (TOCS) Volume 20 Issue 2*, pages 125–160, 2002.

B. Burmeister, A. Haddadi, and G. Matylis. Application of multi-agent systems in trafc and transportation. *IEEE Proceedings on Software Engineering 144 (1)*, page 5160, 1997.

S. Bussmann and D.C. McFarlane. Rationales for holonic manufacturing control. *Proceedings of the second international workshop on intelligent manufacturing systems*, pages 177–184, 1999.

S. Bussmann, N.R. Jennings, and M. Wooldridge. *Multiagent Systems for Manufacturing Control*. Springer-Verlag, Berlin Heidelberg, 2004.

D. Cohen. Earth's natural wealth: an audit. *New Scientist*, (2605), 2007.

B.W. Coile and J.A. Jordan. Method and apparatus for transparently providing a failover network device, August 22 2000. URL `http://www.google.com/patents/US6108300`. US Patent 6,108,300.

G. Conte, G. Morganti, A. A. Perdon, and D. Scaradozzi. Multi-agent system theory for resource management in home automation systems. *Journal of physical agents, Special session on practical applications of agents and multiagent systems*, 3, NO 2, 2009.

D.D. Corkill, K.Q. Gallagher, and P.M. Johnson. Achieving flexibility, efficiency, and generality in blackboard architectures. *Proceedings of the National Conference on Artificial Intelligence*, pages 18–23, 1987.

F. Cottet, J Delacroix, C. Kaiser, and Z. Mammeri. *Scheduling in Real-Time Systems*. John Wiley and sons, Chichester, West Sussex, 2002.

J.F. Cox, J.H. Blackstone, and M.S. Spencer. *APICS Dictionary*. American Production and Inventory Control Society, Falls Church, Virginia, 1992.

M. Dastani. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

D.C. Dennett. *The Intentional Stance*. MIT Press, Cambridge, Mass, 1987.

E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269271, 1959.

N. Duffie and R. Piper. Non-hierarchical control of manufacturing systems. *Journal of Manufacturing Systems*, 5(2):137–139, 1986.

H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping (slam): Part i the essential algorithms. *Robotics and Automation Magazine 13 (2)*, pages 99–110, 2006.

T.W. Ellis, F.A. Smith, and L.L. Jones. Methods and opportunities in the recycling of rare earth based materials. *The Metallurgical Society (TMS) conference on high performance composites*, (IS-M–796), 1994.

K. Finkenzeller. *RFID handbook: Radio-frequency identification fundamentals and applications.* John Wiley, 1999.

K. Fisher. Agent-based design of holonic manufacturing systems. *Robotics and Autonomous Systems*, 27(1-2):3–13, 1999.

future.wikia.com/wiki/Domotics. *http://future.wikia.com/wiki/Domotics.* accessed in 2012, 2008.

H.L. Gantt. A graphical daily balance in manufacture. *Transactions of the American Society of Mechanical Engineers*, 24:1322–1336, 1903.

H.L. Gantt. *Work, Wages, and Profits.* Reprinted by Hive Publishing Company, Easton, Maryland, 1973, 1916.

D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.

B. Gnedenko and I. A. Ushakov. *Probabilistic Reliability Engineering.* Wiley-Interscience, 1995.

B. Gnedenko, I.V. Pavlov, and I.A. Ushakov. *Statistical Reliability Engineering.* Wiley-Interscience, 1999.

S.L. Goldman, R. N. Nagel, and K. Preiss. *Agile competitors and virtual organizationsmeasuring agility and infrastructure for agility.* 1995.

T. E. Graedel, E. M. Harper, N. T. Nassar, and Barbara K. Reck. On the materials basis of modern society. *Proceedings of the National Academy of Sciences*, 2013.

M. Gunther. The end of garbage. *Fortune*, 2007.

K. Hamilton, D.M. Lane, K.E. Brown, and J. Taylor. An integrated diagnostic architecture for autonomous underwater vehicles. *Journal of Field Robotics*, (24(6)):497–526, 2007.

R. Harper. *The Connected Home: The future of domestic life.* Springer, 2011.

R. Harper et al. *Inside the smart home.* Springer, 2003.

M.A. Hearst. Mixed-initiative interaction. *IEEE Intelligent Systems*, september/october:14–24, 1999.

J. Herrmann. *Handbook of Production Scheduling.* Springer, 2006.

B. Heydenreich, R. Mller, and M. Uetz. Mechanism design for decentralized online machine scheduling. *Operations Research*, 58(2):445–457, 2010.

K. Hindriks. Agent programming languages: Programming with mental models. *PhD-thesis University of Utrecht*, 2001.

W.-J. van Hoeve, C.P. Gomes, M. Lombardi, and B. Selman. Optimal multi-agent scheduling with constraint programming. *IAAI 2007 proceedings*, 2007.

N.R. Jennings and S. Bussmann. Agent-based control system. *IEEE Control Systems Magazine*, (Vol 23 nr.3):61–74, 2003.

A. Karageorgos, N. Mehandjiev, G. Weichhart, and A. Hämmerle. Agent-based optimisation of logistics and production planning. *Engineering Applications of Artificial Intelligence*, 21(1):28–32, 2003.

J. Kletti. *Manufacturing Execution System - MES*. Springer-Verlag, Berlin Heidelberg, 2007.

A. Koestler. *The Ghost in the Machine*. Arkana Books, London, 1969.

Y. Koren and G. Ulsoy. *Vision, principles and impact of recongurable manufacturing systems*. 2002.

Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. van Brussel. *Recongurable manufacturing systems*. 1999.

G. Kovacs and Heidegger G. Car-recycling sme network with agent-based solutions. *European Research Consortium for Informatics and Mathematics*, (73), 2008.

R.P.J. van der Krogt, M.M. de Weerdt, N. Roos, and C. Witteveen. Multiagent planning through plan repair. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-05)*, pages 1337–1338. ACM press, 2005. ISBN 1-59593-094-9. URL `http://www.pds.ewi.tudelft.nl/ mathijs/aamas05poster.pdf`.

R. Lafeber, G van den Bosch, M. Murre, J.. Bassa, L.J.M. van Moergestel, and E. Puik. Characterisation of high accuracy, feedback controlled, adhesive bonding. *Proceedings of the International Precision Assembly Seminar (IPAS 2012)*, 2012.

P. Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. pages 979–991, 2009.

G.L. Lilien, P. Kotler, and K. Sridhar Moorthy. *Marketing models*. Prentice-Hall of India, 2003.

R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM*, 19(7):395–404, July 1976.

W.N. Mitchell. *Organization and Management of Production*. McGraw-Hill Book Company, New York, 1939.

L.J.M. van Moergestel. *In Zee met C*. Academic Service, SDU, 2009.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. The role of agents in the lifecycle of a product. *CMD 2010 proceedings*, pages 28–32, 2010a.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. Simulation of multiagent-based agile manufacturing. *CMD 2010 proceedings*, pages 23–27, 2010b.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. Decentralized autonomous-agent-based infrastructure for agile multiparallel manufacturing. *Proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2011) Kobe, Japan*, pages 281–288, 2011.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. Production scheduling in an agile agent-based production grid. *Proceedings of the Intelligent Agent Technology (IAT 2012)*, pages 293–298, 2012.

L.J.M. van Moergestel, J.J.Ch. Meyer, W. Langerak, G. Meerstra, N. van Nieuwenburg, F. Pape, E. Puik, and D.H. Telgen. Agents in domestic environments. *International Conference on Control Systems and Computer Science (CSCS 2013), Bucharest, Romenia*, pages 487–494, 2013a.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. Multiagent-based agile manufacturing: requirement-driven low cost production. *Workshop EUMAS 2013, Toulouse*, 2013b.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. Monitoring agents in complex products enhancing a discovery robot with an agent for monitoring, maintenance and disaster prevention. *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2013)*, 2:5–13, 2013c.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. A versatile agile agent-based infrastructure for hybrid production environments. *IFAC Modeling in Manufacturing proceedings, Saint Petersburg*, pages 210–215, 2013d.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. Embedded autonomous agents in products supporting repair and recycling. *Proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2013) Mexico City*, pages 67–74, 2013e.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, D.H. Telgen, R. van Rijn, and B. Segerius. A multiagent-based agile work distribution system. *Proceedings of the Intelligent Agent Technology (IAT 2013)*, pages 293–298, 2013f.

L.J.M. van Moergestel, J.J.Ch. Meyer, E. Puik, and D.H. Telgen. Agent-based manufacturing in a production grid: Adapting a production grid to the production paths. *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2014)*, 1:342–349, 2014.

L. Monostori, J. Vancza, and S.R.T. Kumara. Agent-based systems for manufacturing. *Annals of the CIRP*, 55(2), 2006.

E. Montaldo, R. Sacile, M Coccoli, M Paolucci, and A Boccalatte. Agent-based enhanced workflow in manufacturing information systems: the makeit approach. *J. Computing Inf. Technol.*, (10), 2002.

C. Muñoz, D. Arellano, F.J. Perales, and G. Fontaned. Perceptual and intelligent domotic system for disabled people. *Proceedings of the 6th IASTED International Conference on Visualization, Imaging and Image Processing*, pages 70–75, 2006.

J.P. Müller. *The design of intelligent agents: a layered approach*. Springer, 1996.

M. Neef. A taxonomy of human - agent team collaborations. *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence (BNAIC)*, pages 245–250, 2006.

J. Nielsen and J. Levy. Measuring usability: preference vs. performance. *ACM*, 1994.

J. Nielsen and R.L. Mack. *Usability Inspection Methods*. John Wiley & Sons, 1994.

R.J.C. Nunes. Home automation - a step towards better energy management. *International conference on renewable energies and power quality*, 2003.

R.J.C. Nunes and S.J. da Silva. Adding intelligence to home automation systems. *IADIS international conference applied computing*, 2004.

D. Ouelhadj, C. Hanachi, and B. Bouzouia. Multi-agent architecture for distributed monitoring in flexible manufacturing systems (fms). *ICRA 2000 proceedings*, pages 2416–2421, 2000.

M. Paolucci and R. Sacile. *Agent-based manufacturing and control systems : new agile manufacturing solutions for achieving peak performance*. CRC Press, Boca Raton, Fla., 2005.

M.P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE 2003)*, pages 3–12, 2003.

H.V.D. Parunak. What can agents do in industry, and why? an overview of industrially-oriented r&d at cec. *Cooperative Information Agents II Learning, Mobility and Electronic Commerce for Information Discovery on the Internet, Lecture Notes in Computer Science*, 1435:1–18, 1998.

M. Poppendieck and M.A. Cusumano. Lean software development: A tutorial. *Software, IEEE*, 29(5):26–32, 2012.

E. Puik and L.J.M. van Moergestel. Agile multi-parallel micro manufacturing using a grid of equiplets. *Proceedings of the International Precision Assembly Seminar (IPAS 2010)*, pages 271–282, 2010.

E. Puik, L.J.M. van Moergestel, and D.H. Telgen. Cost modelling for micro manufacturing logistics when using a grid of equiplets. *International Symposium on Assembly and Manufacturing (ISAM 2011)*, 2011.

M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Eheeler, and Ng A. Ros: an open source robot operating system. *Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, 2009.

R.J. Rabello, L.M. Camarinha-Matos, and Afsarmanesh H. Multi-agent-based agile scheduling. *Robotics and Autonomous Systems Volume 27, Issues 1-2*, pages 15–28, 1999.

P. Rane. Design of modbus controller using vhdl for remote administrations of a network of devices. pages 694–697, Nov 2010.

A.S. Rao and M.P. Georgeff. Agentspeak. *Proceedings of the Seventh European Workshop on Moddeling Autonomous Agents in a Multi-Agent World*, 1038 of LNAI Springer, 1996.

M. Ruta, F. Scioscia, G. Loseto, and E. Di Sciascio. Perceptual and intelligent domotica systems for disabled people. 2006.

M. Ruta, F. Scioscia, G. Loseto, and E. Di Sciascio. An agent framework for knowledge-based homes. *Third International Workshop on Agent Technologies for Energy Systems (ATES 2012). A workshop of the Eleventh International Conference on Autonomous Agents an Multiagent Systems (AAMAS 2012)*, 2012.

G. Salvendy. *Handbook of Industrial Engineering, Technology and Operations Management.* John Wiley & Sons Ltd., 2001.

R.A. Sarker and R. Khan. An optimal batch size for a production system operating under periodic delivery policy. *Computers & Industrial Engineering*, 37 issue 4:711–730, 2013.

R.J. Schonberger. Some observations on the advantages and implementation issues of just-in-time production systems. *Journal of Operations Management*, 3(1):1–11, 1982.

W.T. Shaw. *Computer Control of BATCH Processes.* EMC Controls, 1982.

W. Shen, Q. Hao, H.J. Yoon, and D.H. Norrie. Applications of agent-based systems in intelligent manufacturing: An update review. *Advanced Engineering Informatics*, pages 415–431, 2006.

S. Shingo. *A Study of the Toyota Production System.* Productivity Press, 1989.

B. Shneiderman and P. Maes. Direct manipulation vs. interface agents. *Interactions*, 4 No. 5:42–61, 1997.

M.P. Singh. Agent communication languages: Rethinking the principles. *Lecture Notes in Computer Science*, 2650(0302-9743):37–50, 2003.

J. Song, H. Song, A.K. Mok, and D. Chen. Wirelesshart: Applying wireless technology in real-time industrial process control. *Symposium on Real-Time and Embedded Technology and Applications*, pages 377–386, 2008.

A.S. Tanenbaum and D.J. Wetherall. *Computer Networks.* Prentice Hall, 5th edition, 2010.

A.S. Tanenbaum, F.M. Kaashoek, R. van Renesse, and H.E. Bal. The amoeba distributed operating system - a status report. *Computer Communications*, (14):324–335, 1991.

D.H. Telgen, L.J.M. van Moergestel, E. Puik, and J.J.Ch. Meyer. Requirements and matching software technologies for sustainable and agile manufacturing systems. *INTELLI 2013 proceedings*, pages 30–35, 2013.

K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing real-time communications: controller area network (can). pages 259–263, 1994.

N. Vergunst. *BDI-based Generation of Robust Task-Oriented Dialogues*. Thesis Utrecht University, 2011.

B. Vogel-Heuser, G. Kegel, K. Bender, and K. Wucherer. Global information architecture for industrial automation. *ATP, Automatisieringstechnische Praxis*, 1(2):108–115, 2009.

Y.H. Wang, C.W. Yin, and C.W. Zhang. A multi-agent and distributed ruler based approach to production scheduling of agile manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 16(2), 2003.

M.M. de Weerdt and B.J. Clement. Introduction to planning in multiagent systems (preprint). *Multiagent and Grid Systems An International Journal*, 5(4):345–355, 2009. ISSN 1574-1702. URL `http://www.st.ewi.tudelft.nl/ mathijs/publications/mags09.pdf`.

M.M. de Weerdt, H. Tonino, and C. Witteveen. Cooperative heuristic multi-agent planning. In *Proceedings of the Thirteenth Belgium-Netherlands Artificial Intelligence Conference (BNAIC-01)*, pages 275–282, 2001. URL `http://www.pds.ewi.tudelft.nl/ mathijs/bnaic01.pdf`.

O.W. Wight. *Production and Inventory Management in the Computer Age*. Van Nostrand Reinhold Company, Inc., New York, 1984.

K. Wnuk, B. Fulkerson, and J. Sudol. A scalable architecture for multi agent vision based robot scavenging. *American Association for Articial Intelligence*, 2006.

M. Wooldridge. *An Introduction to MultiAgent Systems, Second Edition*. Wiley, Sussex, UK, 2009.

M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, (10(2)):115–152, 1995.

M. Wooldridge and N. Jennings. Pitfalls of agent-oriented development. *Proceedings of the Second International Conference on Autonomous Agents, ACM Press, New York*, (Agents 98), 1998.

www.knx.org. *http://www.knx.org/knx-standard/main-advantages/*. accessed in 2012, 2012.

W. Xiang and H.P. Lee. Ant colony intelligence in multi-agent dynamic manafacturing scheduling. *Engineering Applications of Artificial Intelligence*, 16(4):335–348, 2008.

Summary

The thesis describes the application of agent technology in product manufacturing and product support. In this context an agent is an autonomous operating software entity designed for a special goal and living in an environment. Agents receive information about the environment, can influence the environment and are capable to communicate with each other. Important issues in the requirements of modern production are short time to market, requirement-driven production and low cost small quantity production. To meet these requirements special low cost production platforms have been developed in our research. These reconfigurable platforms are called equiplets. A grid of these equiplets connected by a fast network is capable of producing a variety of different products in parallel. This is what we call multiparallel agile manufacturing.

The multiagent-based software infrastructure is responsible for the agile manufacturing. Two types of agents play an important role. A product agent is responsible for the production of a single product and equiplet agents will perform the production steps to assemble the product. The equiplet agent will receive product step parameters from the product agent and the equiplet agent will inform the product agent about the completion of a product step. This concept has many differences with standard mass production. Every product needs its own, possibly unique path along the equiplets. The scheduling is product-based and not batch-based. There is no concept as a general production line, but every product should be transported according to its path along the equiplets. The thesis describes the software architecture of this production system and proposes solutions for path planning, scheduling and product transport during manufacturing. For path planning, software has been developed that generates a route with a minimum of transitions between equiplets. The scheduling is based on scheduling concepts, used in real-time operating systems. The transport will be based on automated guided vehicles. The product agent plays an important role in these solutions.

At the end of the production phase, a product has been produced and

there is a software entity, the product agent, that was responsible for the production, that collected production data during manufacturing. This product agent can play an important role in other phases of the life cycle of a product. The concept of the Internet of Things can be implemented by using this product agent, embedded in the product itself of closely tied to the product and living in cyberspace. Several possibilities and advantages of this approach are proposed and investigated in the thesis. To prove these possibilities, several cases have been studied and actually built as a proof of concept.

The third major concept proposed in the thesis is the use of a web interface to enable end users to specify and design their products. The system that has been built, can be characterized in terms of cloud computing as the implementation of a *manufacturing as a service* (MaaS) system. Using equiplets that can perform 3D printing in combination with equiplets that can assemble the printed parts, this creates a new paradigm for automated manufacturing. This paradigm meets the requirements of modern manufacturing as mentioned earlier.

A final topic of research has been the investigation if the proposed manufacturing model could also be applied to situations that differ from the production grid. It turned out that the agent-based architecture can also be applied to situations where equiplets are replaced by human workers. A proof of concept that covers this situation concludes the thesis.

Samenvatting

Dit proefschrift heeft als onderwerp de toepassing van agenttechnologie in productie en productondersteuning. Onder een agent verstaan we in deze context een autonoom opererende software entiteit die gemaakt is om een zeker doel te realiseren en daartoe met de omgeving comuniceert en zelfstandig acties kan uitvoeren. In moderne productiesystemen streeft men ernaar om de tijd van ontwerp tot productie zo kort mogelijk te houden en de productie af te stemmen op de wensen van de individuele eindgebruiker. Vooral dit laatste streven past niet in het concept van massaproductie. Een methode moet gezocht worden om kleine hoeveelheden of zelfs unieke producten tegen een lage kostprijs te fabriceren. Om dit te verwezenlijken zijn voor dit onderzoek speciale goedkope productieplatforms ontwikkeld. Deze herconfigureerbare productiemachines noemen we equiplets. Een verzameling van deze equiplets in een gridopstelling geplaatst en gekoppeld met een snelle netwerkverbinding is in staat om een aantal verschillende producten tegelijk te produceren. Dit noemen we flexibele parallelle productie.

Voor de softwareinfrastructuur is agenttechnologie toegepast. Twee typen agenten spelen hierin een hoofdrol. Een productagent is verantwoordelijk voor de totstandkoming van een enkel product. De productiemachines worden voorgesteld door zogenoemde equipletagenten. De productagent weet wat er moet gebeuren voor het maken van een product terwijl de equipletagent weet hoe een of meer productiestappen moeten worden uitgevoerd. Het hier voorgesteld concept verschilt in veel opzichten van standaard massaproductie. Elk product in wording volgt zijn eigen, mogelijk unieke pad langs de equiplets, de productie wordt per product gescheduled en niet per batch en er is geen sprake van een productielijn. Dit proefschrift stelt de softwarearchitectuur voor en beschrijft oplossingen voor de routeplanning waarbij het aantal wisselingen tussen equiplets geminimaliseerd is, een scheduling die gebaseerd is op schedulingschema's zoals toegepast in real-time operating systems en een op autonome voertuigen gebaseerd transportsysteem. Bij al deze oplossingen speelt de productagent een belangrijke rol.

Na de productiefase is er een product, maar ook een software entiteit, namelijk de product agent die vanaf het begin het bouwplan van het product heeft meegekregen en tijdens de productie belangrijke gegevens heeft verzameld. Gezien deze set van gegevens, is de productagent een interessante kandidaat om een verdere rol te spelen in andere fasen van de levenscyclus van een product. De productagent kan aan de basis staan van de implementatie van 'the Internet of Things'. De productagent kan hiervoor in het product zelf opgenomen worden of in cyberspace voortleven en van afstand contact houden met het product. Verschillende mogelijkheden voor implementatie en voordelen van deze aanpak worden voorgesteld en onderzocht in dit proefschrift. Om de mogelijkheden en voordelen ook te toetsen is een aantal situaties bestudeerd en zijn hier proof of concepts van gegeven door ze daadwerkelijk te bouwen.

Na deze twee onderdelen is als derde punt een webinterface ontwikkeld om het productiegrid vanuit de eindgebruiker te benaderen. Een eindgebruiker kan zo zelf zijn product specificeren of ontwerpen. Het systeem dat op deze wijze is gebouwd kan in termen van cloud computing gekarakteriseerd worden als een Manufacturing as a Service (MaaS) systeem. Door gebruik te maken van equiplets die 3D-printing als productiestap kunnen uitvoeren in combinatie met equiplets die vervolgens de assemblage van de onderdelen kunnen verzorgen, ziet op deze wijze een nieuw productieconcept het daglicht. Dit concept zal de massaproductie niet doen verdwijnen, maar biedt een interessant alternatief. Dit alternatief voldoet aan de eerder genoemde doelstellingen, namelijk dat de tijd van ontwerp tot productie kort is en de eindgebruiker het product geheel naar eigen wens kan specificeren. Het maken van kleine oplagen of enkele producten is op een kosteffectieve manier mogelijk, omdat dit binnen het productiegrid voor verschillende producten gelijktijdig kan gebeuren met relatief goedkope productiemachines.

Als laatste onderzoeksonderwerp is gekeken of het voorgesteld productiemodel ook toe te passen is op andere situaties dan een productiegrid, zoals een productie- of werkomgeving met mensen in plaats van equiplets. Ook hier is een proof of concept gebouwd om de veronderstelling te staven.

Curriculum Vitae

Persoonlijke gegevens

Naam: Leonoardus Joseph Maria van Moergestel
Geboortedatum: 24-03-1955 Geboorteplaats: Dussen

Opleiding

Gymnasium B
Kandidaats natuurkunde met bijvak sterrenkunde
Doctoraal experimentele natuurkunde met bijvak elektrotechniek
Onderzoekscripties: Elektron-atoom botsingen,
Monte Carlo simulatie van ladingstransport in silicium,
Generatie-recombinatieruis in silicium

Werkervaring

1981-1990 Wetenschappelijk systeemprogrammeur en hardwarespecialist bij de Vrije Universiteit in Amsterdam.
1990-2013 Hogeschooldocent aan de Hogeschool Utrecht in onder meer de vakken elektrotechniek, elektronica, programmeren in C, operating systems.
2014: Hogeschoolhoofddocent aan de Hogeschool Utrecht

SIKS Dissertatiereeks

====
2009
====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making
using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks
Based on Knowledge, Cognition, and Quality

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies
(making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation
Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and
Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management:
An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens;
Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation:
a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search
and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

====
2010
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for
Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know?
Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software.
Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway?
How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions

2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning;
Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions

2010-50 Bouke Huurnink (UVA)
Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI)

Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams:
Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UVA)
Combining Concepts and Language Models for Information Access

====
2011
====

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier(UU)
Organizing Agent Organizations.
Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03
Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning;
Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age
- Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16	Maarten Schadd (UM) Selective Search in Games of Different Complexity
2011-17	Jiyin He (UVA) Exploring Topic Structure: Coherence, Diversity and Relatedness
2011-18	Mark Ponsen (UM) Strategic Decision-Making in complex games
2011-19	Ellen Rusman (OU) The Mind ' s Eye on Personal Profiles
2011-20	Qing Gu (VU) Guiding service-oriented software engineering - A view-based approach
2011-21	Linda Terlouw (TUD) Modularization and Specification of Service-Oriented Systems
2011-22	Junte Zhang (UVA) System Evaluation of Archival Description and Access
2011-23	Wouter Weerkamp (UVA) Finding People and their Utterances in Social Media
2011-24	Herwin van Welbergen (UT) Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
2011-25	Syed Waqar ul Qounain Jaffry (VU) Analysis and Validation of Models for Trust Dynamics
2011-26	Matthijs Aart Pontier (VU) Virtual Agents for Human Communication - Emotion Regulation and Involvement - Distance Trade-Offs in Embodied Conversational Agents and Robots
2011-27	Aniel Bhulai (VU) Dynamic website optimization through autonomous management of design patterns
2011-28	Rianne Kaptein(UVA) Effective Focused Retrieval by Exploiting Query Context and Document Structure
2011-29	Faisal Kamiran (TUE) Discrimination-aware Classification
2011-30	Egon van den Broek (UT) Affective Signal Processing (ASP): Unraveling the mystery of emotions
2011-31	Ludo Waltman (EUR) Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
2011-32	Nees-Jan van Eck (EUR) Methodological Advances in Bibliometric Mapping of Science
2011-33	Tom van der Weide (UU) Arguing to Motivate Decisions
2011-34	Paolo Turrini (UU) Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
2011-35	Maaike Harbers (UU) Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for
Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery:
A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz(VU)
Exploring Computational Models for Intelligent Support of
Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues:
design aspects influencing interaction quality

====
2012
====

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair(VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05	Marijn Plomp (UU) Maturing Interorganisational Information Systems
2012-06	Wolfgang Reinhardt (OU) Awareness Support for Knowledge Workers in Research Networks
2012-07	Rianne van Lambalgen (VU) When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
2012-08	Gerben de Vries (UVA) Kernel Methods for Vessel Trajectories
2012-09	Ricardo Neisse (UT) Trust and Privacy Management Support for Context-Aware Service Platforms
2012-10	David Smits (TUE) Towards a Generic Distributed Adaptive Hypermedia Environment
2012-11	J.C.B. Rantham Prabhakara (TUE) Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
2012-12	Kees van der Sluijs (TUE) Model Driven Design and Data Integration in Semantic Web Information Systems
2012-13	Suleman Shahid (UvT) Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
2012-14	Evgeny Knutov(TUE) Generic Adaptation Framework for Unifying Adaptive Web-based Systems
2012-15	Natalie van der Wal (VU) Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes
2012-16	Fiemke Both (VU) Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
2012-17	Amal Elgammal (UvT) Towards a Comprehensive Framework for Business Process Compliance
2012-18	Eltjo Poort (VU) Improving Solution Architecting Practices
2012-19	Helen Schonenberg (TUE) What's Next? Operational Support for Business Process Execution
2012-20	Ali Bahramisharif (RUN) Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
2012-21	Roberto Cornacchia (TUD) Querying Sparse Matrices for Information Retrieval
2012-22	Thijs Vis (UvT) Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
2012-23	Christian Muehl (UT) Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
2012-24	Laurens van der Werff (UT)

	Evaluation of Noisy Transcripts for Spoken Document Retrieval
2012-25	Silja Eckartz (UT) Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
2012-26	Emile de Maat (UVA) Making Sense of Legal Text
2012-27	Hayrettin Gürkök (UT) Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
2012-28	Nancy Pascall (UvT) Engendering Technology Empowering Women
2012-29	Almer Tigelaar (UT) Peer-to-Peer Information Retrieval
2012-30	Alina Pommeranz (TUD) Designing Human-Centered Systems for Reflective Decision Making
2012-31	Emily Bagarukayo (RUN) A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
2012-32	Wietske Visser (TUD) Qualitative multi-criteria preference representation and reasoning
2012-33	Rory Sie (OUN) Coalitions in Cooperation Networks (COCOON)
2012-34	Pavol Jancura (RUN) Evolutionary analysis in PPI networks and applications
2012-35	Evert Haasdijk (VU) Never Too Old To Learn On-line Evolution of Controllers in Swarm- and Modular Robotics
2012-36	Denis Ssebugwawo (RUN) Analysis and Evaluation of Collaborative Modeling Processes
2012-37	Agnes Nakakawa (RUN) A Collaboration Process for Enterprise Architecture Creation
2012-38	Selmar Smit (VU) Parameter Tuning and Scientific Testing in Evolutionary Algorithms
2012-39	Hassan Fatemi (UT) Risk-aware design of value and coordination networks
2012-40	Agus Gunawan (UvT) Information Access for SMEs in Indonesia
2012-41	Sebastian Kelle (OU) Game Design Patterns for Learning
2012-42	Dominique Verpoorten (OU) Reflection Amplifiers in self-regulated Learning
2012-43	Withdrawn
2012-44	Anna Tordai (VU) On Combining Alignment Techniques

2012-45	Benedikt Kratz (UvT) A Model and Language for Business-aware Transactions
2012-46	Simon Carter (UVA) Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
2012-47	Manos Tsagkias (UVA) Mining Social Media: Tracking Content and Predicting Behavior
2012-48	Jorn Bakker (TUE) Handling Abrupt Changes in Evolving Time-series Data
2012-49	Michael Kaisers (UM) Learning against Learning Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
2012-50	Steven van Kervel (TUD) Ontologogy driven Enterprise Information Systems Engineering
2012-51	Jeroen de Jong (TUD) Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

====
2013
====

2013-01	Viorel Milea (EUR) News Analytics for Financial Decision Support
2013-02	Erietta Liarou (CWI) MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
2013-03	Szymon Klarman (VU) Reasoning with Contexts in Description Logics
2013-04	Chetan Yadati(TUD) Coordinating autonomous planning and scheduling
2013-05	Dulce Pumareja (UT) Groupware Requirements Evolutions Patterns
2013-06	Romulo Goncalves(CWI) The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
2013-07	Giel van Lankveld (UvT) Quantifying Individual Player Differences
2013-08	Robbert-Jan Merk(VU) Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
2013-09	Fabio Gori (RUN) Metagenomic Data Analysis: Computational Methods and Applications
2013-10	Jeewanie Jayasinghe Arachchige(UvT) A Unified Modeling Framework for Service Design.
2013-11	Evangelos Pournaras(TUD) Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian(VU)
Knowledge-driven Migration to Services

2013-13 Mohammad Safiri(UT)
Service Tailoring: User-centric creation of integrated IT-based
homecare services to support independent living of elderly

2013-14 Jafar Tanha (UVA)
Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM)
Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU)
Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU)
The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT)
Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD)
Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA)
Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT)
Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN)
Causal Discovery and Logic

2013-23 Patricio de Alencar Silva(UvT)
Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM)
Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM)
Intention-based Decision Support. A new way of representing
and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT)
Architectural Support for Dynamic Homecare Service Provisioning

2013-27 Mohammad Huq (UT)
Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT)
When Complexity becomes Interesting: An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT)
Listening Heads

2013-30 Joyce Nakatumba (TUE)
Resource-Aware Business Process Management: Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT)
Blueprint Model and Language for Engineering Cloud Applications

2013-32 Kamakshi Rajagopal (OUN
Networking For Learning;
The role of Networking in a Lifelong Learner's Professional Development

2013-33 Qi Gao (TUD
User Modeling and Personalization in the Microblogging Sphere

2013-34 Kien Tjin-Kam-Jet (UT
Distributed Deep Web Search

2013-35 Abdallah El Ali (UvA
Minimal Mobile Human Computer Interaction

2013-36 Than Lam Hoang (TUe)
Pattern Mining in Data Streams

2013-37 Dirk Börner (OUN)
Ambient Learning Displays

2013-38 Eelco den Heijer (VU)
Autonomous Evolutionary Art
2013-39 Joop de Jong (TUD)
A Method for Enterprise Ontology based Design of Enterprise Information Systems

2013-40 Pim Nijssen (UM)
Monte-Carlo Tree Search for Multi-Player Games

2013-41 Jochem Liem (UVA)
Supporting the Conceptual Modelling of Dynamic Systems:
A Knowledge Engineering Perspective on Qualitative Reasoning

2013-42 Léon Planken (TUD)
Algorithms for Simple Temporal Reasoning

2013-43 Marc Bron (UVA)
Exploration and Contextualization through Interaction and Concepts

====
2014
====

2014-01 Nicola Barile (UU)
Studies in Learning Monotone Models from Data

2014-02 Fiona Tuliyano (RUN)
Combining System Dynamics with a Domain Modeling Method

2014-03 Sergio Raul Duarte Torres (UT)
Information Retrieval for Children: Search Behavior and Solutions

2014-04 Hanna Jochmann-Mannak (UT)
Websites for children: search strategies and interface design
Three studies on children's search performance and evaluation

2014-05 Jurriaan van Reijsen (UU)
Knowledge Perspectives on Advancing Dynamic Capability

2014-06 Damian Tamburri (VU)
Supporting Networked Software Development

2014-07 Arya Adriansyah (TUE)
Aligning Observed and Modeled Behavior

2014-08 Samur Araujo (TUD)
Data Integration over Distributed and Heterogeneous Data Endpoints

2014-09 Philip Jackson (UvT)
Toward Human-Level Artificial Intelligence:
Representation and Computation of Meaning in Natural Language

2014-10 Ivan Salvador Razo Zapata (VU)
Service Value Networks

2014-11 Janneke van der Zwaan (TUD)
An Empathic Virtual Buddy for Social Support

2014-12 Willem van Willigen (VU)
Look Ma, No Hands: Aspects of Autonomous Vehicle Control

2014-13 Arlette van Wissen (VU)
Agent-Based Support for Behavior Change:
Models and Applications in Health and Safety Domains

2014-14 Yangyang Shi (TUD
Language Models With Meta-information

2014-15 Natalya Mogles (VU)
Agent-Based Analysis and Support of Human Functioning in Complex
Socio-Technical Systems: Applications in Safety and Healthcare

2014-16 Krystyna Milian (VU
Supporting trial recruitment and design by automatically interpreting eligibility criteria

2014-17 Kathrin Dentler (VU)
Computing healthcare quality indicators automatically:
Secondary Use of Patient Data and Semantic Interoperability

2014-18 Mattijs Ghijsen (VU)
Methods and Models for the Design and Study of Dynamic Agent Organizations

2014-19 Vincius Ramos (TUE)
Adaptive Hypermedia Courses:
Qualitative and Quantitative Evaluation and Tool Support

2014-20 Mena Habib (UT
Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

2014-21 Kassidy Clark (TUD)
Negotiation and Monitoring in Open Environments

2014-22 Marieke Peeters (UT)
Personalized Educational Games - Developing agent-supported scenario-based training

2014-23 Eleftherios Sidirourgos (UvA/CWI)
Space Efficient Indexes for the Big Data Era

2014-24 Davide Ceolin (VU)
Trusting Semi-structured Web Data

2014-25 Martijn Lappenschaar (RUN)
New network models for the analysis of disease interaction

2014-26 Tim Baarslag (TUD)
What to Bid and When to Stop

2014-27	Rui Jorge Almeida (EUR) Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
2014-28	Anna Chmielowiec (VU) Decentralized k-Clique Matching
2014-29	Jaap Kabbedijk (UU) Variability in Multi-Tenant Enterprise Software
2014-30	Peter de Kock Berenschot (UvT) Anticipating Criminal Behaviour