# Implementing Manufacturing as a Service:
## A Pull-Driven Agent-Based Manufacturing Grid

Leo van Moergestel[1], Erik Puik[1], Daniël Telgen[1], and John-Jules Meyer[2]

[1] HU Utrecht University of Applied Sciences, Utrecht, the Netherlands
{leo.vanmoergestel, erik.puik, daniel.telgen}@hu.nl
[2] Utrecht University, Utrecht, the Netherlands
J.J.C.Meyer@uu.nl

**Abstract.** User requirements and low-cost small quantity production are new challenges for the modern manufacturing industry. This means that small batch sizes or even the manufacturing of one single product should be affordable. To make such a system cost-effective it should be capable to use the available production resources for many different products in parallel. This paper gives a description of the requirements and architecture of an end-user driven production system. The end-user communicates with the production system by a web interface, so this manufacturing system can be characterized in terms of cloud computing as the implementation of manufacturing as a service, abbreviated to MaaS.

**Keywords:** agile manufacturing, agent technology, MaaS
**Key Terms** Industry, Infrastructure, Machine Intelligence.

## 1 Introduction

At the HU Utrecht University of Applied Sciences, an agile manufacturing system has been developed that is capable of so-called multiparallel production of small batches or even one single product. The need for such a manufacturing system comes from the fact that nowadays the demand for custom end-user specified products is increasing. Internet is offering a method to involve the end-user directly into the production. Also the possibilities of additive manufacturing by using 3D printers offers new ways to set up a manufacturing infrastructure with the focus on the manufacturing of small quantities.

This paper will focus on the interface to connect the end user to the production process. Before going into detail, the manufacturing system itself will first globally be described.

In the next section details about the basic design considerations are given. Because the implementation is based on agent technology, a short description of what an agent is, will be given. The architecture and the connection with the end-user will be the treated next. Finally, the results, related work, discussion and a conclusion will end the paper.

## 2 Global description of the manufacturing system

Every product to be made starts its life as a software entity, that contains the information what should be done to make the product. This software entity is a so-called software agent.

### 2.1 Agents

A common definition of an agent given by Wooldridge and Jennings [13] is:

**Definition** (agent). An agent is an encapsulated computer system or computer program that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives or goals.

The manufacturing system that has been designed is based on a group of cooperating agents. A system with two or more agents is called a multiagent system (MAS). In our design the following properties of agents are important:

- goal: an agent is designed to reach a goal. If reaching the goal is complex, subgoals can be defined as states to be reached to finaly come to the end-goal.
- action: an action is what the agent can do.
- plan: to reach a goal or subgoal the agent builds or receives a plan. Normally a plan consists of a list of actions to reach a goal or subgoal within a certain role.
- role: agents can have different roles. In a multiagent system these roles play an important part in the way agents cooperate.
- behaviour: closely related to the role is the behaviour. This the set of actions that an agent will perform in a certain role.
- belief: a belief is what the agent expects to be the case in the environment.

In multiagent technology other aspects can also be important, but the properties mentioned here are specific for the manufacturing system presented in this paper. The main reason for choosing agent technology is that it offers a natural decomposition of responsibilities and tasks to be completed in this complex manufacturing system. It also means that if one agent fails, other agents can continue to fulfil their own goals and even take over actions or tasks from the failing agent.

### 2.2 The manufacturing grid

The infrastructure of the manufacturing system consists of cheap reconfigurable production machines that we will call equiplets. These equiplets are capable to perform one or more production steps. The set of steps an equiplet can perform depends on its front-end. An equiplet can be reconfigured by changing its front-end. The equiplets are placed in a grid arrangement. In conventional mass production, a line arrangement is used because for all products the same sequence of production steps should be followed. However in our case every product can have a different path along the equiplets, so a grid arrangement is more natural offering multiple mostly shorter paths in case of an arbitrary sequence of equiplets to be visited.

### 2.3 Agent-based manufacturing

The equiplet-based manufacturing description will have its focus on the MAS where the equiplet agent is the representative of the equiplet. An equiplet agent will publish its capabilities. This means it will announce its production steps. It will wait for products to arrive to actually perform the production steps.

The product agent has several roles. It starts with planning the path along the equiplets for the production. Next, it will schedule the production. After successfully scheduling, it 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 agents to check the reliability of a certain equiplet for a certain step with certain parameters. 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.

When the product is finished, the product agent can also have a role in other parts of the life cycle of a product, being a software entity that knows a lot about the product and the actual production. To achieve these roles, the agent could be embedded in the product itself, but being accessible in cyberspace is also a possibility.

## 3 System architecture

In this section a description of the system architecture as well as the constraints on our type of production will be presented.

In figure 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 used in this example. For the MAS layer Jade [1] was used as a platform. Jade is a widely accepted Java-based multiagent environment. The inter-agent communication is implemented by using blackboards. A blackboard is a software entity where agents can publish information that will be available to other agents.

The software for the equiplet is based on ROS. ROS is an acronym for Robot Operating System [10]. ROS is not really an operating system but it is middleware 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. This platform has been chosen for the following reasons:

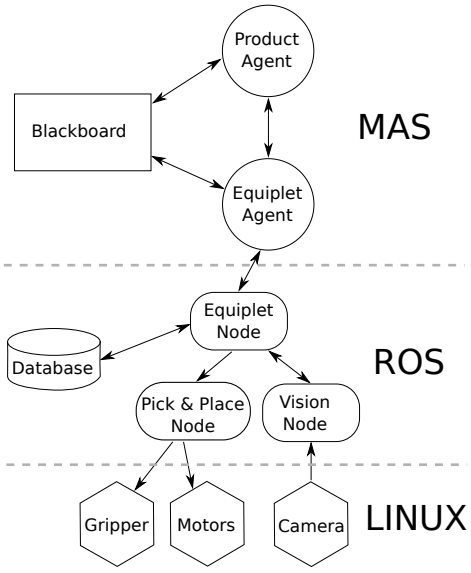


**Fig. 1.** Layered architecture

- 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 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 in combination with a web interface.

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 them 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 a unique QR-code. During the first production steps the trays are filled with all the components required to make the product. This way a kind of construction box 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. Of course within our conceptual model other production models could be used, but the examples given here are based on this model.

## 4 Connecting the end-user

To use the manufacturing 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 manufacturing grid, because the grid itself is offering the webinterface for designing the product. If a product can be made using the webinterface, the grid will be capable to make it. This web interface will be called WIMP as an acronym for Web Interface Managing Production. The addition of a web interface as shown in figure 2

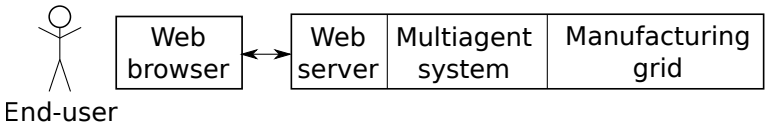


**Fig. 2.** Combination with webinterface

fits neatly in the concept of agile and lean manufacturing [11], 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 3. In this figure blackboards are abbreviated by BB. A web server

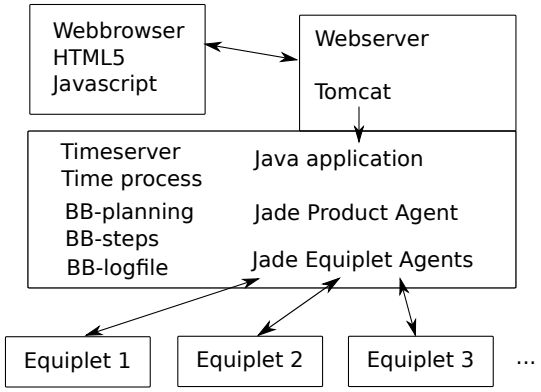


**Fig. 3.** Combination with webinterface

publishes a website where a customer can design his product. This could be a new product if the steps to produce it are within the capabilities of the equiplets in the grid. The webserver will be responsible to offer only those production step possibilities that are present in the grid. 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 4. The numbered components in figure 4 are:

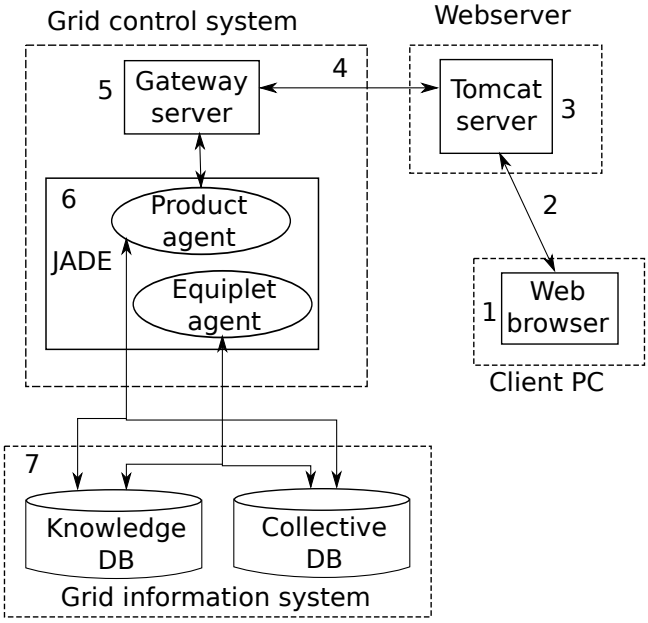


**Fig. 4.** Different platforms and their relations

1. The client PC as used by end-user. The end user can use any HTML-5 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 as will be explained in more detail further on in this paper.
7. The grid information system is a server where the databases and blackboards reside. These are the systems where shared and individual knowledge will be stored.

Agents have to be able to report back to the user. In order to do so, a software solution 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 is not shown in detail in figure 4, but the realisation will be described in the next sections.

### 4.1 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 and 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 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 we were having with the socket behaviour. The problem encountered was the socket behaviour being 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. 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.

### 4.2 WIMP capabilities

At the client side a web-browser receives a web-page in HTML5 format with embedded JavaScript and will display a graphical environment where a product

can be designed. This is the user interface of what has been called the WIMP. At this moment 4 typical product design web interfaces are implemented 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 5. A case with compartments of a certain dimension specified by the user

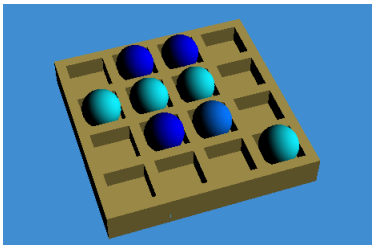

**Fig. 5.** Case with coloured balls in the webbrowser

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. On a canvas, a pixel-based painting using a combination of several colours can be made.

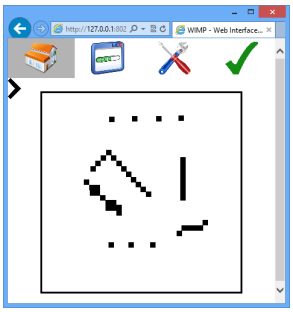

**Fig. 6.** A simple paint example

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 / she did not select the adhesive option for the placement of this part. This part of WIMP is only a basic implementation and in future development

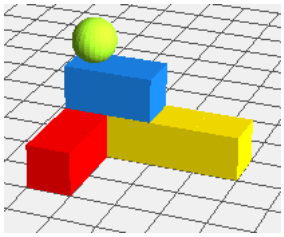

**Fig. 7.** A 3D structure

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.

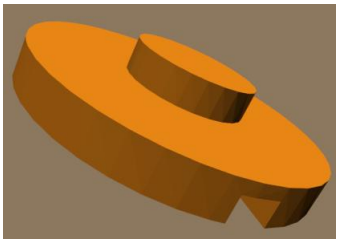

**Fig. 8.** View of an STL-image

At the client side a product is described by JSON. JSON, or JavaScript Simple Object Notation is a popular alternative to XML. XML was 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. Every action described in this information is related to and translated into a production step. In figure 9 the internal structure of a production 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.

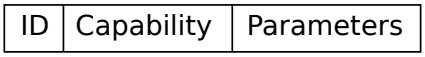


**Fig. 9.** Components of a step object

### 4.3 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 10. 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 3). This

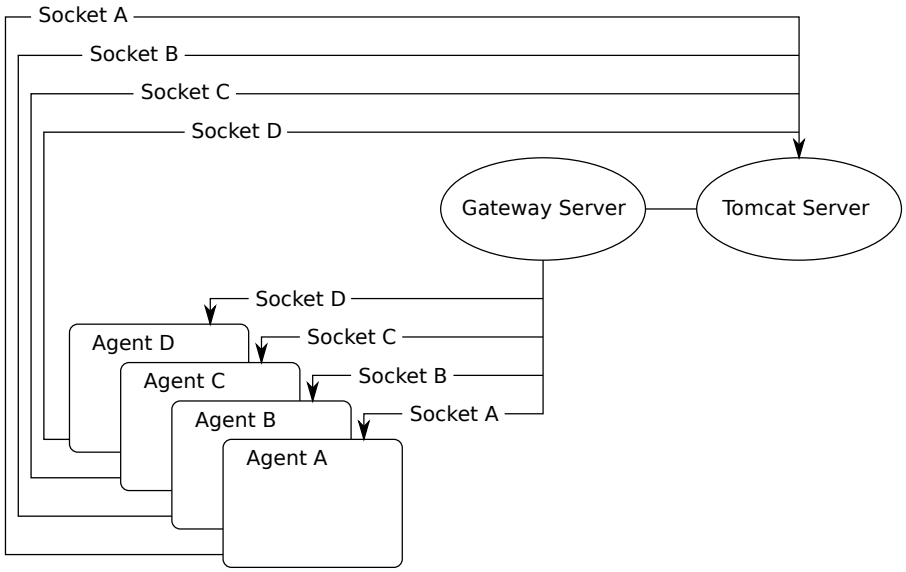


**Fig. 10.** Socket connections between product agents and the user interface

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.

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.

The third blackboard in figure 3 (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. FIPA is an acronym for Foundation for Intelligent Physical Agents and the foundation developed a standard for inter-agent communication. 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.

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.

## 5 Results

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 [5]. 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 [6]. The third step was integrating the MAS with the ROS-based equiplet in the system, so the integration with real equiplet hardware has been accomplished [12]. The latest step is described in this paper. 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 11 a design in the paint application of WIMP is made. In figure 12 the result of this product is shown. Though this example still is very simple, 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 an error recovery system.

## 6 Related work

The concept of using agents for production is not new. Among others a multiagent-based production system has also been developed by Jennings and Bussmann [3][4]. Jennings and Bussmann introduce the concept of a product agent, in their

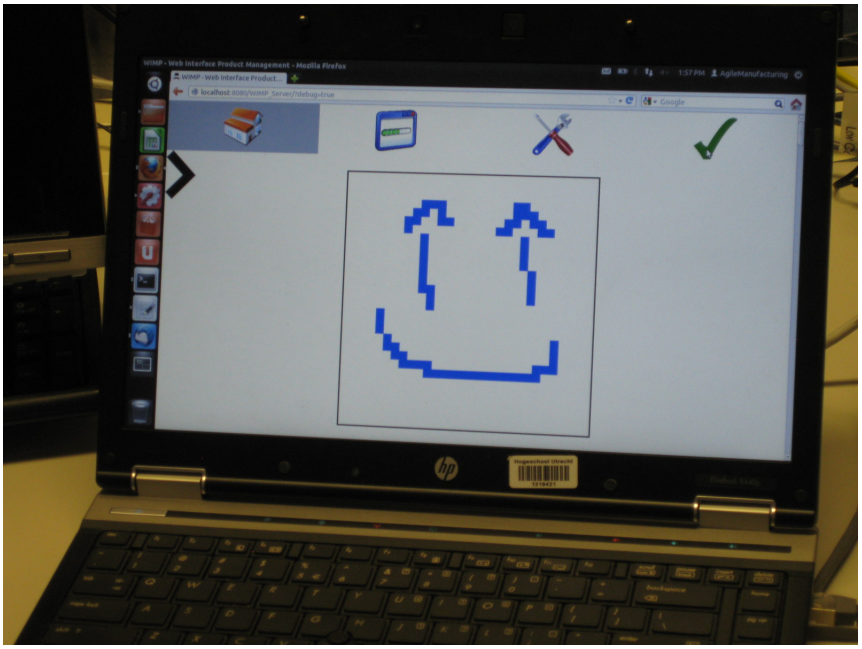

**Fig. 11.** WIMP paint design

terms workpiece agents, during the production. Their system focuses on reliability and minimizing downtime in a production line. This approach is used in the production of cylinder heads in car manufacturing. The roles of the agents in this production system differ from our approach. This has to do with the fact that Jennings and Bussmann use agent technology in a standard pipeline-based production system and the main purpose was to minimise the downtime of this production system. Their agents do not perform individual product logging and only play a role in the production phase. In our approach the product logging is done by the product agent for every single product and could be the basis of the other roles of the product agent in other parts of the life cycle. In the model presented by Jennings and Bussmann the workpiece agent is not so much involved in production details as the product agent in our model. Another big difference is also that our model is end-user driven.

In the field of agent-based production there are several other important publications. Paolucci and Sacile[8] give an extensive overview of what has been done. Their work focuses on simulation as well as production scheduling and control. The main purpose to use agents in [8] is agile production and making complex production tasks possible by using a multi-agent system. Agents are also introduced to deliver a flexible and scalable alternative for MES for small production companies. The roles of the agents in their 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.

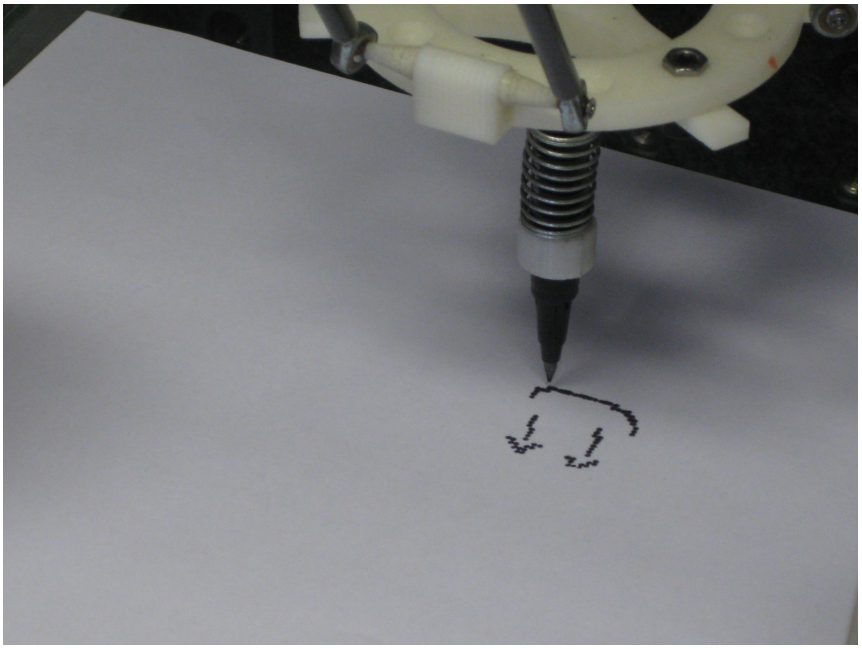

**Fig. 12.** Resulting product on the equiplet

Agent technology is used in parts or subsystems of the manufacturing process. We on the contrary based the manufacturing process as a whole on agent technology and we have developed a production paradigm based on agent technology in combination with a manufacturing grid. This model uses only two types of agents and focuses on agile multiparallel production. The design and implementation of the production platforms and the idea to build a manufacturing grid can be found in Puik[9]. After production the product agents can be embedded, if possible, in the product itself. In [7] the role of product agents in the whole life cycle of a product is discussed.

The term industrial internet [2] is used to describe the possibilities of interconnected machinery, sensors and devices that can be used to enhance production and solving emergent problem on the fly. Research in this field is related to our research. The approach we used however is purely based on the aforementioned cheap reconfigurable equiplets. The introduction of agent technology opens possibilities that go beyond the production phase, as the product agent can play an important role in other parts of the life cycle of a product.

## 7 Discussion and future work

The production approach described here is also applicable to a hybrid system containing human actors as parts of the production system. In this situation

human workers take the position of the equiplets. The production steps for a certain product should be translated to human-readable instructions and humans perform the actual production steps. In that model the equiplet agent 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.

Standard mass production always has the risk of overproduction, especially when new products arrive from other sources offering better performance or a lower price. In the concept of lean manufacturing, this kind of waste should be avoided by so-called pull-driven production. This means that a product will only be made if an end-user is asking for it. This is exactly what has been accomplished in the manufacturing system described in this paper.

For transport of the products between equiplets, automated guided vehicles (AGV) are being developed. However, the use of AGVs is not implemented yet, but the transport between equiplets can be seen as a step needed in the sequence of steps to make a product. This means that from the point of view of the product agent, an AGV is just another equiplet, offering the product transport step fitting in the total sequence of steps needed for manufacturing the product. There is a difference however. The AGV is reserved for a whole sequence of steps, while equiplets are reserved for just a single step or a set of steps if these steps are consecutive and can be realised by the same equiplet.

## 8 Conclusion

In this paper 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 manufacturing 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 and embedded in 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.

## References


1. Bordini, N., Dastani, M., Dix, J., Seghrouchni, A.E.F.: Multi-Agent Programming. Springer (2005)
2. Bruner, J.: http://radar.oreilly.com/2013/01/defining-the-industrial-internet.html (2013)
3. Bussmann, S., Jennings, N., Wooldridge, M.: Multiagent Systems for Manufacturing Control. Springer-Verlag, Berlin Heidelberg (2004)
4. Jennings, N., Bussmann, S.: Agent-based control system. IEEE Control Systems Magazine (Vol 23 nr.3), 61–74 (2003)

5. Moergestel, L.v., Meyer, J.-J., Puik, E., Telgen, D.: Decentralized autonomous-agent-based infrastructure for agile multiparallel manufacturing. ISADS 2011 proceedings pp. 281–288 (2011)
6. Moergestel, L.v., Meyer, J.-J., Puik, E., Telgen, D.: Production scheduling in an agile agent-based production grid. IAT 2012 proceedings pp. 293–298 (2012)
7. Moergestel, L.v., Meyer, J.-J., Puik, E., Telgen, D.: Embedded autonomous agents in products supporting repair and recycling. Proceedings of the International Symposium on Autonomous Distributed Systems (ISADS 2013) Mexico City pp. 67–74 (2013)
8. Paolucci, M., Sacile, R.: Agent-based manufacturing and control systems : new agile manufacturing solutions for achieving peak performance. CRC Press, Boca Raton, Fla. (2005)
9. Puik, E., Moergestel, L.v.: Agile multi-parallel micro manufacturing using a grid of equiplets. IPAS 2010 proceedings pp. 271–282 (2010)
10. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Eheeler, R., A., N.: Ros: an open source robot operating system. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA) (2009)
11. Shingo, S.: A Study of the Toyota Production System. Productivity Press (1989)
12. Telgen, D., Moergestel, L.v., Puik, E., Meyer, J.: Requirements and matching software technologies for sustainable and agile manufacturing systems. INTELLI 2013 proceedings pp. 30–35 (2013)
13. Wooldridge, M., Jennings, N.: Intelligent agents: Theory and practice. The Knowledge Engineering Review (10(2)), 115–152 (1995)