Multiagent-based agile manufacturing: requirement-driven low cost production

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@hu.nl,
WWW home page: http://www.hu.nl
[2] Utrecht University, Utrecht,
the Netherlands

Abstract. The production system described in this paper in an implementation of an agile agent-based production system. This system is designed to meet the requirements of modern production, where short time to market, requirement-driven production and low cost small quantity production are important issues. The production is done on special devices called equiplets. A grid of these equiplets connected by a fast network is capable of producing a variety of different products in parallel. The multi-agent-based software infrastructure is responsible for the agile manufacturing. A product agent is responsible for the production of a single product and equiplet agents will perform the production steps to assemble the product. This paper describes this multiagent-based production system with the focus on the product agent.

1 Introduction

The requirements of modern production systems are influenced by new demands like time to market and customer-specific small quantity production. In other words, the time between product development and production should be minimal and small quantity production must be cheap. To fulfil these requirements new production methods need to be developed. This new approach means new production hardware as well as co-designed software.

Our research group has developed special production platforms that are cheap, agile and easy configurable [11]. These platforms can operate in parallel. We call these platforms equiplets and a collection of these equiplets is called a production grid.

The multiagent-based software infrastructure for such a production grid is highly responsible for this agile and diverse way of manufacturing. In [4] an agent-based software infrastructure is described as well as the reason why agents are used. To test the concepts presented in that paper, a test environment has been built to see how such an agent-based production system would behave. The production planning and scheduling in this multiagent-based production system is published in [5]. The current paper discusses the architecture and the practical implementation of this agile production system.

It focuses on the creation and roles of the agents that are responsible for the assembling of a single product. These agents are called product agents. In this paper first the manufacturing model is introduced and discussed as well as the roles of the agents. Next the architecture and the constraints of the practical realisation are discussed. The implementation of product agents and the software infrastructure for the production of a two-dimensional structure is presented in combination with related issues like error recovery.

2 MAS-based agile manufacturing

First a short introduction of the agent-based agile manufacturing system will be presented here. The basic idea is that every single product is represented by a so called product agent. The production equipment, in this case consisting of the aforementioned equiplets, is also represented by an agent for every equiplet. The product agent knows which production steps are needed to manufacture a product. An equiplet agent knows how to perform one or more production steps. The production system consists of a grid of equiplets. An equiplet is a cheap basic production unit with an on-board computer system for control and communication. Each equiplet comes with a front-end, a production hardware add-on for the equiplet, that enables it to perform certain production actions. The front-end will determine what kind of production actions (combining into production steps) an equiplet can perform. Let us assume that the grid offers a set S_{grid} of N production steps $\sigma_1 ... \sigma_N$. An equiplet with a certain front-end 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 sequence of production steps, e.g. $< \sigma_4, \sigma_7, \sigma_2, \sigma_1 >$. Other possibilities exist. Later on we will show the situation where first two half-products are made that will be combined. In some situations the order of two or more production steps is irrelevant. This will result in a choice of tuples.

The equiplet is controlled by an equiplet agent. This agent is responsible for a certain equiplet and its front-end. It interacts with the production hardware, other agents in the grid and possibly, in a semi-automated environment, with a human equiplet operator. An equiplet agent will:

- announce its steps on a blackboard that is readable for all product agents in its role of publisher;
- wait for clients (product agents) to arrive in its role of waiter;
- perform production steps in its role of step performer;
- inform clients about results of a step in its role of step performer.

In all its roles it will also inform product agents about the feasibility of steps in combination with certain parameters.

Now the so-called product agent comes into the picture. This agent is, as earlier mentioned, responsible for the realisation of the product. Its goal is a

finished product. In its first role as planner the selection of equiplets is done in four steps or phases:

1. Select possible equiplets for a certain production step. Repeat this for all steps needed;
2. Ask equiplets if the production steps with given parameters are feasible;
3. Optimize production path;
4. Schedule the whole production in an atomic action.

A product agents chooses an equiplet based on the set of production steps published by the equiplet. It will direct the yet unfinished product to the equiplet and log the production data resulting from the production step or steps; thus creating a production log. After the equiplet has finished its production steps, the product agent will turn to another equiplet according to the scheduling and direct the product there, and so on until the final production step is completed and the product can be removed from the grid. In this way we can produce different products in parallel as long as the set of production steps offered by the grid fits the needs of the products to be made. The production paths of separate product agents with different production steps result in a fabric as depicted in figure 1. In this figure all steps take the same amount of time. In practice however this will not be the case and in our model all steps take an integer value of a fixed time-unit. In this paper this fixed time-unit is referred to as time step.

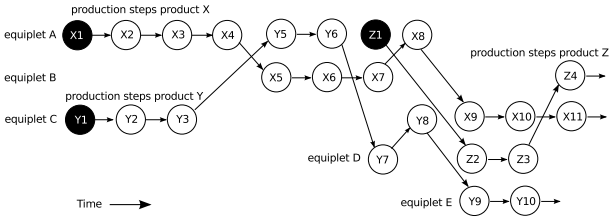

Fig. 1. Fabric of production paths.

2.1 Roles of the product agent

The product agents act fully cooperative in the MAS. The product agent is created in the grid itself. There are also provisions made that their production steps fit within the capabilities of the grid.

Planning and scheduling We define a production path as a sequence of production steps. Consider a product to be built with three production steps, this product has production path:

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

Let us assume a simple grid with three equiplets E_1, E_2 and E_3, 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:

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

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 exist. A possible 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 will be very high if the equiplets each offer a big set of production steps. If the order op production steps is irrelevant, the same thing can happen. However in practice an equiplet offers only one or perhaps two steps. Keeping in mind that only one solution is needed, the number of solutions that are calculated by the product agents is in our situation limited to eight. One solution is not enough, because in case of an infeasible scheduling, other solutions should be at hand. In [5] mechanisms to optimise the scheduling are introduced. Summarised these optimisations are: taking care of the load of a certain equiplet so an alternative equiplet is searched for in case an equiplet has a high load, minimising the amount of movements of the product between equiplets and finally avoidance of unreliable functioning equiplets.

The scheduling implementation The scheduling is implemented as an atomic action for the product agent. The product agent will schedule all production steps it needs, while other agents are temporarily blocked from scheduling. This will prevent deadlocks. The products allocates available free timeslots for all equiplets it needs for production. If the complete path of steps is within the deadline, the scheduling is considered successful. If the scheduling fails the product agent will do a reschedule. This reschedule is based on the "earliest deadline first" (EDF) approach. This approach turned out to give a high success rate [5]. The product agent that encounters a failing scheduling, will ask all agents with a later deadline to hand over their scheduling and the product agent with the failing scheduling will try to reschedule itself and all agents having a later deadline according to the EDF-approach. If this results in a feasible scheduling for all involved product agents, the new scheduling will be reported to all agents that temporally gave up their scheduling. If this rescheduling fails for one or more agents, the product agent that did the rescheduling will reports a scheduling failure to its maker and gives up. The other agents continue with their old scheduling schemes.

Planning and scheduling complex production 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 2 a simple example of this situation is presented. In a more formal approach this means that the product steps are still a tuple, but the members of these tuples could be sets of tuples. The product path of figure 2 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 production 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.

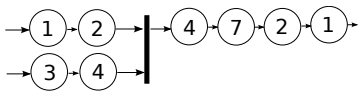

Fig. 2. Combining two half products.

Guiding the product 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 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 4.9.

Role in other parts of the life cycle of a product When the production is finished the product agent could cease to exist. However, 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. An overview is given in [6]. In some situations shown in this overview the role of the product agents will be to represent the product in the Internet of Things.

3 System architecture

In this section a description of the system architecture as well as the software will be presented. In figure 3 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.

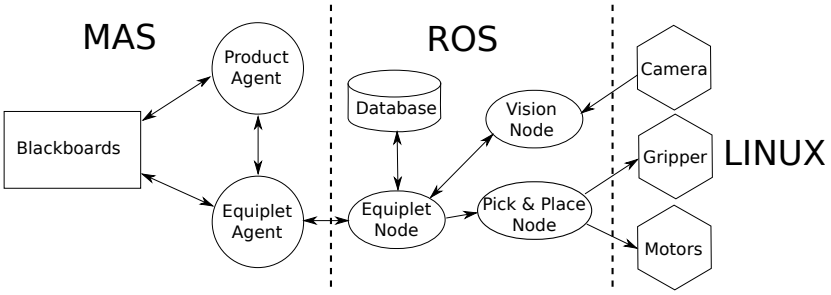

Fig. 3. Layered architecture.

For the MAS layer Jade [1] was used as a platform. The reasons for choosing Jade are:

- the production grid is a multi-agent-based system. Jade provides most of the requirements we need for our application like platform-independence and inter agent communication;
- Jade is Java-based. Java is a versatile and powerful programming language;
- because Jade is Java-based it also has a low learning curve for Java programmers;
- the product agents should be capable to negotiate to reach their goals. Jade offers possibilities for agents to negotiate.
- agents can migrate, terminate or new agents can appear.

The Jade runtime environment implements message-based communication between agents running on different platforms connected by a network. The software for the equiplet is based on ROS. ROS is an acronym for Robot Operating

System [12]. 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 4 shows the relation between two nodes and one topic. Node /talker communicates with node /listener by means of the communication mechanism or topic /chatter

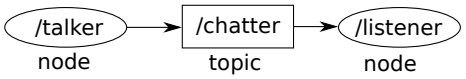

Fig. 4. Two nodes connected by a topic.

This 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 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 3 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.

4 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.

4.1 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. 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. 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.

4.2 Final architecture and software

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 architecture of the software of the manufacturing system is depicted in figure 5. Every block will be described in more detail. First the entire picture will be

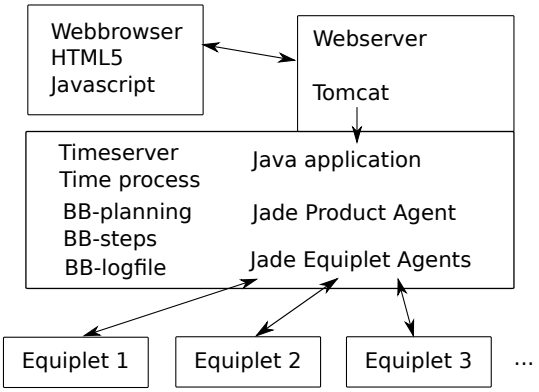

Fig. 5. Combination with webinterface.

explained. 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.

As a simple example we will demonstrate the production of a mosaic consisting of coloured balls in a 4×4 box as shown in figure 6. The reason why this example is chosen is that it is a nice demo of the pick and place capabilities of an equiplet that has been constructed by our group and the web-interface is still rather simple.

4.3 Client side

At the client side a web-browser receives a web-page in HTML5 format with embedded JavaScript to display a graphical environment where a product can be designed. A simple example of a mosaic is shown in a screen-shot in figure 6.

The client side 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

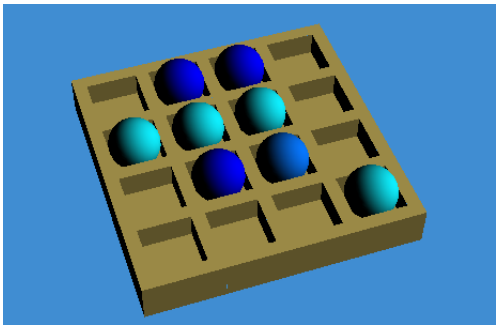

Fig. 6. A mosaic in the webbrowser.

not select the adhesive option for the placement of this part. At the client side a product is described by XML. 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 XML information. By clicking the submit button, the XML information is posted to the webserver.

4.4 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. Jade Gateway is used in the Jade environment to achieve this functionality. This newly spawned agent will also receive the XML information about the product to be made. From this information, the needed product steps are generated by the product agent.

4.5 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 acquisition 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.

4.6 Blackboard and timing

The blackboard system as described in the architecture was implemented as actually three separate blackboards. This has to do with the fact that the performance of the system could be better and also the read and write access permissions became 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 equiplet. A time-pointer is pointing at the current time step. This pointer is updated by the timing process, that will also clear the time step that have 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 contain the unique product agent id that belongs to the agent that has reserved that slot. To synchronise all agents, a timeserver has been added to the system. As already explained, 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 [5]. The third blackboard 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.

4.7 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.

4.8 Interaction

The webinterface will deliver an XML-description of the product. This is translated to product steps by the product agent. The product agent will parse the XML information and generate a step for every part of the XML file the actually changes the product. The webinterface takes care for deletion and retries,

so the XML file itself should not be adapted by the product agent. The XML information for a 2x2 tray of the example looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<mosaic rows="2" columns="2">
  <row>
    <cell object="ball" colour="red"/>
    <cell object="ball" colour="green"/>
  </row>
  ...
</mosaic>
```

This will result in three similar production steps (pick-and-place) from the point of view of the product agent. Now the product will interact will 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 the following FIPA interaction de product agent is abbreviated as PA and the equiplet agent as EA.

```
PA: query-if(action(object))
// pick-and-place(ball)
EA: confirm or disconfirm
EA: inform(yes/no, time duration)
```

When the step is actually carried out, the following interaction willl be used:

```
PA: request(action(parameters, ..)
EA: agree
EA: inform(status)  // success - failure
PA: request(information) // product log
EA: inform(information)
```

For every interaction, a watchdog timer will take care of possible time-outs. 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 paper 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 [13]. 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 send to the ROS-layer that will control the hardware and the commands are actually carried out by the connected hardware.

4.9 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, 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. 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.

5 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 [4]. 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 [5]. The latest milestone is described in this paper where we made the two final steps. First we integrated the MAS with the ROS-based equiplet 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 example can be executed on the three available

equiplets. So the total chain from design to production is working. 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.

Future research will focus on more equiplets having complexer production steps and automatic reconfiguration of equiplets.

6 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. Paolucci and Sacile[10] give an extensive overview of what has been done in this field. Their work focuses on simulation as well as production scheduling and control[8]. The main purpose to use agents in [10] is agile production and making complex production tasks possible in a standard production environment by using a multi-agent system. Our approach is a co-design of hardware and software. 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. We on the contrary based the manufacturing process as a whole on agent technology. In our case a co-design of hardware and software was the basis.

Bussmann and Jennings [2][3] 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. Some characteristics of their solutions are:

- The production system is a production line that is built for a certain product. This design is based on redundant production machinery and focuses on production availability and a minimum of downtime in the production process. Our system is a grid and is capable to produce many different products in parallel;
- The roles of the agents in this 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. In our system the negotiating is between the product agents, thus not disrupting the machine agents;

The solution presented by Bussmann and Jenning has the characteristics of a production pipeline and is very useful as such, however it is not meant to be an agile multi-parallel production system as presented here.

Other authors focus on using agent technology as a solution to a specific problem in a production environment. In [9] a multi-agent monitoring is presented. This work focusses on overall monitoring a manufacturing plant but doe not focus on every single product. The approach we use monitors the production of every single product. The work of Xiang and Lee [14] 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. The implementation and a simulation of the performance is discussed. In our approach the negotiating is between product agents and load balancing is possible by encouraging product agents to use equiplets with a low load. We did not focus on a specific part of the production but we developed a complete production paradigm based on agent technology in combination with a production grid. There is a much stronger role of the product agent and a product log is produced per product. The design and implementation of the production platforms and the idea to build a production grid can be found in Puik[11].

7 Conclusions

In this paper an agile agent-based production system is presented. 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. The production is completely demand driven 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 [7].

References

1. Bordini, N., Dastani, M., Dix, J., Seghrouchni, A.E.F.: Multi-Agent Programming. Springer (2005)
2. Bussmann, S., Jennings, N., Wooldridge, M.: Multiagent Systems for Manufacturing Control. Springer-Verlag, Berlin Heidelberg (2004)

3. Jennings, N., Bussman, S.: Agent-based control system. IEEE Control Systems Magazine (Vol 23 nr.3), 61–74 (2003)
4. 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)
5. 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)
6. Moergestel, L.v., Meyer, J.-J., Puik, E., Telgen, D.: Monitoring agents in complex products enhancing a discovery robot with an agent for monitoring, maintenance and disaster prevention. ICAART 2013 proceedings 2, 5–13 (2013)
7. Moergestel, L.v., Meyer, J.-J., Puik, E., Telgen, D.: A versatile agile agent-based infrastructure for hybrid production environments. IFAC Modeling in Manufacturing proceedings, Saint Petersburg pp. 210–215 (2013)
8. Montaldo, E., Sacile, R., Coccoli, M., Paolucci, M., Boccalatte, A.: Agent-based enhanced workflow in manufacturing information systems: the makeit approach. J. Computing Inf. Technol. (10) (2002)
9. Ouelhadj, D., Hanachi, C., Bouzouia, B.: Multi-agent architecture for distributed monitoring in flexible manufacturing systems (fms). ICRA 2000 proceedings pp. 2416–2421 (2000)
10. Paolucci, M., Sacile, R.: Agent-based manufacturing and control systems : new agile manufacturing solutions for achieving peak performance. CRC Press, Boca Raton, Fla. (2005)
11. Puik, E., Moergestel, L.v.: Agile multi-parallel micro manufacturing using a grid of equiplets. IPAS 2010 proceedings pp. 271–282 (2010)
12. 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)
13. Telgen, D., Moergestel, L.v., Puik, E., Meyer, J.-J.: Requirements and matching software technologies for sustainable and agile manufacturing systems. INTELLI 2013 proceedings pp. 30–35 (2013)
14. Xiang, W., Lee, H.: Ant colony intelligence in multi-agent dynamic manafacturing scheduling. Engineering Applications of Artificial Intelligence 16(4), 335–348 (2008)