Decentralized Autonomous-Agent-Based Infrastructure for Agile Multiparallel Manufacturing

Leo van Moergestel, Erik Puik and Daniël Telgen
Dep. of microsystem technology
Faculty of Science and Technology
Utrecht University of Applied Sciences
Utrecht, the Netherlands
Email: leo.vanmoergestel@hu.nl

John-Jules Meyer
Intelligent Systems Group
Dep. of Information and Computing Sciences
Utrecht University
Utrecht, the Netherlands
Email: jj@cs.uu.nl

***Abstract*—This paper describes an agent-based software infrastructure for agile industrial production. This 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 underlying systems uses two kinds of agents: an agent representing the product and an agent representing the equiplet.**

I. 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 transition time from product development to production should be minimal and small quantity production must be cheap. To fulfill these requirements we need to develop new production methods. This 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 easily configurable [13]. 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 this concept is that we need a production system that is capable of producing many different products in parallel. This is what we call multiparallel manufacturing. The software infrastructure for such a production grid is highly responsible for this agile and diverse way of production. In this paper we will propose a software model based on agent technology. Though we based our model on our own designed production hardware, we expect this agent approach to be useful in other production environments.

II. Equiplet-Based Production

The basic production platform is the equiplet. An equiplet has a standard basis and can be equipped with one or more frontends. These frontends give the equiplet the capability of production. This means that 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. These production steps are needed to build microdevices with a three dimensional structure. An equiplet has a local computer for running control software depending on the applied frontends. The equiplets in a grid do not necessarily have the same frontend. Some frontends are unique, some frontends are available on several equiplets. The equiplets are connected by a transport system to move the products from equiplet to equiplet and to serve as a temporary storage for unfinished products during the production.

III. The Standard Automation Approach

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. A lot of the same products are produced.

A. Standard Automation Software

The software for standard production systems is based on a layered model [15]. This model is mostly referenced as the automation pyramid (figure 1).

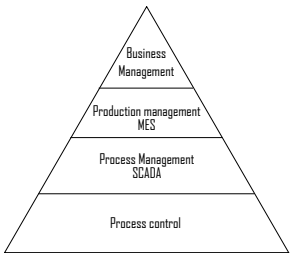

Fig. 1. Automation pyramid

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 connection with clients, suppliers, etc. is handled.
- The production management layer software is mostly covered by software systems called MES [9]. MES is an abbreviation of Manufacturing Execution System. This software enables high level control over production facilities in a broad sense.
- The 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 [16].

- The process control layer is responsible for the actual production process itself. In an automated environment it is the software that controls all kind of actuators like motors, heating devices, robot arms etc.

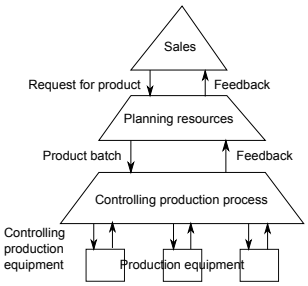

Fig. 2. Layers in the automation pyramid at work

We will use figure 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 resources needed to make the product and checks for availability. 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) when the product batches are ready for shipment. This description is a simplification, but gives an idea about the production process as a whole.

B. Properties and problems of Standard Automation

In this section we will describe two problems of standard automation that make it inadequate for agile production. One problem is switching to a different product. Standard production automation is designed for producing large quantities of the same product. Normally these products are produced in batches. 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 production supply fill up. 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 big, but still some time is required. This time is used for preventive maintenance of the production equipment, for filling up component trays, etc. A switch of type two takes a longer time because in addition to the normal batch-switch time we also need time to reconfigure the software on the lower two layers and perhaps the hardware of the lowest layer. As a consequence batch switching of this type introduces a lot of overhead.

Another problem is the *time-to-market*. 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 3.

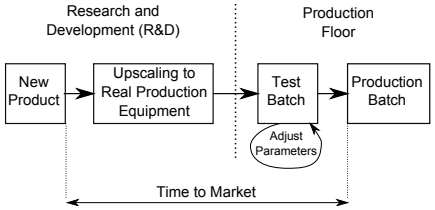

Fig. 3. Steps involved for a new product

Summary of properties of standard production automation:

- huge batches for cost-effective production;
- small overhead introduced by batch switching;
- large overhead introduced by switching to another product;
- hard transition from product development to product production;
- most SCADA and MES implementations are not suited for decentralization.

C. Solutions Offered by Equiplet-Based Production

To produce small scale batches or even unique single products, standard batch production automation is inadequate because of the properties mentioned and summarized in the previous subsection. When we use the concept of equiplets one should think of multiple production systems capable of producing a lot of different products in parallel. At any moment we can start the production of a new or different product. In the following sections we will explain in more detail how the equiplet-based approach will result in cheap small quantity production without batch switching overhead.

To make the transition from product development at the Research and Development department much easier, we use the equiplets in the product development as well as in the final production process. So we combine development, production automation and testing. This alleviates the aforementioned time-to-market problem. In figure 4 this approach is visualized.

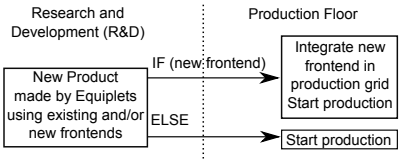

Fig. 4. Developing a new product using equiplets

IV. Software Infrastructure

Before we investigate possible software architecture solutions, let us first take a closer look at the hardware model and the basic software for the equiplets.

A. Hardware

Figure 5 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. These equiplets are connected by a standard (fast) ethernet infrastructure based on switches. These switches are standard hardware data communication devices that connect all attached systems. To monitor the grid we have a system that is a standard personal computer. The storage of production information and necessary software components and the control of the several parallel production processes is done by a central server system.

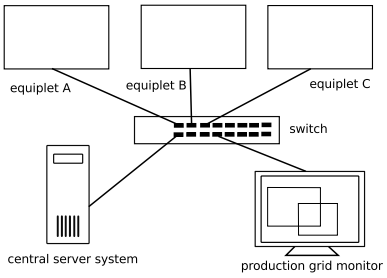

Fig. 5. hardware infrastructure

B. Software Infrastructure Requirements

Considering the software infrastructure we come to these requirements:

- efficient use of the equiplets. We need 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;
- small-scale production in parallel;
- the time to market of a newly developed product should be minimal.

To give an idea how the production of three products will look like, we have plotted so called production paths that these products will follow along certain equiplets (figure 6). 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. The plotting of the paths of three products results in a fabric of production paths along the available equiplets. The first production step is shown as a black circle.

C. Possible Software Architectures

To meet the requirements of our software infrastructure we can apply different models or architectures:

1) a centralized system, controlling the equiplets. The equiplets have a minimal software configuration.
2) a distributed system where the on-board software of the equiplet is 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.

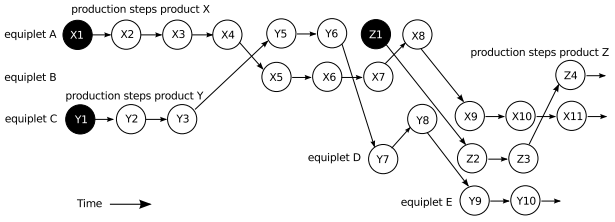

Fig. 6. Product path fabric

Solution 1 has the advantage that the equiplet software is kept to a minimum, but it results in a complex multi-threaded software system running on the central server. This server software will be a single point of failure. Solution 2 should give a better uptime because the local software on the equiplets could continue the production process in case of temporary failure in the central system or in one of the equiplets. We also use the available processing power in the equiplets and this will scale better when we add more equiplets. 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 all kind of different software tasks to perform certain production steps to the equiplets;
2b a system composed by autonomously operating parts, working together on the production tasks as depicted in figure 7. Every autonomous part is controlling the production of a single product.

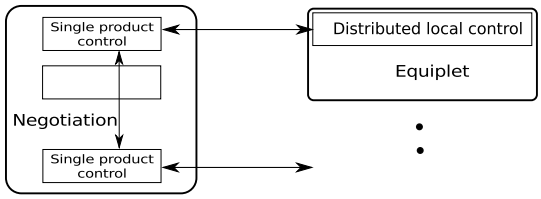

Fig. 7. Distributed production control by autonomous subsystems

In case of a coherent distributed software system, we still need a complex software system. This complex central server software system has its drawbacks:

- hard to maintain;
- single point of failure for the whole production process;
- not easy to scale;
- not easy to adapt to new situations, in other words not agile.

However, in the grid of equiplets it is easy to separate the production of product X from product Y. An autonomous software component or subsystem should be able to make a single product. The set of these product-making software entities only share the resources or environment, but have their

own path along the equiplets. By using these autonomous entities we get a flexible, scalable and reliable production system as we will show in the next sections. The autonomous software entities must fit these requirements:

- autonomy (as already stated);
- cooperative (the entities represent a production team);
- communicating (for good cooperation);
- reactive (acting on events in the production);
- pro-active (have motivation and ways to reach a goal).

This concept also fits the best in the grid concept [13], because autonomous software entities with communication capabilities can be easy implemented on a grid infrastructure. 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.

V. Agents

There are many definitions of what an agent is. We use here a common accepted definition by Wooldridge and Jennings [18] *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 8 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.

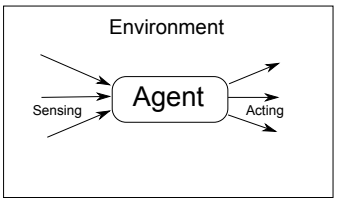

Fig. 8. An agent in its environment

In figure 8 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 some aspects that are important for our final software architecture. Literature and papers about agents introduce among others, two types of agents that seem to fit in our software solution:

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 [6] [2]. An internal schematic of a BDI-agent can be seen in figure 9 [17].

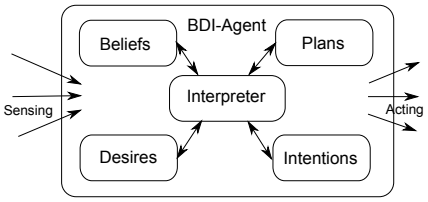

Fig. 9. BDI-agent

The beliefs, desires and intentions could be viewed as the mental states of a BDI-agent.

- 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 agents preferences;
- *intentions* characterize the 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 of the plan;
2) the precondition of the plan;
3) the course of action to carry out.

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

A. Multi-Agent Systems

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

- *role*: what is the role of a certain agent in a multi-agent system. Perhaps an agent has more than one role;
- *permission*: what are the constraints the agent is tied to;
- *responsibility*: i.e. the responsibility an agent has in achieving the global goal;
- *interaction*: agents interact with each other and the environment

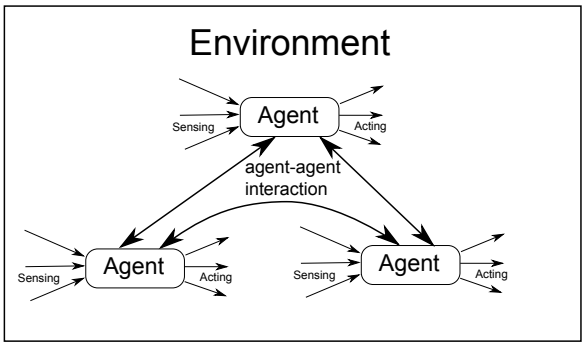

Fig. 10. Multi-agent system

When we map our system requirements 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.

VI. Multi-Agent Production System

This section describes the multi-agent production system and consists of two parts. In the first part we look at the possibilities we should consider. In the second part we describe the implementation.

A. Design considerations

To realize our system we define two roles. These roles are the main roles in the system.

1) making a certain product by searching and using a set of production steps;
2) offering and performing production steps.

We ascribe these roles to two agents. 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 [8], though there are some important differences that we will discuss later. The product agent has the purpose of the product being produced. The purpose of the equiplet agent is to accomplish 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. Because of the complex environment and the many possibilities to reach their goals, both types of agents should have the capability to deliberately choose a plan. The BDI-type of agents meets this requirement.

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. $< \sigma_4, \sigma_7, \sigma_2, \sigma_1 >$.

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 [4] or shared tuple space [7]. Then they wait until a product agent wants to use its service.

When we want to implement the product agent, we have two choices:

1) a product agent can run on the central server and make the transition to other equiplets by using another network address to reach the on-board equiplet agent as in figure 11.

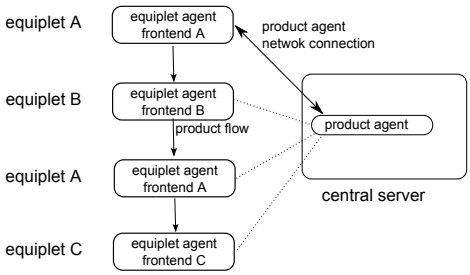

Fig. 11. Product agent on central server

2) a product agent can be mobile and migrate to the on-board equiplet hardware to interact with the equiplet agent as in figure 12;

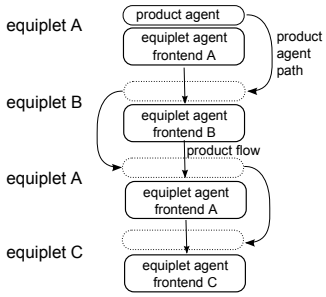

Fig. 12. Mobile product-agent

The central server based approach is easier to implement, but this solutions does not scale well. When we add more equiplets, both the server as well as the communication channels will be overloaded. Mobile agents are harder to implement but use the onboard computer system of the equiplet as a platform so this solution makes a better use of the distributed processing power. On the other hand we have to consider the amount of data communication when all these mobile agents are traveling along the equiplets (figure 13a). 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 13b).

Production steps can be in line so our path is a single thread but one could think of other possibilities. Maybe a set of steps could be replaced by another set of steps. Figure 14 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 case is a structure that starts with more parallel threads of steps. In this case it is possible to use real parallel production steps that are joined to complete

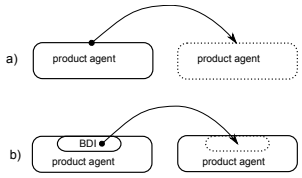

Fig. 13. Mobile agent implementations

the product. A special case of this situation is combining two half-products or so called half-fabricates.

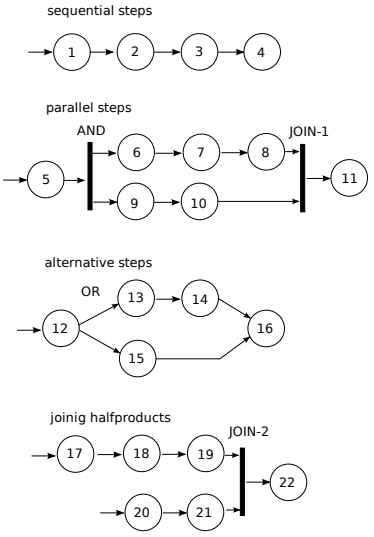

Fig. 14. Different combinations of production steps

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 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.

Based on the BDI architecture [14], it is rather straightforward to construct the basic model of a product agent in case of a single step path. Most BDI agents are capable of acting to achieve its intentions [5]. The ultimate desire or goal of such an agent is the product being completed.

In this paper we do not go into detail about scheduling though it is of course a very important part of the final realization. Figure 15 shows the way scheduling is accomplished by the participating agents. 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. The scheduling could be realized in a multi agent negotiation setup between the product agents.

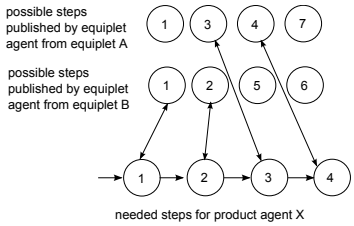

Fig. 15. Selecting a set of production steps

B. implementation

For the implementation we setup a hardware infrastructure as in figure 5. 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 [1] as a platform. The reasons for choosing Jade are:

- the simulation 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;
- in this first approach at least the equiplet agents are not that intelligent that we need special multi-agent 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 [5] or Jadex [1]. Both 2APL as well as Jadex are based on Jade but have a more steep learning curve for Java developers;
- 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. In figure 16 the Jade platform environment is depicted.

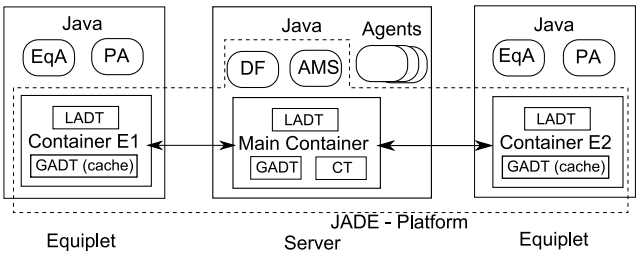

Fig. 16. The Jade platform

The Jade platform itself is in this figure surrounded by a dashed line. It consists of the following components:

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

The equiplet agents (EqA) and product agents (PA) run on top of this platform. Due to performance reasons, we decided to use our database-based blackboard instead of the DF of the Jade platform. We will now take a closer look at the agents.

1) 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 product 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 behavior 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 behaviors 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 wil 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.

2) 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,

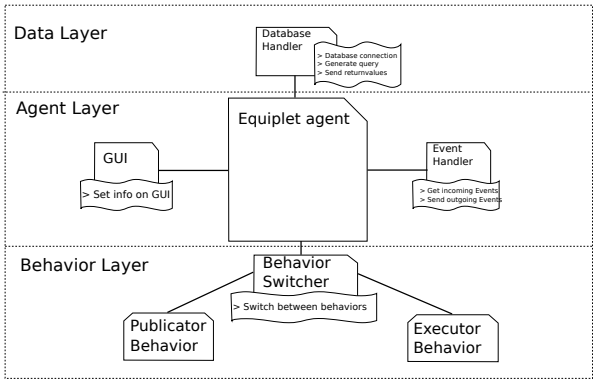

Fig. 17. Equiplet agent architecture

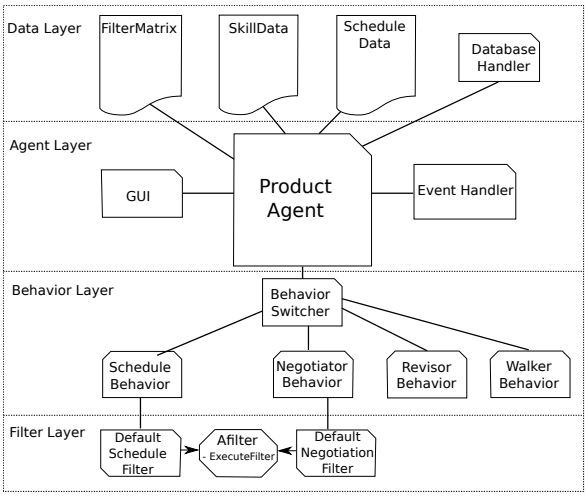

Fig. 18. Product agent architecture

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 behavior (it walks along the equiplets). The product agent is a mobile agent, walking from equiplet to equiplet as depicted in figure 13a. During walker behavior,

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 negatiate with and it will ask these production agents with walker behavior to negotiate about a certain production step. If a walker agent is willing to revise its scheduling it will enter the role of revisor and try to revise its scheduling thus making place for the step needed by the agent with the failed scheduling. Several scenarios have been build to test this concept and it works to our expectation [10].

VII. 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 [12] give an extensive overview of what has been done in this field. This work focuses on simulation as well as production scheduling and control [11]. The main purpose to use agents in [12] 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 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.

Bussman and Jennings [3] [8] 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 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 met 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.

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 multi-parallel production. The design and implementation of the production platforms and the idea to build a production grid can be found in Puik [13].

VIII. Conclusion

We have a co-design of production platforms and production software. Using agent technology came up as the best solution to implement the software infrastructure of our production grid. By introducing the concept of a product agent, creating a product agent will result in the product being produced. Because our approach integrates readily with web technology, new trends like e-Manufacturing are easy to implement in this model.

When we compare our approach with the existing solutions for production automating we have better scalability as well as agile, multiparallel production. Though we expect a lot of advantages, we must admit that this concept is not yet proven technology. A Jade framework agent based distributed simulation has already been developed and works according to our expectation [10]. We should build a prototype with equiplet agents that are capable of perfroming real production steps and do research to give a proof of concept. This is left as future work.

References

[1] N.R. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni. *Multi-Agent Programming*. Springer, 2005.

[2] M.E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, Mass, 1987.

[3] S. Bussmann, N.R. Jennings, and M. Wooldridge. *Multiagent Systems for Manufacturing Control*. Springer-Verlag, Berlin Heidelberg, 2004.

[4] 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.

[5] M. Dastani. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[6] D.C. Dennett. *The Intentional Stance*. MIT Press, Cambridge, Mass, 1987.

[7] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985.

[8] N.R. Jennings and S. Bussman. Agent-based control system. *IEEE Control Systems Magazine*, (Vol 23 nr.3):61–74, 2003.

[9] J. Kletti. *Manufacturing Execution System - MES*. Springer-Verlag, Berlin Heidelberg, 2007.

[10] 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, 2010.

[11] E. Montaldo, R. Sacile, M Coccoli, M Paolucci, and A Boccalatte. Agent-based enhanced workflow in munufacturing information systems: the makeit approach. *J. Computing Inf. Technol.*, (10), 2002.

[12] 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.

[13] E. Puik and L.J.M. van Moergestel. Agile multi-parallel micro manufacturing using a grid of equiplets. *IPAS 2010 proceedings*, pages 271–282, 2010.

[14] A. S. Rao and M. P. Georgeff. Decision procedures for bdi logics. *Journal of Logic and Computation*, 8(3):293–343, 1998.

[15] J.E. Rijnsdorp. *Integrated Process Control and Automation*. Elsevier Science Publishers, 1991.

[16] R.I. Williams. *Handbook of SCADA systems*. Elsevier Science Publishers, 1992.

[17] M. Wooldridge. *An Introduction to MultiAgent Systems, Second Edition*. Wiley, Sussex, UK, 2009.

[18] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, (10(2)):115–152, 1995.