FAKULTÄT FÜR !NFORMATIK

Implementation and Coordination of Multi Agent Systems for Production Automation Simulation using Coordination Patterns

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften
(Mag.rer.soc.oec.)

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Clemens Gondowidjaja
Matrikelnummer 0126225

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: Univ.Prof. Dipl.-Ing. Dr. Mag. Stefan Biffl
Mitwirkung: Univ.-Ass. Mag. Thomas Moser

Wien, Oktober 2008

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Acknowledgement

I would like to thank the IFS institute for providing me the possibility to write this thesis. Especially I want to thank Mag. Thomas Moser and Dipl.-Ing. Dr. Stefan Biffl for supporting me at any time they could.

Furthermore I want to show gratitude to my colleagues Dindin Wahyudin, Uwe Szabo and Klemens Kunz for their time and ideas during many discussions.

Last put not least I want to thank my girlfriend Verena and my family for supporting me during my study period.

Eidesstattliche Erklärung

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

_____

Clemens Gondowidjaja, 0126225

Abstract

Production automation, e.g., for assembling more complex products from simpler parts, is a manufacturing process that coordinates a range of hardware entities (such as robots and transport systems) to achieve predictable system behaviour. Designers of production automation systems have many parameters to influence the properties of the overall production system, such as effectiveness and efficiency of the production automation system, which are hard to predict and costly to try out with real-world hardware. Multi-agent systems allow representing the hardware entities, their behaviour, and communication in a simulation environment, which facilitates efficient evaluation of design and parameter choices in order to optimize system performance.
Coordination describes the behaviour of a group of agents to optimize a known goal function while observing a set of constraints. A major challenge is to model the coordination of agent groups as agents may belong to several agent groups at one point in time or change group affiliation over time.

This thesis introduces coordination patterns that allow describing the coordination of agents on several levels in the context of a multi-agent simulation of a production automation workshop. The simulation consists of two major layers: 1. the business (or production planning) layer where the dispatcher transforms customer orders into more detailed work orders, which describe the machine functions in the workshop to carry out in order to fulfill a customer order, and prioritizes the work orders in order to optimize the overall system performance; and 2. the technical (or production automation) layer where the multitude of agents that represent hardware entities cooperate to carry out the current set of work orders coming in from the dispatcher.

Another aim of the development of the simulator and its coordination component is to test and analyze different workshop scheduling strategies which can me mapped to a real-world manufacturing plant.

Key contributions of the thesis are:
An architectural overview and guidelines for designers of multi-agent systems for tasks those are similar to production planning and automation. Coordination patterns describing the coordination of the production planning and automation layers in order to optimize the overall system performance. Design and implementation of a coordination component to simulate complex production processes and analyze the different production strategies. Design and implementation of a test case management system to run test scenarios with the simulator and measure the results for performance evaluation. A systematic evaluation of the performance of a range of production strategies and coordination patterns.

Major results of the work are:
1. The coordination with advanced production strategies is significantly more efficient than local optimization of agent behaviour; 2. The definition of guidelines for the development of a coordination component for agent-based production systems; 3. The development and

implementation of a general interface, based on coordination pattern, between the business layer and the workshop layer for manufacturing system similar to the one used; and 4. Measurement of various workshop scheduling strategies based on the simulation results.

Abstract

Produktionsautomatisierung, beispielsweise die Montage von komplexeren Produkten aus verschiedenen Subprodukten und Ressourcen, ist ein Fertigungsprozess bei dem eine große Anzahl an Entitäten, wie z.B. Roboter oder Maschinen, koordiniert werden müssen. Bei der Planung solcher produktionsautomatisierten Systeme müssen Designer gewisse Parameter, die die Eigenschaften des gesamten Produktionssystems beeinflussen, beachten. Diese Parameter auf einem „Real-world"-Produktionssystem zu testen braucht im Normalfall eine hohe Anzahl an Ressourcen und ist damit schwer zu prognostizieren. Multi Agenten Systeme erlauben es diese Entitäten darzustellen, ihre Eigenschaften und ihr Verhalten zu definieren und die Kommunikation innerhalb einer vorgegebenen Umgebung zu simulieren.
Koordination beschreibt das Verhalten einer Gruppe von Agenten unter bestimmten Umständen. Dies führt zur Optimierung von definierten Funktionen und Zielen. Die Koordination von Agenten aus verschiedenen Gruppen bzw. Gruppenzugehörigkeiten zu verwalten ist eine besondere Herausforderung.

Diese Diplomarbeit beschäftigt sich mit Koordinationsmustern welche die Koordination der Agenten innerhalb eines Multi Agenten Simulator für automatisierte Produktionssteuerung auf mehreren Ebenen beschreibt. Die Simulation besteht aus zwei Hauptteilen: 1. Die „Geschäftsprozessebene" (oder Produktionsplanung), wo ein Lastverteiler Kundenbestellungen in detaillierte Arbeitsschritte einteilt. Diese Arbeitsschritte beinhalten Maschinenfunktionen innerhalb des Fertigungssystems und legen Prioritäten fest, welche die Gesamtleistung des Systems steigern sollen. 2. Die „technische Ebene" (Produktionsautomatisierung), wo eine Gruppe von Agenten miteinander kooperiert um die Arbeitsschritte des Lastverteilers zu erfüllen.

Ein weiteres Thema dieser Diplomarbeit ist das Testen und Analysieren von definierten Produktionsstrategien. Die Simulationsergebnisse können nach der Analyse auf „Real-world" Produktionssysteme abgebildet werden.

Beiträge dieser Arbeit:
Diese Arbeit beinhaltet einen architektonischen Überblick und definiert Richtlinien für Designer von Multi Agenten Systeme für Produktionsautomatisierung. Dazu werden Patterns verwendet um die Koordination der Produktionsplanung und Produktionsautomatisierung zu beschreiben. Ziel ist es die Gesamtleistung des Systems zu steigern. Weiters wird eine Koordinationskomponente entworfen und implementiert. Diese wird eingesetzt um komplexe Produktionsprozesse zu simulieren und Produktionsstrategien zu analysieren. Das Agenten System und seine Koordinationskomponenten werden mittels Testszenarien getestet. Hierzu wird ein Test Management System entworfen und implementiert. Die dadurch gewonnen Simulationsdaten werden analysiert, um eine systematische Leistungsauswertung von definierten Produktionsstrategien zu erhalten.

Ergebnisse der Arbeit:

1. Die Koordination von weiterentwickelten Produktionsstrategien ist effizienter als eine lokale Optimierung der Agenten.
2. Die Definition von Richtlinien für die Entwicklung einer Koordinationskomponente für Produktionssysteme basierend auf ein Multi Agenten System.
3. Die Entwicklung und Implementierung einer generellen Schnittstelle, basierend auf Koordinationsmustern, zwischen der Geschäftsprozessebene und der Betriebsebene für ähnliche Fertigungssysteme.
4. Analysieren und Messung von Produktionsstrategien basierend auf Simulationsergebnissen.

Table of contents

List of Tables

List of Figures

List of Listings

1. Introduction

Production automation, e.g., for assembling more complex products from simpler parts, is a manufacturing process that coordinates a range of hardware entities (such as robots and transport systems) to achieve predictable system behaviour. Designers of production automation systems have many parameters to influence the properties of the overall production system, such as effectiveness and efficiency of the production automation system, which are hard to predict and costly to try out with real-world hardware.

Multi-agent systems allow representing the hardware entities, their behaviour, and communication in a simulation environment, which facilitates efficient evaluation of design and parameter choices in order to optimize system performance.

Coordination describes the behaviour of a group of agents to optimize a known goal function while observing a set of constraints. A major challenge is to model the coordination of agent groups as agents may belong to several agent groups at one point in time or change group affiliation over time. This thesis describes the design, the evaluation, and the improvement of software engineering processes, methods and models to make (the engineering of) reconfigurable software-intensive systems in production automation more effective, efficient, robust, and usable. The used MAS and the associated coordination components represent such a system for production automation.

Production automation systems consist of two major parts, the manufacturing support system and various facilities (e.g., production machines or material handling equipment) that actually carry out the manufacturing tasks. The manufacturing support system is used to manage production and to solve technical and logistic problems (e.g., ordering materials or machine utilization) that can appear during the production. Facilities represent the factory and all its equipment (for example conveyor belts or assembly machines) needed to organize fulfilling a costumer’s orders. The production processes include several products (that get assembled from simpler products) with individual complexity and attributes that directly influence the production steps.
Another important part of production automation is the factory personnel. Without human beings production can not be realized because not every production step or process can be automated. But since the main focus of this thesis is on the automation steps, these semi automated steps are not taken into consideration.

The overall production system usually includes a huge number of parameters to influence the properties which influence the effectiveness and efficiency of the production automation system. These properties are monitored by control systems that manage the manufacturing systems. Such systems reduce the service activities done by human beings. Such automated processes include:

- Automated assembly machines
- Robots that conduct working steps

- Automated inspection systems for quality control
- Routing mechanism
- Sorting machines

The various simulated entities can be configured with a range of parameters depending on their responsibilities and abilities. The automation of such manufacturing entities can increase labour productivity, reduce costs, reduce labour shortages, improve worker safety and product quality, reduce manufacturing lead time, reduce unit costs, and accomplish processes that cannot be done manually. Beside these advantages, there are also some challenges against automation. Some tasks can be realized only by humans; Table 1 compares the abilities of humans and automated machines.

Relative strengths of Humans	**Relative strengths of Machines**
• Sense unexpected stimuli • Develop new solutions to problems • Cope with abstract problems • Adapt to change • Generalize from observations • Learn from experience • Make difficult decisions base on incomplete information	• Perform repetitive tasks consistently • Store large amounts of data • Retrieve data from memory reliably • Perform multiple tasks at the same time • Apply high forces and power • Perform simple computations quickly • Make routine decisions quickly

Table 1. Humans versus Machine Strenghts (LeMaster, 2001) [1].

The simulator used in this project is based on Multi-agent systems (MAS) and has its roots in the Distributed Artificial Intelligence (DAI) domain. The automated production system simulated during the project consists of various distributed agents, which act as community to solve certain problems. Such problem solving communities are based on the DAI research field. To simulate the various manufacturing processes and the used entities, MAS can be implemented.

MAS allow representing the hardware entities, their behaviour, and communication in a simulation environment, which facilitates efficient evaluation of design and parameter choices in order to optimize system performance. Therefore various agents/units are implemented for each entity in such a production planning system. Each agent is heterogeneous, which is reflected in the different behaviours it can have, and can manage its activities based on its local state and the information exchanges its associated agent group. These agents can be components of a system and can interact as entities which represent different parties with interests and interactions. In manufacturing processes there are different machines or transport systems that have various local problems to solve, but still try to achieve a common goal. This is one of the core abilities of a MAS. Therefore the agents have to coordinate and communicate with each other. For example a defect assembly machine has to inform a neighbouring crossing if it is not reachable. In a “real” factory a machine that encounters a serious problem has to send a message to its neighbours to prevent follow-up failures.

A model of the assembly workshop used as working example in this work is situated at ACIN/TUW; the software simulator has been implemented based on a production system simulation kit from Rockwell Automation International Research situated in Prague. This simulation includes production system agents like docking stations, machines, conveyor belts, crossings, or sensors. The coordination between the agents is realized using the Agent Communication Language (ACL) [1].
This work includes an extension of the Rockwell simulator with coordination agents, which represent the interface between the business layer (order dispatching) and the technical layer (workshop floor simulator).
The MAS technology is used in this work to study dynamic scheduling strategies for production systems. In addition, parallel machine scheduling strategies are tested. The parallel machine scheduling problem is defined as a production system that has to fulfil *n* tasks that underlie a certain number of conditions on *m* machines. The production system can be defined as a closed-queuing transfer network with redundant paths. During a simulation run different mechanism, like load balancing, can influence the production.

Coordination is the systematically process of analyzing a group of agents to reach a known solution that depends on a set of constraints. As the various agents act depending on constraints like work orders or system utilization, they have to coordinate to achieve the overall common goal. Therefore a coordination component needs to have a global overview of the system and has to decide which production or management steps should be done next. These decisions are made by the "dispatchers", which need to have an overall knowledge of the system. The coordination of the agent-based system is defined with coordination patterns that provide a reusable guideline for designer of such distributed systems. The coordination patterns describe the hierarchical organisation of the agents within the system and the way they communicate with the business layer, where the dispatcher acts, and within the workshop layer, where the various manufacturing entities fulfil production steps. So the patterns form the design interface between the business layer and the workshop layer.

This thesis introduces coordination patterns that allow describing the coordination of agents on several levels in the context of a MAS simulation of a production automation workshop. Patterns can be found in different domains of every day life. Like for textiles or buildings (architecture), it is possible to define patterns for software systems [2]. Software patterns are prototypes for a software solution with a given problem and context. The most important components and parts of the systems are reflected in such patterns in order to allow comprehensible reconfiguration and reuse. The pattern used in this thesis describes the communication between the major parts of the system: the business layer and the technical layer.
The business (or production planning) layer is responsible for the business processes that should be simulated on the simulator. Here a dispatcher transforms incoming customer orders into more detailed work orders, which describe the machine functions in the

[1] www.fipa.org

workshop to be carried out for fulfilling a customer order, and prioritizes the work orders in order to optimize the overall system performance. These work orders are forwarded to the various simulation agents in the technical layer.
The technical (or production automation) layer is represented by the simulator. The simulator consists of various agents that represent hardware entities cooperate to carry out the current set of work orders coming in from the dispatcher.
The interaction between the business layer and the technical layer is monitored by the coordination interface which is designed and implemented as part of this work. To guarantee the reusability of the interface it has to be described in an abstract and general way. This is realized by the coordination pattern. The reusability allows exchanging the technical layer without the need to change the interface or the business layer.

The implemented systems also help to analyze and optimize various workshop scheduling strategies. Optimization means modifying a process or system to get the optimum performance. The goal of most companies is to optimize their processes, like the system throughput or the distribution of resources. This usually means to increase the profit by modifying a production system for example. In the production automation process for instance the number of products produced in one shift can be optimized by using different scheduling strategies. Before the optimization an analysis has to be done to know where and how processes can be improved.
Optimization does not mean that systems will work optimally afterwards, because typically after an optimization one part of the system will work more efficiently but this may be at the expense of the overall system performance. For example if one increases the quality control to produce high-quality products, this will also result in an increase of production time. So a trade off between the different attributes and measurements has to be found. Here an analysis has to point to the part with the highest need for optimization. Besides calculating the outcome of an optimization it is important to check the costs required to optimize the process. The simulation results can be mapped to a real-world hardware manufacturing plant, in this case in the Odo Struger lab[2] at the ACIN, TU Vienna.

Key contributions of the thesis are:

1. **An architectural overview and guidelines for designers** of MASs for tasks which are similar to production planning and automation. As such production planning systems are very complex and hard to optimize these guidelines provide a reusable technique for a configuration of such a system.

2. **The development of a coordination pattern**, which coordinates the production planning and automation layers, in order to optimize the overall system performance. The pattern provides a reusable solution for an interface between a coordination part and the simulator. The coordination with advanced production strategies is significantly more efficient than local optimization of agent behavior.

[2] http://www.acin.tuwien.ac.at

3. **Guidelines for deriving a concrete design and implementation** from a set of a coordination pattern, simulator technology, and production automation context. Designers of distributed systems with structure similar to the one used in this thesis will get an overview of the architecture of such agent-based systems.
This coordination component is the interface between the business layer which controls the incoming orders and the job shop layer which represents the simulator. The interface is reusable for various simulator layouts and is extendable and can be adapted for different use cases. It provides a solution for designer to configure an automated production planning system.

4. **The design and implementation of a coordination component** simulating complex production processes and analyzing the different production strategies. These strategies are independently implemented and represent a separated component in the system. This results in a simulation environment that provides an easy exchange of strategies without much effort. During the thesis various strategies are defined and integrated into the production system.

5. **A coordination component which monitors the agent-based system** and represents the communication medium between the dispatcher and the MAS. So it is the interface between the business layer and the workshop layer and is described in Coordination Patterns. This modular architecture allows flexible exchange of either the business layer or the MAS.

6. **The design and implementation of a test case management system** to run test scenarios with the simulator and measure the results for performance evaluation, and a systematic evaluation of the performance of a range of production strategies and coordination patterns. The test case management component is implemented separately from the simulator and the coordination component. It can be extended and exchanged without effecting the production planning system. The simulator provides an interface for the test case management component where the data and information needed for the analyses can be accessed.

7. **A systematic evaluation of the performance of a range of production strategies and coordination patterns.** We evaluated the system in a simulator that represents the test bed in the Odo-Struger laboratory[1] (Automation and Control Institute (ACIN); Vienna University of Technology). Based on the simulation results, we analyze the performance of various workshop scheduling strategies.

The remainder of this work is structured as follows: Chapter 2 summarizes relevant related work; Chapter 3 defines the research issues derived from the related work. Chapter 4 gives an overview of the practical work done for this thesis. This is the main part of the work and describes the agent system used for the simulator, the coordination patterns and the simulator. Chapter 5 takes a closer look on the results of the thesis and compares these

results with the research issues from chapter 3. Chapter 6 discusses the work with related work; Chapter 7 concludes the thesis and suggests further research work.

2. Related work

The related work chapter gives an overview of the techniques and research areas touched in this work. The project application is based on a Multi Agent System (MAS) including a certain number of agents. Therefore the chapter starts with an overview of Software Agents and describes one possible organizational structure of them, the Multi Agent System. The next lines will include the main part of this thesis, the coordination of an agent-based system. This coordination component can be described in coordination patterns, which are the subject of the next part.

2.1 Software Agents

The simulator used in the project is based on MAS. A MAS is an organizational structure of two ore more software agents which fulfil certain tasks. This section starts with the principals of software agents; next the various types are summarized. The agents within the implemented systems can be classified as collaborative but there are several other agent types which will be described during this section.

2.1.1 Principals

A software agent is an artificial component that operates in a software environment. It is a software component which is authorized to fulfil a task on its own and decide which action is appropriate for solving a certain problem.
Software agents can be used in multi agent system in order to solve complex problems. So they are a part of the Distributed Artificial Intelligence (DAI) research field. Therefore they provide modularity, speed and reliability.

Agents can be used for different problem solving situations and can be divided into groups depending on their attributes and entities. Nwana [3] classifies the agents using five characteristics:

- Mobility: Agents can be moved around different networks. They are either static or mobile.
- Deliberative or reactive: Deliberative “agents posses an internal symbolic, reasoning model and they engage in planning and negotiation in order to achieve coordination with other agents” [3]. Reactive agents have the ability to response to the present state of the environment in a selective way.
- Roles: Agents can adopt different roles with various abilities. For example agents in the World Wide Web do often act as information retrieval component.
- Hybrid: The combination of different types of agents combine the benefits of each individual agent.
- Several ideal and primary attributes: The basic principles of agents are that they act autonomously, they learn and cooperate. Autonomy means that they can act on their own without being controlled by a human being. An important attribute for their autonomy is the fact that they can take the initiative, so they can work proactively. The cooperation plays one important role in agent-based systems. To

reach a common goal the agents have to communicate with each other. So they need a kind of social community in order to coordinate their action and inform other agents about their own state.
The ability to learn is another key attribute of any intelligent being. With the help of these three basic attributes it is possible to group different agents depending of their attributes, like in figure 1:

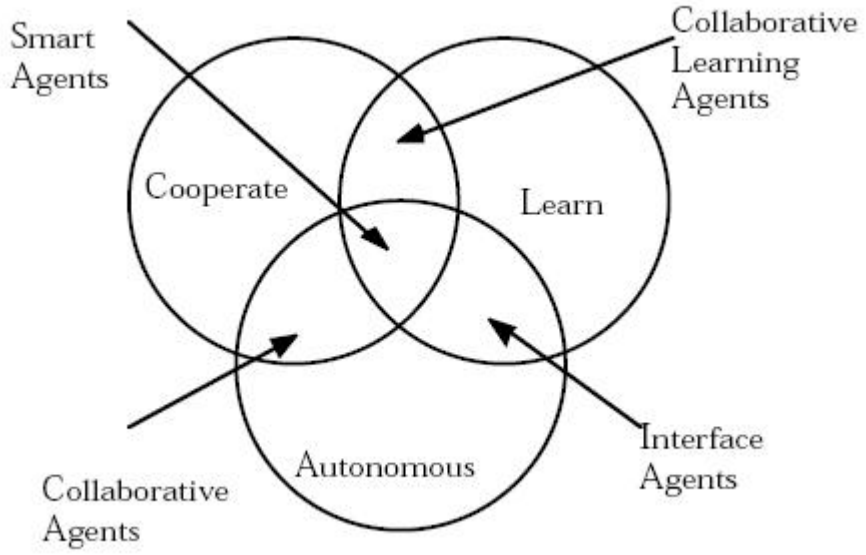


figure 1: Typology based on Nwana's [3] primary attribute dimension.

2.1.2 Agent Types

As already mentioned the used agents in the project are mostly collaborative. Besides this kind of agents Nwana [3] identifies six further types of agents:

- Collaborative agents
- Interface agents
- Mobile agents
- Information/Internet agents
- Reactive agents
- Hybrid agents
- Heterogeneous agent systems

Collaborative agents
Collaborative agents are mostly autonomous and cooperative in order to fulfil the tasks of their owners, as shown in figure 1. The learning aspect is not the main prospect of these agents.
Such agents can be used if one or more of the following attributes are covered:

- The problem to be solved is too large for a centralized single agent. This can be applied if the resources are limited for example.
- Interconnection and Interoperation of multiple existing legacy systems
- Solution for a distributed problem like air-traffic control or production automation

- Provide solutions which are drawn from distributed information sources
- Increase modularity (results in complexity reduction), speed, reliability, flexibility and reusability at the knowledge level

Interface agents

Interface agents work mostly autonomously and learn from the past actions done by the user. They assist and collaborate with the user. Collaborating with a user may not require an explicit agent communication language, in contrast to the collaboration with an agent. The agents support the user while learning to use an application and provide better ways of fulfilling the tasks. Typically, the agent for itself learns to better assist the user. This can be done by one of the following four ways [4]:

- By observing and imitating the user
- Through receiving positive and negative feedback from the user
- By receiving explicit instructions from the user
- By asking other agents for advice

Mobile agents

The increasing importance of the internet resulted in the need of having agents that can browse the different wide area networks for certain information. Therefore the agents are autonomous and they cooperate. For example they have to interact with foreign hosts, exchange data or information with other agents without necessarily publicise all of their own information.

An important benefit of using mobile agents is that, like the name expresses, they do not have to be stationary. One domain is for example the flight ticket reservation over the World Wide Web. “A static single-agent program would need to request for all flights leaving between these times from all the databases, which may total more 200 and take up many kilobytes. It would also require a list of all the connections and proceed to narrow down search” [3]. Such a request would produce huge information traffic if it is done by a static-single agent. Mobile agents would just need about 2 kilobytes to search through the network of airline reservation systems, query the databases locally and return the information.

So the benefits of using mobile agents would include [3]:

- Reducing communications costs: The agents would search for the requested information locally instead of transferring information.
- Limited local resources: The users processing power and the storage of the machine can be limited.
- Easier coordination: The coordination of remotes and independent requests is easier.
- Asynchronous computing: The mobile agents can work autonomously, so the user can do something else while waiting for the results.
- Flexible distributed computing architecture
- Radical and attractive rethinking of the design process in general

Information/Internet agents

Information agents help to manage the increasing growth of information. They can manage, manipulate and collect information from different distributed sources. The difference between this kind of agents and the agents mentioned before is that information agents are defined by what they do, on the other side the collaborative or interface agent are defined by what they are.

The main goal of information agents is helping "to create a simple user interface so that information search and retrieval using information agents will become as natural as picking up a phone or reading a newspaper" [3]. To guarantee the realization of the goal the agents have to know where to look, how to find the information and how they can collect it. Information agents are very flexible. They can be static or mobile, be non-cooperative or social and they may be able to learn or not. So there is not a standard of how they have to be implemented. For example they can be part of a browser used to gather information.

Reactive agents

"Reactive agent represents a special category of agents which do not posses internal, symbolic models of their environment. Instead they act and response in a stimulus-response manner to the present state of the environment in which they are embedded"[3]. Such agents are relatively simple and they interact with other agents in basic ways. They have no explicit symbolic representation, no explicit symbolic reasoning and an emergent functionality. The benefit of using reactive agents is that they are more robust and fault tolerant than other agent-based systems. Besides they are flexible and adaptable in contrast to the inflexibility and slow response times of classical artificial intelligence systems. Reactive agent applications are often used in the games and entertainment industry. They can be found in digital video and 3-D graphic-based animations.

Hybrid agents

The agents mentioned above have different benefits and dependencies. If the system has to fulfil more than one of these abilities so that the strengths are maximized and the deficiencies are minimized the agents can be combined. Such a combination of two or more different agents is called hybrid agent. So the goal is to unify the different benefits of the individual agents.

Heterogeneous agent systems

Heterogeneous agent systems consist of two or more agents who belong to two or more different agent classes. These agents can also be hybrid agents. The increase of software products which provide distributed services for different domains results in a growth of having interoperating agents. For the coordination of agents a common communication protocol has to be added. This is done by the agent communication language or protocol (ACL).

For further information about software agents see "Intelligente Softwareagenten" [5].

2.2 Multi Agent System (MAS)

This section gives an overview of the agent technique used in the project. After a common definition and the motivation reasons of using MAS, one of the most important ability of MAS is described, namely cooperation. The end of the section will take a closer look at the limitations of MAS.

2.2.1 Definition

Multi Agent System has their roots in the Distributed Artificial Intelligence (DAI) research field and is related to the Distributed Problem Solving (DPS) system [6] [7]. It is a network including more than one problem solver. The solvers can interact with each other to find a solution to a given task which is beyond the individual knowledge and capability of each problem solver. The so called problem solvers are represented as agents.
MAS are a kind of distributed artificial intelligence. The components can be displayed as interacting entities which represent different parties with interests and interactions. This includes competition as well as cooperation.
A Multi Agent System often represents real world problems mapped into a software system. These problems are often very complex, distributed and can change dynamically and frequently. Therefore systems dealing with such problems have to be modular and abstract. A Multi Agent System shows these abilities and fulfils them by having "a number of functionally specific and (nearly) modular components (agents) that are specialized at solving a particular problem aspect" [8].

An agent within a MAS is usually defined as *autonomous*, which means that it can operate without any intervention of any other component or human being and has its own control over its actions and states. It has a kind of *social* ability, which means that it can communicate with humans or other components, and is *reactively*, so it can inform the agent about its environment and react autonomously on different events or states.
In the last few years the interest in such Multi Agent Systems increased. Especially the promises for designing, conceptualizing and implementing distributed, large, open and heterogeneous software systems are of certain interest. Therefore environments like the internet can make large profit of MAS.

In the next sub-section an overview of the advantage and a motivation for using MAS is described.

2.2.2 Motivation

To simulate distributed systems, like production automation, a lot of resources are needed. MAS can help to solve problems that are too large for centralized systems by finding bottlenecks in the performance and react on them.
Changing the business requirements is usually very expensive and the reconfiguration of the system needs a lot of time. A better solution is to map such systems onto MAS which can be exploited by other software components. Therefore an agent wrapper is needed.
Another advantage in using MAS "is to provide solutions to problems that can naturally be regarded as a society of autonomous interacting components-agents" [8].

MAS can also save costs by using distributed information sources efficiently by collecting and forwarding them to the agents which can react on the incoming messages. The usage of MAS also results in performance advancement along different dimensions [8]:

- *Computational Efficiency* because the concurrency of computation is exploited
- *Reliability* by recovering component failures
- *Extensibility* because of the flexibility of using different numbers of agents solving a problem
- *Maintainability* because of modularity
- *Responsiveness* by handling anomalies locally
- *Reusability* by using the same agents in different problem solving cases

Another advantage of MAS is the various cooperation techniques which allow organising the agent community in different ways. This subject is described next.

2.2.3 Cooperation

To solve common problems within an agent group cooperation is needed. The community has to work together and exchange information to know the state of the work of each component. Ferber [9] defines six categories of cooperation needed to organize MAS: grouping and multiplication, communication, specialisation, collaboration by sharing tasks and resources, coordination of actions, conflict resolution by arbitration and negotiation.

Grouping and multiplication

Grouping means a simple arrangement of several agents in a physical way. This cooperation does not only play an important role in the digital world. For example animals build groups to be safer against enemies. So the individuals within a group act not only for their own but also for the profit of the group.

Multiplication just means the increase of the number of agents in a group. This can result in an improvement of the productivity but can also lead to problems (e.g. communication problems).

Communication

Communication is on core part of cooperation within MAS. “In effect, communication expands the perceptive capacities of agents by allowing them to benefit from information and know-how that other agents posses” [9]. The sharing of information and states is realized by sending messages or signals to partners who coordinate their activities based on this information.

Specialisation

Agents often have to be specialised to fulfil a certain task. Within agent groups this means that the individuals can have different abilities and are more qualified for certain jobs than others. For example transport agents have other core tasks than coordination agents.

Collaboration by sharing tasks and resources
In every community where different units/agents work on a common problem, collaboration is done. It allows agents to distribute problems, information and resources. The agents have to know about the activities of the others and from which agents they are dependent.

Coordination
Coordination within MAS is one of the central issues to guarantee a community free of conflicts. The coherence between the agents is an important factor to prevent chaos during the problem solving process. To reach the common goal, which is the background of the usage of MAS, the agents have to coordinate within a group and they do not try to solve their problems and views without communicating with the others [10].
Global constraints let agents work together, so no one can go its own way for solving a problem. The community between the actors is strengthened with such bindings. For further information see chapter 2.3.1 “Coordination within MAS”.

Conflict resolution by arbitration and negotiation
If a certain number of agent works together to solve a common problem, conflicts can cause problems. Arbitration and negotiation are two techniques to solve such conflicts. Arbitration defines rules of behaviour which will lead to constraints within an agent group. Negotiation means a bilateral agreement between agents to solve conflicts.

2.2.4 Problems

The missing of global viewpoints, global knowledge and global control can lead to misunderstandings and inconsistencies. This can result in problems within the agent goals, plans, knowledge, beliefs and results. To prevent the agent systems from such disparities the appeared problems have to be recognized and resolved. One solution would be to generate an agent that has the overall knowledge of all components in the system. It would monitor all states of the agents and would know were the problems are and how they can be resolved. This approach can lead to problems by having a bottleneck at the omniscient agent.
The main approach to prevent misunderstandings and problems in MAS is negotiation. As Sycara [8] says that negotiation with in MAS is a method for coordination and conflict resolution. This can either result in goal misunderstanding by planning or task inconsistencies by determining organizational structure. Sycara [8] further characterize negotiation as “(1) the presence of some sort of conflict that must be resolved in a decentralized manner by (2) self-interested agents under conditions of (3) bounded rationality and (4) incomplete information”. For more information about negotiation in MAS take a closer look at Sycara [8]

2.2.5 MAS for Production Automation

There are several existing agent frameworks for a coordination component for production automation. Two approaches are analysed and compared during this thesis. The first one is

the ProPlanT [11] [12] which is a production planning system for TV transmitters. The agents are organised in three different groups with various responsibilities.
The second approach is a more complex agent framework by Lim and Zhang [13]. The agents are classified in execution and information agents. The system is monitored by four agent manager which can have subagents to fulfil their tasks. The more detailed description about the framework can be found in the paper by Lim and Zhang [13].

2.3 Coordination

The main part of this thesis is the implementation of a coordination component for the existing simulator. Coordination plays a central role for distributed artificial intelligence systems, like MAS.

2.3.1 Coordination within MAS

Within MAS coordination is one of the main issues to ensure that a community of agents acts in a coherent manner. Coherence means that the agents within a group follow a common goal and do not conflict with each another. It guarantees that the community works together. There are several reasons why coordination within MAS plays such an important role [14][15]:

- *Preventing anarchy or chaos*: One character of agent based systems is that they are decentralised which can cause chaos. The agents just have local views, goals and knowledge which may result in disparities and inconsistencies. To prevent such misunderstanding or failures within an agent group, they have to be coordinated.
- *Meeting global constraints*: A group of agents may have global constraints, for example a certain budget they can expend to solve a problem. To consider such constraints the agents have to coordinate their behaviour.
- *Distributed expertise, resources or information*: The agents within a community usually have different capabilities or specialised knowledge. But still they are all work together to solve a common problem although they have different sources of information, resources, reliabilities or responsibilities. Like during building a house where architects, bricklayers or plumbers have to work together, agents and their activities within MAS have to be coordinated.
- *Dependencies within agents' actions*: Actions done by agents can depend on each other. Like in the real world, where during building houses the walls have to be set up before the roofer can start their work, agent sometimes have to wait for each other.
- *Efficiency*: During a process agents may discover important information which can be important for further processes. So the agents forward such information to the others.

Usually agents communicate with each other. They distribute their goals, intentions, results and states to their agent group. Coordination is used to address several DAI and distributed computing issues, including [10]:

- Creating network coherence by maximizing the team work within an agent group.
- The distribution of tasks and resources to the agents.
- Discovering and resolving misunderstandings and conflicts in goals, facts, beliefs, viewpoints and behaviour.
- Define clear organisational structure.

The outcome of the coordination results in preventing deadlocks and livelocks during processes. Deadlocks are actions between agents which are impossible and livelocks occur if agents act without producing any outcome.

The coordination within an agent community can be managed in different ways. These various techniques are described in the next subchapter.

2.3.2 Coordination techniques

Nwana, Lee and Jennings [10] classify the coordination techniques in four different categories which will be explained in that section:

- Organisational structuring
- Contracting
- Multi agent planning
- Negotiation

Organisational structuring

Organising the agents in defined structures is rather simple. Every agent has its responsibility, capability, connectivity and control flow. Through the definition of communication paths, roles and authority relationships, the activities and interactions are provided. This results in a long-term relationship between agents where a “master” agent distributes the tasks to a certain number of “slave” agents. The hierarchical technique is implemented in a couple of ways:

- The master agent plans and allocates fragments among the other agents. The slave agents have to report the master about their work and can communicate with each other.
  While the master has full autonomy over the slaves, the slaves have just partial autonomy.
- A coordination base is integrated by a classic blackboard where agents can post and read. A scheduling agent is responsible to supervise the processes on the blackboard. For more information see section 2.4.5 “Blackboard Pattern”.

The problem in such an organisational structure is that if there is no direct agent-to-agent communication the blackboard can become a bottleneck. This would work against the defined benefits of distributed artificial intelligence like speed, reliability, concurrency, robustness, graceful degradation, minimal bottleneck, etc. So an agent based system using this coordination technique has to have homogeneous and rather small-grained agents.

Contracting

The contracting technique can be applied to distribute tasks and resources among agents and define an organizational structure. This results in a decentralized market structure and the agents can have two different roles within the system:

- *Manager*: Such agents divide a common problem into sub problems and search for contractors to solve the task. Additionally they monitor the problem's overall solution.
- *Contractor*: Contractors are fulfilling their allocated tasks, whereas they can act as managers to divide their problem into sub tasks and forward them to other agents.

Before the manager can bind contractors to a sub problem they have to announce the task. The contractors analyze the task and decide if they can fulfil it with their abilities and commitments. If the contractors are qualified to solve the problem they offer their services to the manager. The manager chooses a certain number of contractors and waits for the results. The technique is a completely distributed scheme which is organized in a self-organising group of agents. Huhns and Singh [16] suggest using contracting if:

- the application is organized in a well-defined hierarchical nature,
- the task has a coarse grained decomposition,
- dependencies within sub problems are loose.

The benefits in using such a dynamic task allocation lead to better arrangements; agents can be registered and removed dynamically. Dynamic task allocation provides distributed control and failure recovery and includes automatically a mechanism for load balancing. The problems by using contracting appear because the technique neither detects nor resolves conflicts. The agents within the contract are very passive, benevolent and non antagonistic which does not reflect real world problems.

Multi-agent planning

Another coordination technique for an agent based system is multi agent planning. It provides a structural plan of the agent group and a detailed overview of the future actions and interactions which are required during the problem solving process. It helps to avoid inconsistence and conflicts. The agent plans can either be centralized or distributed.

The centralized solution includes a coordinating agent which knows about all partial or local plans of the different agents. To avoid inconsistencies and conflicts, the coordination agent analyses, modifies and combines these plans to a multi agent plan. Within the final plan communication commands are defined to synchronise the agent's actions. If a conflict would be identified the agents would build a conflict group and initiate a negotiation process.

The distributed approach assumes that each agent has a model of the other agents plans. The communication between the agents results in updating their own plans and the models of the others until a conflict is solved. Critique for the technique is that it generates a lot of

computing and communication traffic. The distributed approach needs more capacity then the centralized one.

Negotiation

Negotiation is not only a coordination technique on its own, but it is also involved in other coordination schemes. Bussman and Muller [17] define negotiation for agent based system as:

"… negotiation is the communication process of a group of agents on order to reach a mutually accepted agreement on some matter."

Negotiation is done by agents which possess beliefs, desires and intentions of other agents in the community. With these abilities it "has led to the development of techniques for the following: representing and maintaining belief models, reasoning about the other agents` beliefs, influencing other agents` intentions and beliefs" [10].

The extensive use of negotiation can be classified into three categories:

- game theory based
- plan based
- human inspired and miscellaneous AI based

For the further information about the different approaches of negotiation see Nwana, Lee, Jennings [10].

2.4 Coordination Patterns for Agent based Systems

A coordination pattern is a special software pattern [2] and so has its roots in the field of building architecture [18]. It always expresses the relation between a certain context, problem and solution.

Agent based systems are implemented to solve a distributed problem within an agent group. The agents have to communicate and be coordinated to find a solution. Efficient and failure free coordination between a numbers of agents can be based on patterns. These patterns present a possible solution to a certain problem.

Before searching for a relevant pattern Deugo, Weiss and Kendall [19] suggest to define the global forces of the given coordination problem.

2.4.1 Classification

As there are different kinds of patterns for textiles there are also different types for software systems.

Architectural Patterns

Architectural patterns describe the basic structure of a software system. Therefore the system is divided into sub parts where each pattern describes the responsibilities, relationships and communication between the different system parts. Like mentioned before the description of the sub systems should not be detailed, it should just reflect an

overview of the different classes. The step deeper, into the classes, is done by design patterns.

Design Patterns
The class structure and the relationships between the different classes are defined in design patterns. The results are often expressed in class or sequence diagrams.

Idioms
Idioms are patterns based on the programming language. So they describe a more detailed way how the design patterns are realized with a certain programming language. Idioms are more an implementing view on the system.

Anti patterns
Anti patterns should document how and why a solution idea was not successful, to prevent doing the same mistakes again.

2.4.2 Content of a Pattern

Patterns follow a consistent structure format. A basic standard is not defined. Karl Eilebrecht and Gernot Starke [20] defined the following format for a pattern:

- **Scope**: First of all it is important to define what benefits will come with the usage of the pattern.
- **Scenario**: Describing a concrete example where the pattern can be used and which problem can be solved.
- **Problem/Context**: An overview of the structural and technical context of the problem and how it can be solved.
- **Solution**: The detailed description of the contextual solution including the basic conditions. To reflect the solution a diagram can help the visualization.
- **Advantages**: What advantages does the usage of the pattern implicate?
- **Disadvantages**: If basic conditions work against each other like speed and space, disadvantages can occur.
- **Usage**: Examples of scenarios where the pattern can be used. Design and Idioms patterns can also be visualized by using pseudo codes.
- **References**: References to related patterns.

2.4.3 Design

„Fundamental principles have to be kept for the design of object oriented systems as well as they build the basics for most patterns”[20].
Eilebrecht and Starke define the following design rules for patterns:

- **Simplicity**: The solution should always be as understandable as possible. Patterns with a complicated description will not find a lot of attention in future projects.
- **Principle of least astonishment**: means that astonishing solutions are mostly difficult to understand.

- **Replication**: Patterns should not include the same structures or logics twice.
- **Separation of Concerns**: Each class has fixed responsibilities. A class should not be overloaded with responsibilities.
- **Open Closed Principle**: Software components should be extendable but not modifiable.
- **Interface Segregation Principle**: means that clients should not be depended on any services. The interfaces belong to their clients and should not be a part of the class hierarchy.
- **Common Reuse Principle**: If one part of the pattern is reusable, all parts have to be.
- **Acyclic Dependency Principle**: Do not use circularly dependencies within the patterns. It just reduces the flexibility and the maintainability and you just can test the whole circle.
- **Stable Dependencies Principle**: Do not use dependencies between components that are modified very often.

2.4.4 Global Forces

Forces within patterns are factors which influence the design and the implementation. They are different criteria which get the highest priority by the engineers. They often work against each other like space and speed. There are several forces that influence the coordination between agents.

Mobility and Communication

Mobile agents which coordinate their activities with other mobile or static agents, resources and hosting environments have to communicate with the target object. Before starting the communication at least one of the partners has to know the location of the other agents. The sender can include his location in the message, so that the receiver can filter the response location from the message.

There are three possible ways for an agent to get the location data about its partners. One is that the agents know the location of their partners in advance. This would mean that the agents have to have more memory for saving the location data all the other agents. These extra data would need even more memory if the agent system is complex. Hence this solution is better for smaller agent networks.

The second way to know about the agents' location is a look up services where all agents are registered and the agents can search for their partners. The usage of such a service can result in synchronization problems if agents change their location very fast and often. Another problem is the consistency of the service because if it fails no agents are reachable. The last solution would be an agent broadcast which means that a message is sent to all units in a system. Besides the security problem by sending all messages around the system, the message overhead is a big problem. Furthermore not every agent need to receive every message and so messages can get lost or important ones need a long time to reach its destination.

Standardization
Like everywhere standardization is needed in the pattern language. Using different languages can easily cause misunderstandings. Especially the content of the messages has to be interpreted by all communication partners in the same and correct way.

Spatial and Temporal Coupling
Spatial coupling means that the agents share a common name space and can communicate with each other by naming the receiving agents explicitly. They can use a naming or locating service to get the agents identification. The communication is realised using a peer-to-peer manner with agreed protocols, locations and times. The connections between the agents have to be stable and reliable. This assumption is not always realized, e.g. mobile agents often change their location. In contrast static agents can benefit from spatial coupled models by interacting without having the name of their partners. In such cases the information inside the message is more important than the identity of the sender or receiver.
Using temporal coupled models means to synchronize the agents within a community. They have to agree on what to share, when to share it, and how to share it which results in an increase of the system complexity and computational requirements. “Moreover, the model requires that the agents share a common knowledge representation and are aware of schedules and positions for information exchange “[19].

Problem Partitioning
In MAS problems are divided into sub problems. These tasks are partitioned among the agents to get an increase in reliability, performance and accuracy. Therefore the agents have to coordinate with each other and the results have to be stacked together.

2.4.5 Blackboard Pattern

The Blackboard Pattern is one solution proposal to design a communication medium between the agents. It helps to coordinate and monitor the information and data flow in a community. Hayes-Roth’s BB1 system [21] is one solution where a blackboard pattern works as control component.

Scope
The benefit of using a blackboard pattern is a centralized messaging board where all agents can read/write information. All information and messages are stored in an object called blackboard. The message exchange can be managed by a supervisor.

Context
The system includes agents with different special abilities to solve a problem. The results of the activities done by the agents have to be monitored and managed.

Problem
Each agent works on its task for itself but is also dependent on the activities of others. Therefore the cohesion of the agents has to be realized.

Forces
The agents have to solve complex and distributed tasks. Therefore they have to collaborate, which means to create a common way of representing relationships. If the communication would be realized in a direct way a lot of traffic would be created. To prevent such an overflow a centralized cooperation mechanism would help to solve the bottleneck.

Solution
Each agent can add data to a blackboard. By subscribing to information of their interests other agents can get access to the blackboard. A Supervisor is monitoring the blackboard and has the responsibility to decide which agent may modify information inside the board and when a solution to a given problem is found. The Supervisor acts just as a scheduler and does not interact with the agents.

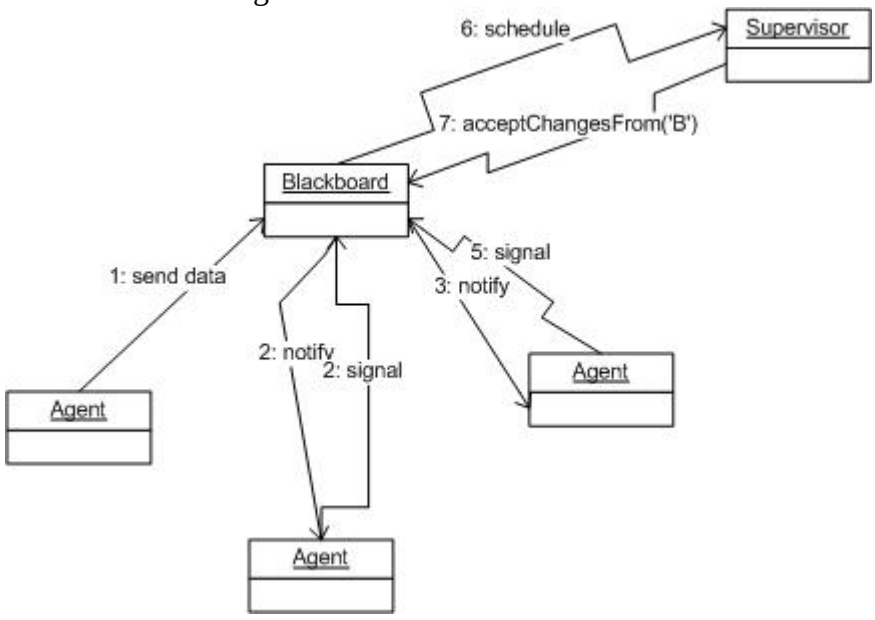


figure 2: Interaction in Blackboard Pattern (c.p. [19])

Rational
The Blackboard Pattern decouples interaction between the agents because, instead of communicating directly with each other, they interact through the blackboard. This results in a transparency of time and location. The receiver can pick up its message any time and the sender/receiver does not know the location of the other agents. The agents have to subscribe to get certain messages. Another advantage of the Blackboard Pattern is that it guarantees mobility because the agents can change their location and access their message from any location.
The blackboard is a passive medium because agents do not get an order how to react on the message they receive.

2.4.6 Meeting Pattern

The Meeting Pattern simulates a virtual environment where individual can meet to discuss certain problems. The IBM's Aglet's framework [22] and Concordia [23] project include such a Meeting Pattern solution.

Scope
The Meeting Patterns provide a central meeting point where agents can exchange their data and information.

Context
Agents need to coordinate their activities but they do not have to know the explicit name of their partners. The agents can be static or mobile. Direct messaging is possible.

Problem
How can the agents manage and coordinate their activities?

Forces
To get a secure, fast and simple messaging mechanism all the agents should be in the same environment. It is usual that the agents are not located on the same system; hence a remote environment has to provide the information. The agents can interact directly if they have a connection between agents' environments.
Messaging services provide a naming service that can be contacted to get the name and location of the communication partner. This increases the mobility of the system.

Solution
A solution is the creation of named meeting places, where the agents can send messages to a statically located and named agent, which manages the meeting place. This manager is called Meeting Manager. It is responsible for the notification of other agents and accepts messages from remote or local agents. The agents which want to be part of a meeting place have to specify their identity and their location. The controlling of the meeting is the task of the Meeting Manager which also informs agents which are interested as soon as it starts.

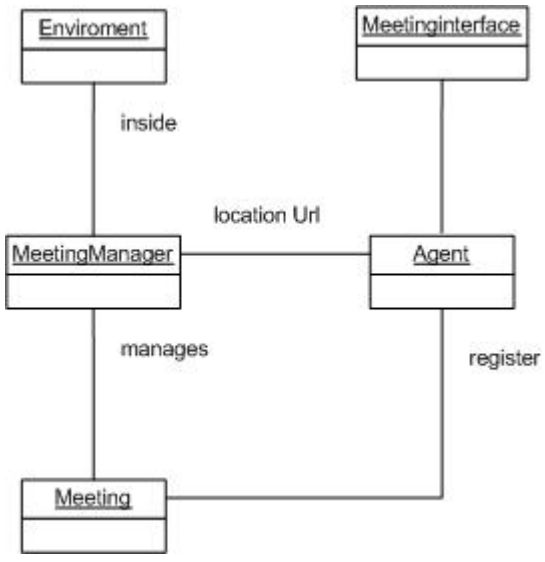


figure 3: Meeting Pattern (c.p. [19])

Rational
The Meeting Pattern reduces the names of the agents which have to be known by others. It does not completely remove knowledge about other agents because everyone in the meeting has to know the Meeting Managers name and location. The Meeting Manager has to know the names of the members. So it is a loosely coupled environment.
The overhead of messaging is reduced because the network is not involved and the security is "localized to the responsibility of the environment the meeting is held at" [19]. Additionally the coordination is not influenced by network failures or delays because all agents are in one meeting place. If the environment of the meeting place crashes, the previous state can be rebuilt.
The meeting pattern does not include a defined mechanism of how agents collaborate. As soon as the meeting starts the agents are responsible for the communication process.

2.4.7 Market Maker Pattern

The Market Maker Pattern simulates a virtual market where sellers can offer their goods and services to buyers. The Contract Net protocol by Smith [24] and the Magnet Market Infrastructure by Steinmetz [25] deal with the Market Maker Pattern.

Scope
The Market Maker pattern provides a virtual market where a seller can offer its goods to buyers.

Context
The relationships between the agents can be substituted by an on-the-fly locating of transaction partners.

Problem
The question is how an environment like on a "real" market with buyers and sellers can be created?

Forces
The agents have to collaborate to solve complex problems and have to be designed independently. The embedment of coordination logic into the agents results in an increase of the complexity and affects the global application design. "Although a logical separation between algorithmic and coordination issues increase the cost of the coordination mechanism, one needs the flexibility to implement and modify coordination protocols, while keeping any changes hidden from the collaborating agents" [19].

Solution
The virtual market is managed by a Broker or Market Maker which receives bids from Buyers and forwards them to the Seller. The Broker handles the coordination, collects the bids and introduces the selected Seller to the Buyer. It acts on the behalf of the Buyer. The Sellers decide if they want to provide the requested goods and submit bids. They can add conditions, for example a minimum price to their goods.

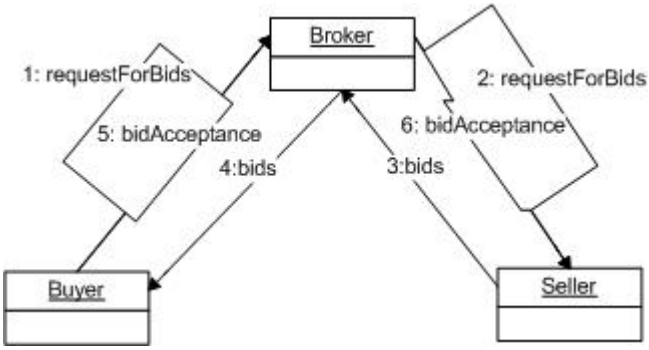


figure 4: Interaction in Market Maker Pattern (c.p. [19])

Rationale
The Broker acts as coordination medium with an active role. It controls the interaction between the agents. The other agents delegate their coordination duties to the Broker. Designers do not have to implement coordination protocols they just need to define the application logic. Buyers sometimes have to negotiate with the Sellers about the terms of the transaction this would lead to a negation pattern.

2.4.8 Master-Slave Pattern

The Master-Slave Pattern is a common solution for dividing a certain problem into different tasks and delegating them to problem solvers. The Consult project by Carriero [26] uses such a pattern solution.

Scope
The Master-Slave Pattern divides a problem into sub problems and distributes them to different agents. The Master can monitor the activities of its slave.

Context
The problem to be solved is divided into sub problems to increase reliability and performance.

Problem
Divide a complex problem into a certain number of subtasks. These tasks have to be delegated to other agents. To improve the efficiency and prevent failures an agent has to monitor the execution of the tasks.

Forces
An agent can divide its problem into sub tasks and forward them to other agents while it can continue with another work parallel. This solution increases the overall computation effort so simple problems do not have to be partitioned into sub tasks.
The partition should be transparent and the agents return their results to the master/client. At the end the results are stacked together to have one solution.

Solution

The partition of a problem into sub tasks is done by a Master which delegates the sub problems to Slaves and is responsible to collect all results for the final solution. The Master can work parallel to the Slaves. The Slaves can work in a remote location and can be assigned to new tasks when they finished their current problem.

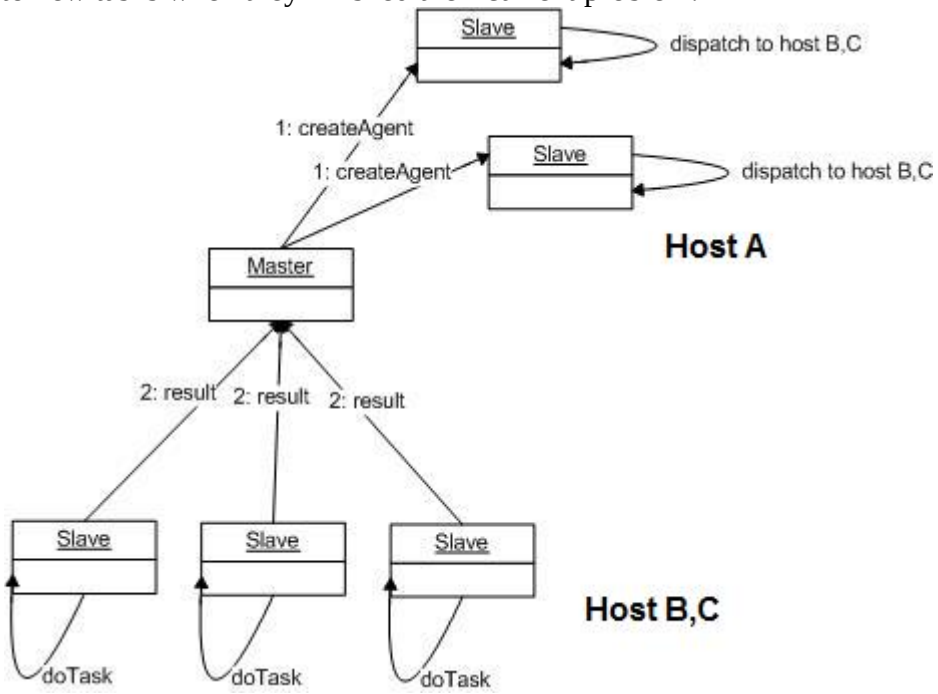


figure 5: Interaction in Master-Slave Pattern (c.p. [19])

An example for a Master-Slave Pattern is parallel computation where a number of Slaves are assigned to a work pool controlled by the Master. Each Slave offers the Master its service. A client sends a request to the Master and it divides the problem into sub tasks and forwards them to the Slaves.

Rationale

The Master-Slave Patter provides a vertical coordination for the activities of two or more agents. Vertical coordination means that an agent fulfils the sub task of another but the problem is still a logical part of the former agent's task.

2.4.9 Negotiating Agents Pattern

The Negotiating Agents Pattern is an interacting framework for agents. The agents act as peers to each other and align their actions.

Context

Agents interact as peers. During the problem solving process conflicts can lead to inconsistence.

Problem

In an agent-based system a huge number of components and units have to work together. During the problem solving process conflicts can occur. The Pattern tries to prevent such misunderstanding and problems.

Forces

Agents synchronize their actions which "can be useful, desirable, or event essential to the achievement of their individual goals" [19]. One solution for this theory is the "ostrich's algorithm", which means to ignore the possible conflicts and handle them after they have occurred. The problem of the appointment is that sometimes a "rollback" to the previous state is not possible anymore.

Solution

The agents tell their partners about their working plans. According to these plans the agents can re-plan their own actions which can also include alternative plans. The negotiation process starts with a declaration of an Initiator about its intentions to its peers. The peers are the other agents, who will act as Critics and test if there will be a conflict with their own activities. If there is no conflict the Critics accept the further working steps, otherwise they make a counter-proposal or reject the action. The counter-proposal would be an alternative activity and a rejection is solved outside of the negotiation framework. Agents can act as Critic or as Initiator at the same time. To handle the different roles, there has to be someone who manages the negotiation which is done by a Participant. To understand the negotiation between the agents, the actions and alternative activities can be represented as decision trees. So if an action of the Initiator conflicts with an activity of a Critic they have to find an alternative way to solve the problem.

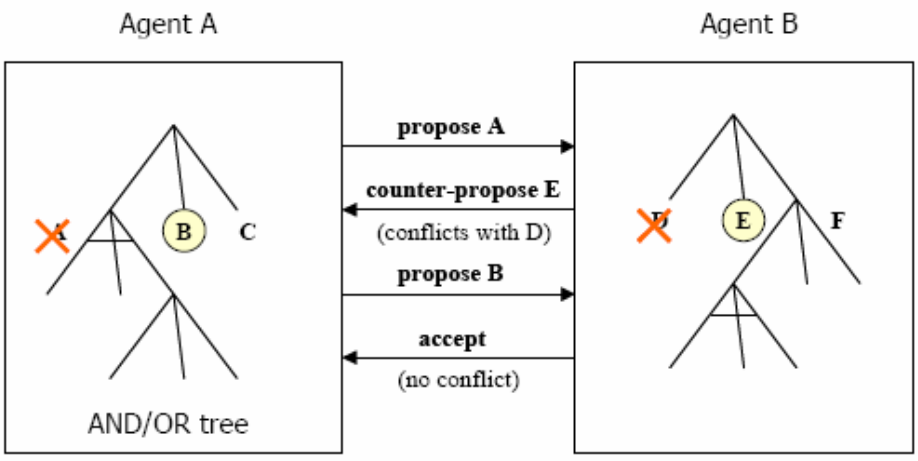


figure 6: Illustration of using the Negotiating Agents pattern [19]

Rationale

The Negotiating Agents pattern assumes that the agents act as peer and their actions depend on each other. The agent can exchange their intentions through multitude protocols. The Telecommunication [27] and MAGNET [25] project use the Negotiating Pattern for the agent coordination.

3. Research issues and research method

This thesis gives an overview and defines guidelines for designers of Multi Agent Systems (MAS) for automated production planning system based on a simulator. Production automation is a manufacturing process that coordinates and controls the various entities needed to achieve predictable system behaviour. The various automated entities are represented as simulator where business orders, like assembling products, can be simulated. Multi Agent Systems can be used to develop such a simulator and to represent the hardware entities, their behaviour, and communication in a simulation environment, which facilitates efficient evaluation of design choices in order to optimize system performance. For the simulation of various manufacturing processes the used agents have to be coordinated. Therefore a separated coordination component, which represents the interface between the business layer and the workshop layer (simulator), has to be developed. This component is described in a coordination pattern which should be reusable for manufacturing systems similar to the one used in this thesis.

The project is based on a simulator kit developed by Rockwell Automation. The agent system already includes a certain number of simulation agents which have to be modified and extended. The main aim is to implement a coordination component which has a global overview of the system.

3.1 Problem statement

3.1.1 Multi Agent System for production planning

Modern production planning systems usually consists of a huge number of entities, like robots or transport system, and need a lot of time, space and resources to be tested. Multi-agent systems (MAS) offer a convenient and cheap way of modelling processes and simulate such assembly systems. MAS possess the ability of being distributed over space and time, an easy reconfiguration of certain states, increasing flexibility, reducing complexity and enhancing fault tolerance. Depending on the functionality of the production system facility, the agents have different behaviours and abilities. With this artificial intelligence the agents have the possibility to interact with each other, which results in self-coordinated actions within an agent community. This ability is one of the big advantages of using MAS for complex and distributed systems.
The simulator is based on the Multi Agent System Tool (MAST) developed by Pavel Vrba, Rockwell Automation, Prague [28], and the layout used for testing represents the existing pallet transfer system[3] at the Institute for Automation and Control, Vienna University of Technology.
The basic idea of the thesis is the development of a coordination component for the agent-based production automation system which represents the interface between the business layer and the workshop layer (MAS simulator). The design process of such a coordination

[3] http://www.acin.tuwien.ac.at/en/forschung/Projekte/255/

component defines guidelines for deriving a concrete design and implementation from a set of coordination patterns, simulator technology, and production automation context.

3.1.2 Coordination of agents

Coordination describes the behaviour of a certain number of entities and the optimization of a known function while observing a set of constraints. In our project the entities are represented by the various agents. The complexity of such agent systems required a well organized coordination object which is realized by a separated coordination component. The development of such a component results in an interface between the business layer and the workshop layer. The business layer is responsible for incoming tasks and has to provide a global overview of them. The management of these tasks is done by a dispatcher which translates customer orders into more detailed work orders. For the fulfilment of these work orders the simulation agents have to be coordinated. This coordination is the main research field of this thesis. The goal is to have a coordination component that controls the interaction between the simulation agents and the dispatcher. Therefore coordination agents have to be implemented. These agents represent the interface between a dispatcher and the simulator. The dispatcher is responsible to administer orders (lists of products) and forwards them to the simulation. The coordination component is completely separated from the simulator and is easily extendable. During the implementation process new functions, like new workshop scheduling strategies, are added.
The coordination agents are based on JADE[4] and communicate with the simulation agents over the Agent Communication Language (ACL).

3.1.3 Coordination Pattern

This thesis introduces coordination patterns that allow describing the coordination of agents on several levels in the context of a multi-agent simulation of a production automation workshop. The coordination of such an agent system is usually very complex because of distribution and autonomy of the individual software agents. Thus the coordination is described by patterns. Patterns are prototypes for a software solution with a given problem and context. During this thesis a coordination pattern is developed and implemented. It gives an architectural overview and guidelines for designers of multi-agent systems for tasks that are similar to production planning and automation. The pattern is the interface between the dispatchers, which act in the business layer, and the simulation agents, which represent the workshop layer. Therefore dispatchers transform customer orders into more detailed work orders, which describe the machine functions in the workshop to carry out in order to fulfill a customer order, and prioritize the work orders in order to optimize the overall system performance. The dispatchers are implemented as coordination agents and are based on JADE.

3.1.4 Self coordination

A well defined organization of the agents can increase the efficiency of agents under certain conditions. By changing these conditions the performance of the whole agent

[4] jade.tilab.com

community or of an individual agent may increase or reduce. For example the position of the agent within the simulation would influence the efficiency of the agents working together. Another example is the communication between the agents. A predefined communication scenario is often not possible because the agents react during the run-time on certain conditions or states. That is why agents can decide with the help of their behaviours what they have to do next. The routing function within the simulation is a good example for that situation. If a conveyor belt has a failure the agents can change the route of the pallets within the simulation. This is achieved by sending messages to the other agents. This artificial intelligence of the agents helps them to organize themselves. The existing simulator already includes such a coordination component.

3.1.5 Workshop scheduling strategies

The optimization of the different production strategies that can be used during the manufacturing processes is another part of the research for this thesis. The optimization is done by comparing the various workshop scheduling strategies during a set of test runs. These test runs differ in the definition of the input parameters. The parameters include a certain workload package (orders) and a production strategy. In order to decide which strategy is the most efficient, a certain number of simulation data have to be logged, saved and analysed. The data analysis is supported by mathematical algorithms. In the first analyses the number of finished products and the machine utilization play the most important role. The different simulation data have to be reflected in ratio values to compare which workshop scheduling strategy is most efficient.
After these first - rather simple - analyses, more complex ones can be done. Therefore the logging component needs to be extendable without much effort.

3.1.6 Proof of concept

The coordination concepts and the implemented workshop scheduling strategies are tested by a developed test management system. This proof of concept will show on the one hand if the developed pattern is able to coordinate a complex and distributed agent-based system. On the other hand the effectiveness of the various production strategies under certain conditions can be analysed and valuated.

The developed agent framework solution for the coordination component is compared with existing implementation. Therefore a literature study will give an overview of the differences between our solution and other agent frameworks.

3.2 Research methods

The research work is done within a project group, which goal is to develop an automated production process based on a simulator. The research method of the thesis is mostly the implementation of such a system. The work is based on an existing simulator. This simulator is implemented in Java and uses the JADE (Java Agent Development Framework) as agent technique. Thus the extensions are also developed in Java.

The ideas, methods and strategies of the manufacturing industry are tried to be mapped to the application. The information is achieved by performing different literature researches and discussed within the project group. The information is tried to be converted into a format that they can be implemented within the project. Therefore different roles within the group are defined. After the implementation of each component the results are analysed within the research group and improvements or extensions are discussed. This new information is similarly tried to be implemented and discussed. This incremental way of the development is used to reach the defined goals.

3.3 Research Questions

This section gives an overview of the research items and questions of this thesis.

- ***Is a MAS based simulator a possible solution for simulating production planning systems?***
  Production planning is a manufacturing process that coordinates a set of hardware entities (such as machine and transport systems) to achieve predictable system behaviour. Such systems are very complex and need a lot of resources. MAS can be used to represent these hardware entities, their behaviour, and communication in a simulation environment in a relatively cheap way.

- ***How can simulation agents (manufacturing entities) be coordinated to simulate various business processes for production automation more effectively and efficiently?***
  The main part of this thesis is to implement a coordination component for an agent-based simulator. The usage of such a component guarantees an efficient and reliable way of monitoring a MAS in the production planning domain.

- ***How can coordination pattern be used to describe the negotiation between the agents?***
  The coordination of MAS is usually very complex and distributed. Therefore coordination patterns are used to describe the negotiation and communication between the agents.
  The developed pattern for the coordination component has to be extensible, thus it is important to analyse which coordination patterns can be used to extend the existing application to improve the system. Therefore the developed pattern has to be extendable and provides interfaces for further implementations.

- ***Is the developed coordination pattern for the agent framework comparable to existing solutions?***
  As there are existing solutions for similar problems, a comparison of the developed agent framework with existing solutions is done. Therefore a literature study of existing agent frameworks for production planning systems helps to compare the developed solution with others.

- ***Is the developed coordination framework for the agent-based simulator reliable and useful?***
  A proof of concept shows if the developed coordination pattern successfully can be used to simulate manufacturing processes and where improvements regarding to the used workshop scheduling strategies can be done.

3.4 Goals and Research Contributions

The following section describes the basic ideas of the thesis and the goals that should be achieved.

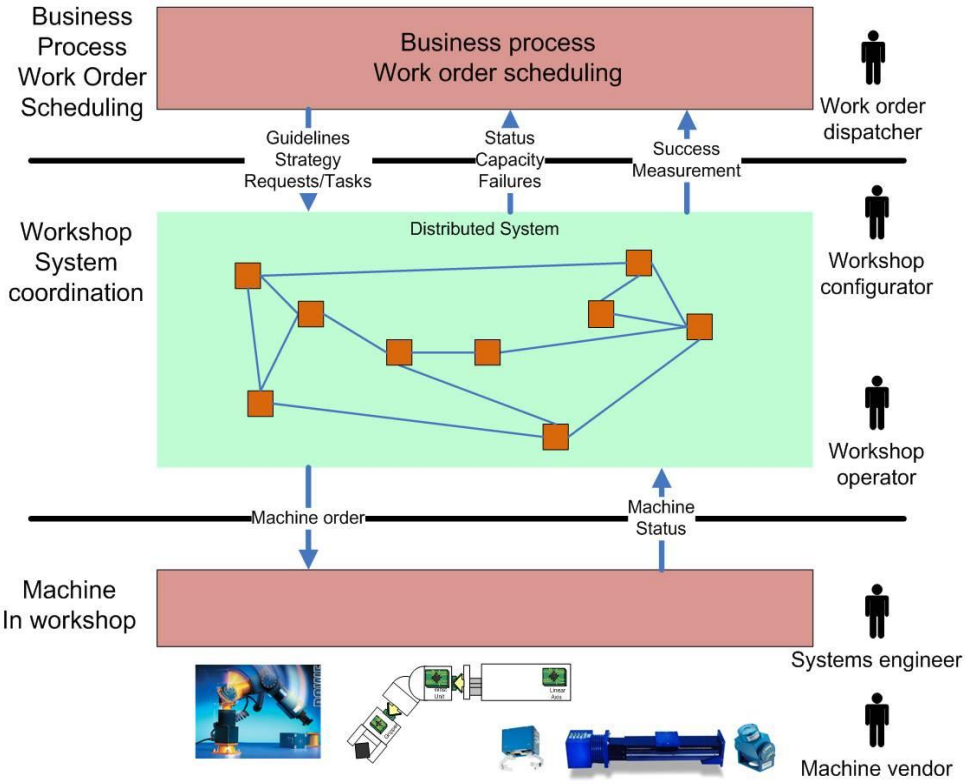


figure 7: Production Automation System Layer:

Figure 7 shows the layer model of the production automation system used in the project. The work order dispatcher in the first layer is responsible for the business processes. The business processes represents guidelines for the layer among, the workshop layer. The workshop layer is symbolized by the developed simulator. The simulator has to fulfil the various tasks getting from the business layer and has to report about the different status, capacities, failures and the measurements. The simulation results are forwarded to the third layer which is represented by the “real” hardware machines.
The contribution of this thesis lies in the interface between the business layer and the workshop layer. This connection is realized by a coordination component which controls and manipulates the agents within the simulation to fulfil the orders getting from the business layer.

The research work helps to get a first overview of the implementation of MAS for manufacturing plants. As such manufacturing systems are usually very complex and need a

lot of resources an agent-based system can be used to simulate different production scenarios in a relatively cheap way. This first implementation draft includes a simulator based on MAS, coordination components, a test management and measurement components. They help to test different workshop scheduling strategies and measure their impact on the production flow, outcome and machine utilization. The results can be mapped to miniature hardware simulation test bed at the ACIN lab[5] at the TU Vienna.
One main goal of this thesis is to implement a running and representative simulator with coordination and measurement components which was presented to "end-users" at the CEBIT 2008. Therefore simple scenarios were defined and developed. Important for that part of the thesis was the fact that the user will get an idea about the different affects of scheduling strategies on the production. At the CEBIT the user could act as dispatcher deciding under which conditions he/she wants to produce.

Another goal is to implement a test management system that can be used for data analyses. This helps to get more knowledge about the different efficiencies of the various workshop scheduling strategies. Therefore a proof of concept with various input parameters is done.

[5] http://www.acin.tuwien.ac.at

4. Practical Application

This chapter deals with the practical part of this thesis. It describes the implementation of the various techniques used to develop the simulator. The first subchapter gives an overview of the different agents used in the simulator. The following section takes a closer look on the implementation of coordination patterns for agent-based systems before the data model used for the implementation is described. The connection between production planning and modelling and MAS is the issue of the next part. The last chapter deals with the developed prototype and its scenarios.

4.1 Implementation of MAS

The application of the research project is a software simulator for a production system, to be more precise for an assembly workshop. It can also be used to analyse the effectiveness of dynamic workflow scheduling strategies. Such manufacturing systems are very complex and distributed. Multi Agent System (MAS) are one possible solution to simulate such assembly mechanisms. The simulator is based on a decentralized control architecture and has centralized control and scheduling functionality.
The simulation results can be mapped to a “real world” assembly workshop. The usage of a digital simulator instead of a physical has a lot of advantages like, low operating costs, the easy reconfiguration or parallel testing.
A miniature hardware model of the assembly workshop is situated at ACIN/TUW; the software simulator has been implemented based on a production system simulation kit from Rockwell Automation International, Prague.

An already existing simulation framework from Rockwell has been modified and extended by new agents and components. To monitor and measure the assembly processes a coordination part has been implemented. Therefore coordination agents have been added to the simulator. These agents have to solve assembly problems with the help of the simulation agents. Thus they divide their problem in subtasks and delegate them to the simulation agents which have to fulfil them. During the problem solving process the agents communicate with each other in order to report about their states and actions. The coordination between the agents is realized using a messaging system, based on the agent communication language (ACL), which has already been implemented by Rockwell. The demonstrator was presented to an “end user” audience (at CEBIT 2008) and is used by researchers for testing and measurements of different production planning experiments. Therefore beside the agents, a test management has been added to the original simulator, including job scheduling, visualization components and data analysis algorithms.

This chapter will take a closer look at the agents used during a simulation run. They can be clustered into two types: Simulation agents and Coordination agents.

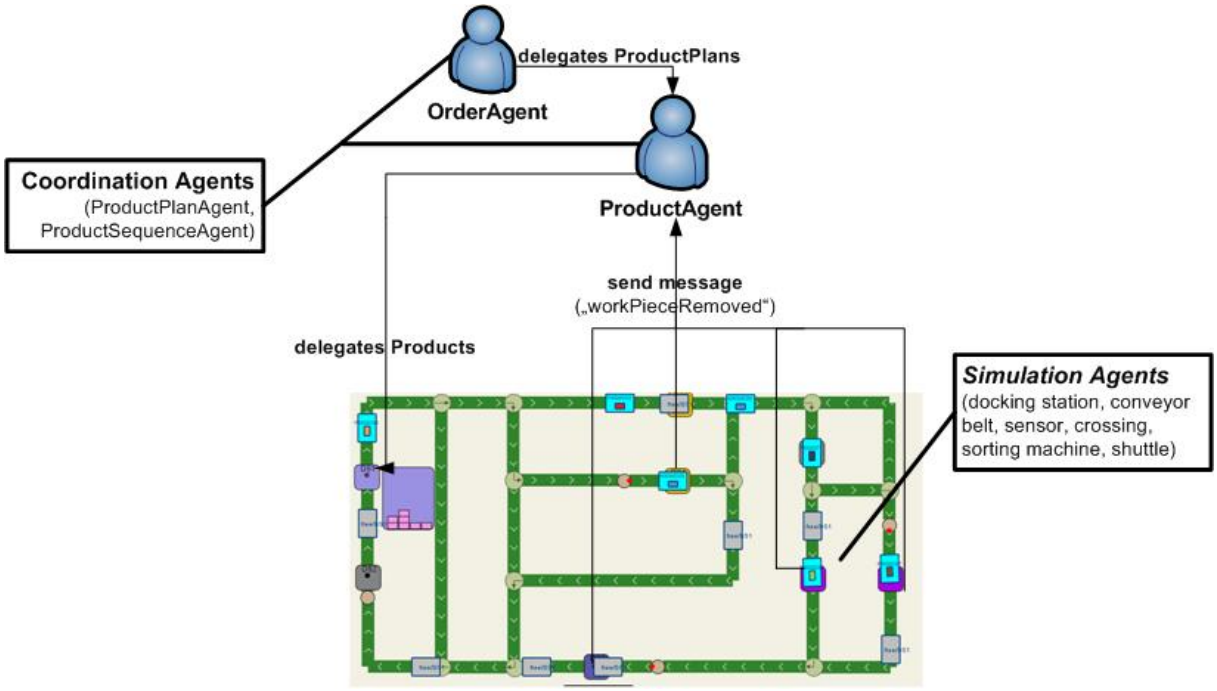


figure 8: Multi Agent System overview (CEBIT Project)

4.1.1 Simulation Agents

The simulation agents represent the components of the simulator. The simulator is implemented as a closed queuing transfer network with redundant paths, where each machine station (docking station) is connected to at least one transfer path (conveyor). For the configuration of a simulator the users can choose between six different components:

- Docking Station (machine)
- Conveyor Belt
- Sensor
- Crossing
- Sorting Machine
- Shuttle

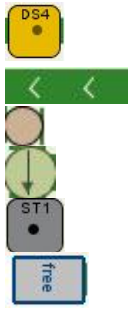


figure 9: Simulation components

DockingStation (machine)

The Docking Station represents a machine where shuttle can stop and unload there goods. It is more a kind of black box where different functions (for example, assembly or inventory machine) can be simulated. Every machine consists of a component class, that represents the component in the simulator and an agent class that is the agent representation of the particular component. The component class implemented by Rockwell, was extended by functions to fulfil the needed requirements.
The following method were modified or added:

- InSensor()

- removeWorkPiece()
- addProducttoShuttle()

InSensor()

The *InSensor()* method is invoked when a shuttle (workPiece) arrives at the machine. The given method, implemented by Rockwell, was extended. The extensions are responsible for production monitoring. Therefore an IF-statement checks if the incoming shuttle's destination is the one it enters. If not the shuttle is just send through, otherwise the extended methods will be invoked as described in the pseudo code (Appendix 8.1).

removeWorkPiece()

The *removeWorkPiece()* method is called by the *inSensor()* method as soon as a product is removed from the arriving shuttle. It was implemented during the thesis and added to the component class. It sends a "removeWorkPiece" message to the other agents in the system to inform that a product arrived and has been removed. Therefore the message is initialized as an ACL message and sent over the inter-agent communication channel. This message is received by the Product Agents which are responsibly for the production of a product plan (read more about the Product Agent in 4.1.2).

```
String content = "<removeWorkPiece>\n" + currentWorkPiece.toXML() +
                                                    "\n</removeWorkPiece>";
gui.simulation.sendInform(this.getName(), content);
DockingStationAgent agent = (DockingStationAgent) this.getAgent();
agent.informSubscribedAgents("<removeWorkPiece/>", content);
```

listing 1: creation and sending a "removeWorkPiece" message

addProducttoShuttle()

The *addProducttoShuttle()* method is invoked by the Product Agents in order to add a product to the current shuttle in the machine. Therefore the method has a ProductPlan as input parameter which represents the product that should be transported by the shuttle. Before sending the shuttle to its destination the method checks if load balancing is needed. This analysis is done if the product can be manufactured on redundant machines. For more information about load balancing take a closer look at chapter 4.4.8.

Conveyor Belt

Conveyor belts are responsible for the transportation of a shuttle from one point of the simulation to another. They are always connected either to a docking station, sensor, crossing or sorting machine. The classes are based on the Rockwell implementation and were not modified.

Sensor

Sensors are needed to prevent traffic jams at docking stations (machines). They have to be placed in front of a machine. After a shuttle passes the sensor and enters the docking station, the sensor detains following ones to pass, as long as the original shuttle has not left

the docking station. The agent and component classes are based on the Rockwell implementation and were not modified.

Crossing

The crossings can act as junction with three different constructions: bottleneck, bifurcation and crossing.

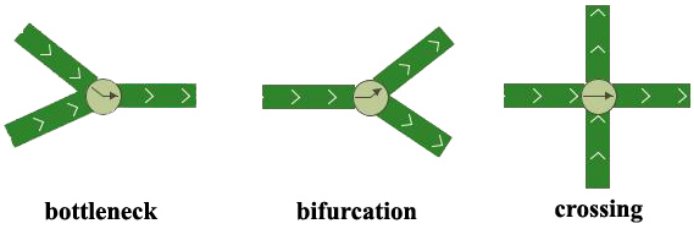


figure 10: three types of crossings

In our system, crossings are limited to one or two conveyor belts as in-nodes and one or two as out-nodes. Since the agent and component class have not been modified.

Sorting Machine

Based on a docking station the sorting machine agent was added to the simulation. It is responsible for controlling the shuttle transfer to one or more machines (docking stations) in a predefined order. Therefore it is located in front of a certain number of assembly machines. To guarantee the right order of arriving products at their assigned destinations, the Sorting Machine Agent has to check whether a product depends on another or a sibling product already passed and has been sent to a machine. If both conditions do not occur, the shuttle with the product is sent into a waiting loop. A waiting loop is a closed cycle of conveyor belts. The sorting machine has to be located in such a cycle and checks for each passing good its dependence to other products.

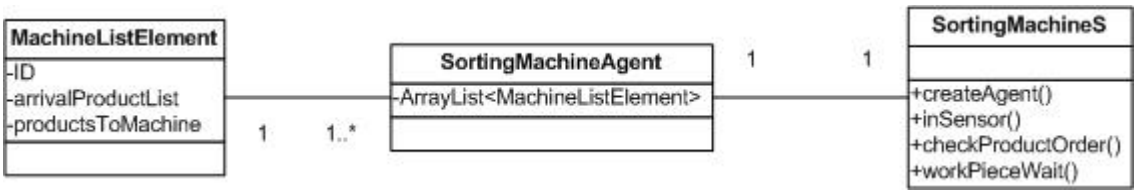


figure 11: class diagram of sorting machine agent

Every component registers an agent class (1 to 1 relationship). For the assignment of a sorting machine to a set of assembly machine(s) the MachineListElement class is important. Each sorting agent class has an ArrayList of MachineListElement elements. Each entry on the list represents a relationship to a machine. Each MachineListElement has an ID attribute which is the ID of the assembly machine it represents and two ArrayLists. The arrivalProductList represents the list of products which should arrive at the assembly machine. This is important in order to find the sibling product of the arriving product. The productsToMachine list represents the list of products that already arrived at the machine. If the arriving product does not have an sibling, it will be removed from the list as soon as

it arrives at the machine. If the product has one or more siblings, it will be removed from the list as soon as all siblings has been arrive.

The agent class is similar to the one of the docking station (see DockingStation). The differences are in the component class which includes extended methods:

- inSensor()
- workPieceWait()
- checkProductOrder()

inSensor()

Like in the docking station (machine) class the *inSensor()* method is invoked when a shuttle arrives at the machine. For the sorting mechanism two conditions have to be analysed before the decision is made.

1. Check if the arriving product depends on another product and if the sibling product already has passed the sorting machine, therefore the *workPieceWait()* method is invoked.
2. If (1) is not achieved, check if a sibling product is already waiting in an assembly machine or a machine is free, therefore the *checkProductOrder()* method is invoked.

workPieceWait()

The method is responsible for checking if the sibling product of the arriving one already has been sent to an assembly machine. For more information read Appendix 8.2.

checkProductOrder()

The *checkProductOrder()* method has two important parts. The first one checks if the sibling product of the incoming product already has passed the sorting machine and waits in a machine or is on the way to it. If this is not the case the method searches for a free machine. If all machines are occupied the shuttle with the product is sent into a waiting round (waiting loop). For further information read Appendix 8.3.

4.1.2 Coordination Agents

The coordination agents are responsible for fulfilling various orders and can be classified into two different groups. The following agents are needed to simulate a test case: OrderAgents and Product Agents.

OrderAgent

The *Order Agent* (OA) is responsible for incoming orders and acts in the business layer. It is represented by the work order dispatcher. The work order dispatcher manages the incoming business orders. A business order consists of a list of certain set of products which have to be manufactured. Before starting with the production processes the OA sorts the product list depending on the used workflow scheduling strategies defined in the input

parameters. For each product to produce the OA registers a *ProductAgent* (PA) and delegates the product plan, which represents the product to be produced, to the PA.

Product Agent

The *Product Agent* (PA) is the main coordination component developed in this thesis. It represents the interface between the business layer (*OrderAgent*) and the simulator. The PA gets work order in form of so called products plan. This plan is analysed and divided into more detail working steps, like assembling or transporting. After this analysis the working steps are delegated to the simulation agents.

During the simulation run the PA is responsible for monitoring the production processes of one product and sends a message as soon as all processes needed for producing the particular product have been finished. After each status change or event which influence the PA, it analysis the product plan and decides which step has to be done next. Therefore the following methods have been implemented:

- **createProduct()**
  The method starts the production process of the product, the PA is responsible for. The PA analyses the product plan and searches for products which do not have any child (sub-) products. These products are added to the inventory list. The inventory sequentially takes products from the list and puts them on the next (free) shuttle arriving at the inventory of the basic material (goods) and sends it to its destination.

- **handleInform()**
  The *handleInform()* method receives *sendInform*- messages from agents within the simulation. To receive such messages the agent has to subscribe to the other agents (see chapter 4.4.6). The PA is just interested in messages from machine agents (docking station) which inform that a product has been removed („removeWorkPiece“) from the simulation. Therefore the PA filters the received messages. The filtering process helps the PA to know which machine sends the messages. Within the message the sequence and order ID of the arriving product is included. Based on this information, the PA knows whether it is responsible for the product.
  Within the product plan the arriving product is marked as “finished” and the PA invokes its *checkHierarchy()* method in order to decide which steps have to be done next.

- **checkHierarchy()**
  This method analyses the product plan and decides which task has to be done next. If all tasks have been finished and the end product has arrived at its destination, the OA is informed about the successful production of a product.
  For more information about this method take a closer look to Appendix 2.4.

4.1.3 Java Agent Development Framework (JADE)

One possible way of implementing multi agent system is the Java-based JADE Framework [29]. Jade is specified by the Foundation for intelligent Physical Agents (FIFA) [30]. It is a middleware which includes an own runtime environment, where agents can be executed, a library of agent classes and graphical tools to administrate and monitor the agents activities. The runtime environment is also called Container and includes a certain number of agents. It is possible to register different agents in various containers, such a set of containers is called platform. The first container which is active in such a platform is the Main Container; all others containers have to be registered to it.

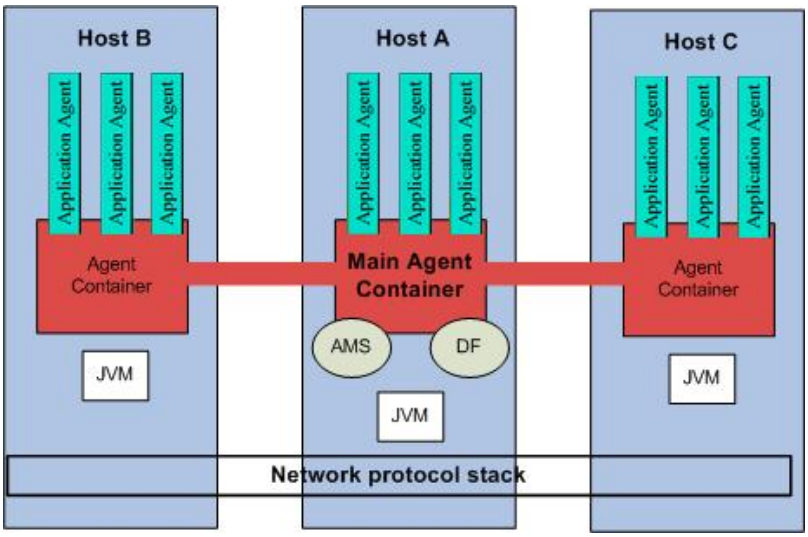


figure 12: distributed architecture of JADE [29]

Agents within a container are running in a multi threaded environment and have a unique identifier. The Main Container provides two different services to look up for partners. One is the Agent Management System (AMS) which includes a naming service and represents the authority in the platform. The other service is the Directory Facilitator (DF) which provides a Yellow Pages service where agents can find communication partners. The communication is done by sending/receiving messages over the Agent Communication Channel (ACC). These messages are Java objects based on the Agent Communication Language (ACL). If the agents are in the same container, the messages are sent as event object. Communication beyond the containers border is based on Java RMI (Remote Message Invocation). If the agents are existing on different platforms, the ACL message is converted to a String-object and is sent by Internet Inter- ORB Protocol (IIOP) or HTTP.

The agent library includes the agent classes. Each agent has to instantiate a class and is implemented as a thread following the FIPA lifecycle for agents. It is initialized by the *setup()* method. The unique name is given by the agent-identifier (AID) class which can be invoked by *getAID()* method. Besides the name the AID also includes a number of addresses which represent the platform the agent is registered to.

```
<nickname>@<platform-name> - sortingmachine1@P1
```

listing 2: agent identifier in JADE

Software Agents are used to solve different tasks. This ability is defined in the behaviour class. The *addBehaviour()* method adds a defined behaviour to a certain agent. There are several predefined behaviours like sending/receiving messages which can be extended or modified.

JADE also provides basic security mechanisms. The access to a platform can be limited by an authorization function which is based on the Java Authentication and Authorization Service API (JAAS). The rules are defined using a policy model. The integrity of the messages and the sender is proved by signatures. The confidentiality of the messages, which means that trespassing is being avoided, is guaranteed by encryption. JADE also includes agent authentication based on JAAS.

4.2 Implementation of Patterns

Patterns are prototypes for software solutions, describing the most important parts of a system in an abstract way. They are based on past experiences and reflect a successfully solved problem.
Multi Agent Systems are used to solve complex and distributed problems. They often contain a lot of different components/agents which form a working community to solve a given problem. The usage of patterns can help finding possible solutions for the agent system's requirements. This can either be in the construction of the system or in the coordination. Patterns help to save time and costs.
As already described there are some famous patterns for the coordination in agent-based systems. In this project two kinds of patterns are identified which are described in the previous chapter: Master/Slave Pattern and the Blackboard pattern.

4.2.1 Extended Coordination Pattern

The developed pattern for the coordination component is based on the Master/Slave pattern. It consists of three layers and represents the interface between the business layer which is located in the first layer and the work shop layer located in the third layer.

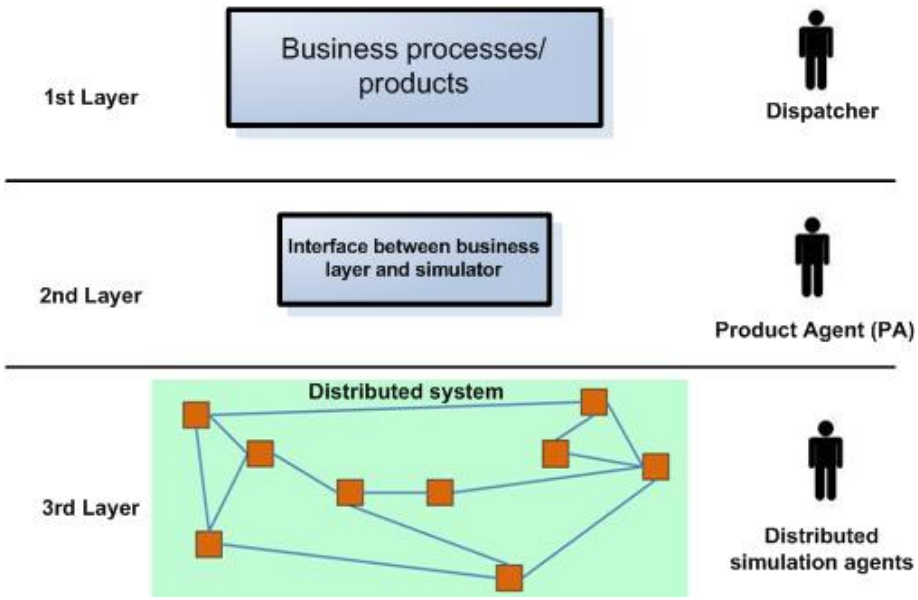


figure 13: Coordination Pattern for SAW project

The *Order Agents* (OA) of the first layer act as Master and are responsible for incoming work orders. The work order consists of one or more products. The products sequence within an order is depended on the work shop scheduling strategy. After this sorting function the OAs register for each product a *Product Agent* (PA) and hand over the product as a so called product plan.
The *Product Agent* (PA) of the second layer act as Slave for the OA as well as acts as Master for the simulation agents. It is responsible to convert the product plan into more detailed work orders and forward them to the simulation agents. Furthermore the PA is responsible to monitor the simulation/production processes of the simulator influencing its product.
The simulation agents within the third layer work as Slaves of the PA and have to fulfil and report about the tasks delegated by the PA of the second layer.

4.2.2 Master/Slave

The Master/Slave pattern is a common solution for the coordination of different individuals. As soon as there is a component that can divide its problems into sub tasks and is able to delegate them to other partners, the principal of the pattern is fulfilled. The solution increases the reliability, performance and accuracy in the system. Besides parallel working is possible because during the Slaves fulfil the tasks of the Master, the Master can work on other problems as well.
In this project the production processes are monitored by a so called *Product Agent* (PA), which gets the product plan of one product. It represents the Master who communicates with the simulation agents. The Master/Slave pattern defines that a given problem is divided into sub tasks. These tasks are distributed to the Slaves. The Slaves in the project are the simulation agents, like shuttles, machines or belts. The problem to be solved is the product plan which has to be fulfilled.
The Slaves can also act as Masters. They may have Slaves which fulfil their tasks. In the project there are four layers of Master/Slaves relationships as seen in figure 14:

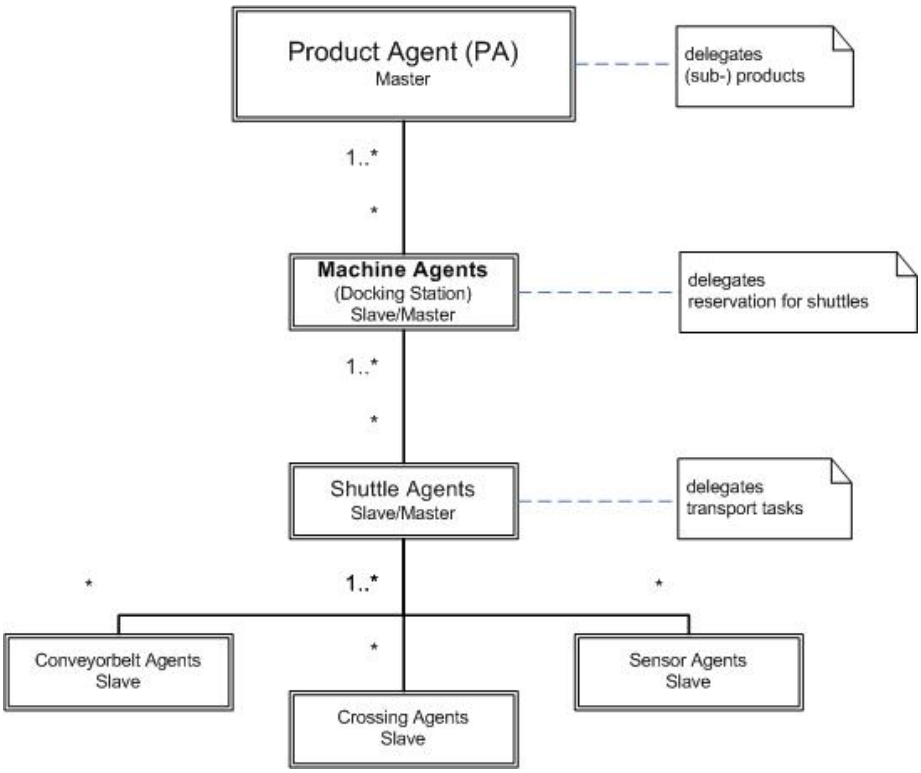
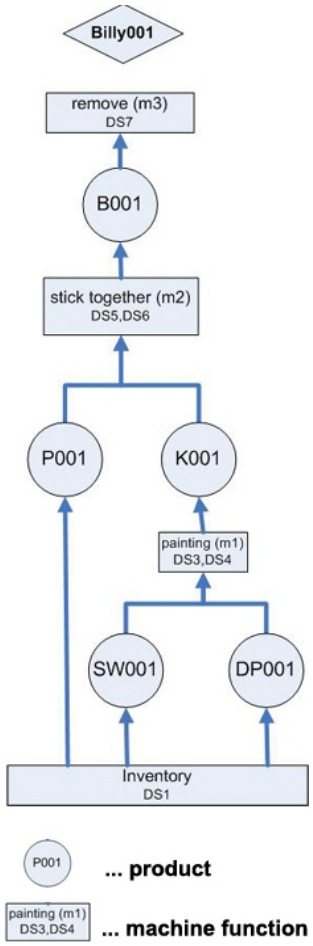


figure 14: Master/Slaver relationship in SAW-project

The PA gets a product plan it is responsible for. It analyses the plan and divides it into different machine functions and (sub-) products. In the next steps it sends messages to the simulation agents, concretely to the machine (docking station) agents. These agents are the Slaves of the PA and have to fulfil the tasks forwarded to them.
The machine agents adopt the role of the Master in contrast to the shuttle agents, which represent the Slaves of the machine agents. The shuttles have to transport a (sub-) product to a certain destination. Therefore they act as Master and use the conveyor belt, crossing and sensor agents as Slaves.
The simulation agents (Slaves) send messages to their Masters as soon as they have fulfilled their tasks. In case of the PA (Master) the machine agents send back a message as soon as a (sub-) product has arrived at its destination. In this message the product identification is included, so the PA can check its product plan and marks the arriving part as “finished”. Afterwards it delegates, if necessary, the next subtask, which includes the transport of the next (sub-) product, to its Slaves. As soon as all subtasks are finished the PA informs the OrderAgent, which is responsible for a whole workload package, that it has finished its product plan.
The communication between the PA and the machine agents (DS) is displayed in figure 15:



1. The Master (PA) divides the product plan into machine functions and (sub-) products
2. The Master (PA) delegates sub tasks (next free shuttles should pick up) to Inventory machine (DS1)
    a. P001 à send to DS5 or DS6 to fulfil (m2)
    b. SW001 à send to DS3 or DS4 to fulfil (m1)
    c. DP001 à send to DS3 or DS4 to fulfil (m1)
3. As soon as a product arrives the Machines (Slaves) send a message to the Master (PA)
    a. P001 arrives at DS5
    b. SW001 arrives at DS3
    c. DP001 arrives at DS3
4. The Master (PA) checks the product plan and delegates the next sub task
    a. If P001 arrives wait for K001
    b. If SW001 and DP001 arrive delegate shuttle to transport K001 to DS5
5. Machine DS5 (Slave) sends message to Master (PA) that K001 arrived
6. Master (PA) checks the product plan and delegates the next sub task
    a. If K001 and P001 arrive delegate shuttle to transport B001 to DS7
7. Machine DS7 (Slave) sends message to Master (PA) informing that B001 arrived
8. Master (PA) knows that the product has been finished

figure 15: Master/Slave (PA/machine) interaction in SAW project

4.2.3 Blackboard Pattern

The Blackboard pattern provides a centralized messaging/data mechanism. A Blackboard is used to hold data; meanwhile other agents can subscribe to get access to the data of their interest.

In the project agents have to solve a production problem. Therefore they have to transport certain (sub-) products to different destinations (machines). At the end all transport mechanisms and machine functions have to be fulfilled. During the problem solving process the agents have to collaborate with each other to know the different states and tasks, which have been finished already. The coordination of the individuals which have various abilities and tasks to complete is realized by a messaging system. The agents send messages as soon as their state changes or an event is deployed. A subscription function allows the agents to receive just the information which influences their behaviour. They do just get the messages of their interest and so an overload can be prevented.

For example the *Product Agent* (PA) is responsible for the production of a certain product. Therefore it has to communicate with the agents system and monitor if the simulation

agents have fulfilled all the tasks it has delegated to them. During the assembly process the simulation agents send messages to other agents to publicize their states and actions. The PA is just interested in messages of machine agents because it just needs to know when a (sub-) product is removed from the simulation. Therefore it is subscribed to machine agent messages. Besides the sender filtering the PA has a message type filtering.
The next lines show how the PA is subscribed to the machine (docking station) agents and filters the message types (removeWorkPiece):

```
for(int i=0; i < dockingStationList.size(); i++) {
      this.sendSubscribe(dockingStationList.get(i).getName(),
                        "<events><removeWorkPiece/></events>");
}
```

listing 3 : Java implementation of agent subscription (CEBIT project)

The subscription method needs the agent identification (ID) and the message type. Like in the case of the PA (like in listing 3) it would be all machine agents and the "removedWorkPiece" messages.

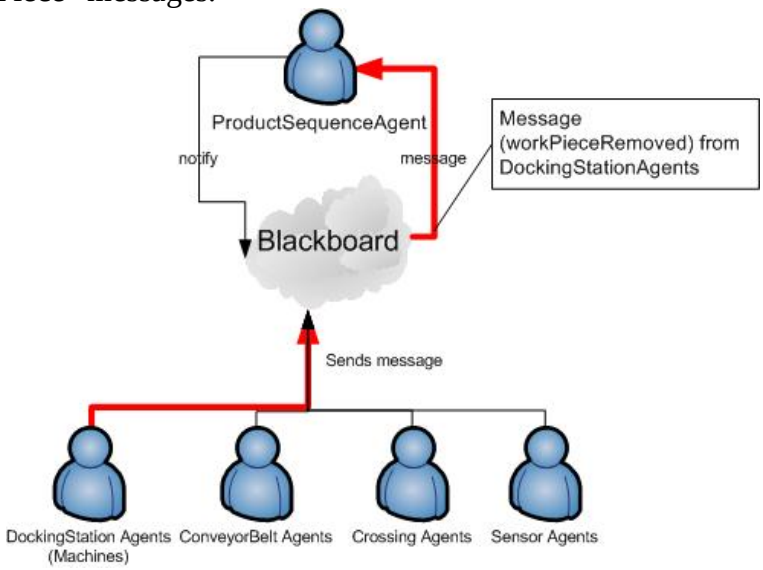


figure 16: Interaction in Blackboard Pattern – Docking Stations and PA

The Agent Communication Language (ACL) from Java is used as communication protocol. The communication mechanism was implemented by Rockwell Automation and is also used by the extended Agents as well.

In the Blackboard pattern defined by Deugo, Weiss and Kendall [19] the agents can add data to a Blackboard and subscribe to information of their interests. The agents do also have the possibility of changing data and monitor the Blackboard. They can get access to the data of their subscription any time.
Such a Blackboard has not been implemented in the project. The agents do just send messages to their partners. There is no kind of Blackboard where the messages are saved. As soon as a message is sent, the agents have to react and answer.

The current messaging system does not provide a container where messages can be saved and accessed at any time. A memory would allow picking up data and messages whenever needed or update them. To provide such a Blackboard the messaging system can be combined with a data memory, or the like.
Another missing component in the project in contrast to the definition of Deugo, Weiss and Kendall [19], is the Supervisor. A Supervisor has the responsibility and control over the Blackboard. It decides which agents can modify the data and knows when a task has been fulfilled.

4.3 Implementation of Production Planning Process for MAS simulation

Production planning is a core part of manufacturing systems. Its goal is to minimize the production time and costs, maximize the efficiency of work as well as the consumption of available resources. During the production process an output according to a production plan is produced. The production process is linked with strategic operation goals and coordination operations. Besides strategic decisions which influence the definition of the business areas and the linked production strategies, tactical decisions of the complexity of the production plans and the basic structure of the production system have to be calculated [31].

The complexity of manufacturing systems increase more and more in the last years. Older systems were small enough to handle the processes with technical advances and humans. Today such systems have to be controlled by information technology systems. The huge number of machines, employees, information, states and roles needs a centralized workflow management tool to ensure faultless processes.
With growing knowledge about information systems the usage of them in companies has increased. Meanwhile the whole production systems can be simulated.

One solution for mapping production planning processes into the digital world is a Multi Agent System, as described in this thesis. It “brings a radically new solution to the very concept of modelling and simulation in environmental sciences, by offering the possibility of directly representing individuals, their behaviour and their interactions” [9].
The Manufacturing Agent Simulation Tool (MAST) is used as a suitable tool for testing production plans for an empirical study. The tool provides a simulator consisting of different individuals/agents like assembly machines or conveyor belts. The agents within the system are autonomous units, which work as subsystems to execute their tasks for the overall system. One goal of the simulator is to analyse the different workflow scheduling strategies which can be used to fulfil different orders. Another important factor during simulation runs is the machine utilization rate. Companies usually want to have a full utilization because of high operation costs for machines and personnel, especially during idle time. Besides these first test issues tests with different shop layout and simulation runs with different parameters can be tested.
Therefore a simulator, implemented by Rockwell Automation, is extended and test cases are defined.

4.3.1 Business process cycle

The business process cycle of this project is divided into different layers and roles. Each role has different responsibilities and passes information to the other roles. Such a layer model is shown in figure 17. The information flow can be seen either top-down as well as bottom-up.

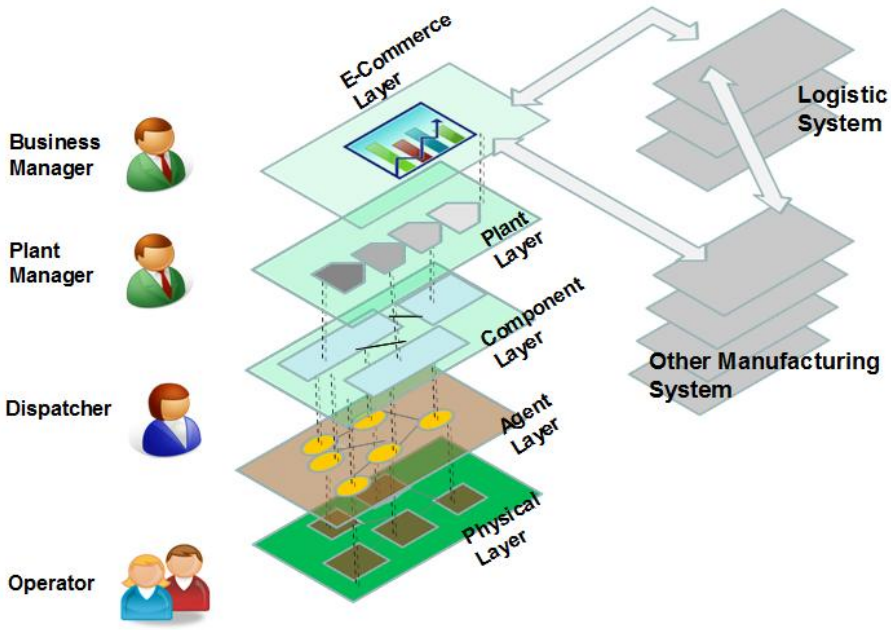


figure 17: Layermodel for assembly workshop

The *Business Manager* gets orders from customers and prioritizes them to get certain sequence of business orders. She/he forwards the plans to the *Plant Manager*. In return the *Business Manager* gets management ratios from the lower layer among.
The *Plant Manager* gets the production orders from the layer above and analyses if they can be realized in the scheduled time. Therefore she/he evaluates the existing resources and transfers the business orders into work orders. Next she/he allocates the order to a certain shift.
The *Dispatcher* is responsible for the mapping of the orders to the physical layer. She/he has to ensure that all products in the current shift are produced. Therefore she/he divides the order into different working steps and production steps. She/he is connected with the physical layer and can influence the production by manipulating different input parameters or work orders.
The *Operators* are responsible to accomplish the assigned production steps. In the project the physical layer is represented by a simulator.

4.3.2 Implementing the physical layer

In a manufacturing oriented production system a lot of machines with different functions and abilities are needed to produce certain products. Besides these resource problems the fast changing requirements have to be adopted into the system which may lead to reconfigurations of production systems. “The ever fast changes of customers’ needs and demands ask for reconfigurable and adaptive production systems, which can provide

companies with the proper level of agility and effectiveness, without disregarding at the same time cost factors" [32]. The flexibility and effectiveness of MAS provides one possible solution to such complex and distributed production planning systems. The agents within MAS can act as different roles. "Each role is described by a set of possible behaviour" [33]. This ability allows configuring agents for different working steps. In a manufacturing cycle, for example, different machines have to fulfil various functions. An assembly machine works on products while conveyor belts transport units from one point to another. The behaviour of agents is linked with their internal states. The states can be changed by interactions between agents or events, which influence the agents' activities. Besides the different states the agents usually act as problem solvers. This means that they have to fulfil certain tasks. These tasks are solved with different behaviours, which are connected with the various states. The interaction between the tasks and the states is shown in figure 18.

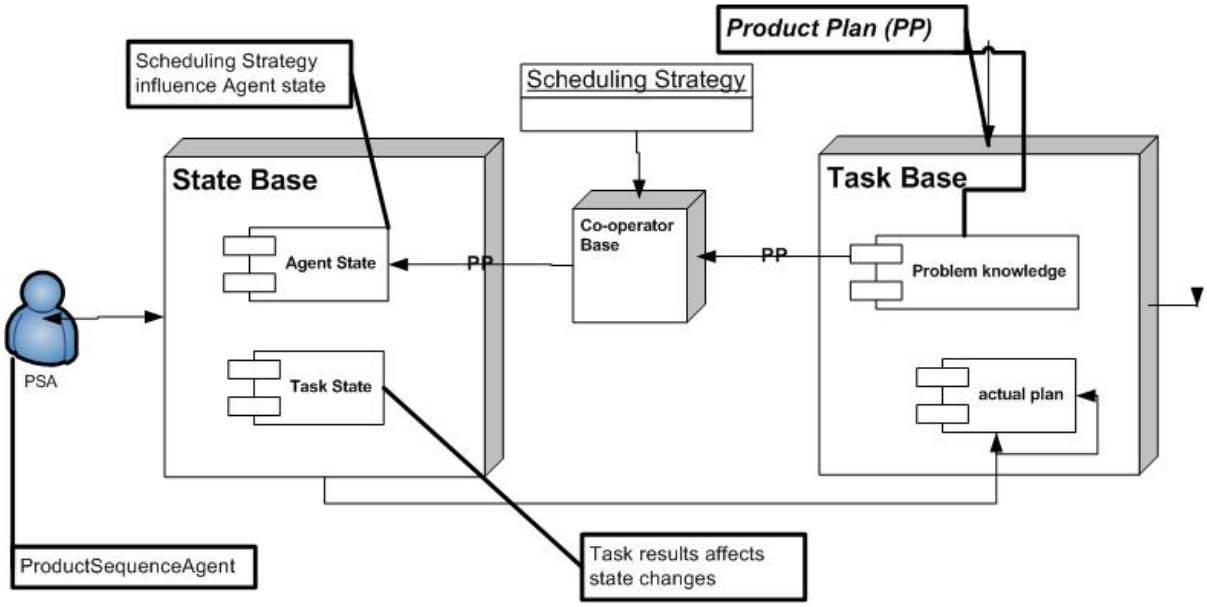


figure 18: State and Task base of an agent (OA) [11]

One example in the production planning process of the project is the *Order Agent* (OA). It is responsible for an order and adopts a set of product plans to fulfil. These plans represent the task base and describe a knowledge problem to be solved (figure 18). The workshop scheduling strategy affects the order of the products. The OA is set to a new state where it registers *Product Agents* (PA) for each product plan. The PAs send back their results and the OA checks its actual plan if all product plans have been finished.
The scheme of figure 18 can be mapped to each agent in the simulation. An agent gets a task to fulfil. This task sets the agent into a new state where it can solve the problem.
With the possibility to create different agents with various abilities it is possible to implement a complete manufacturing system with MAS. To read more about the used MAS for the production control system in the project have a closer look at 4.1 "Implementing Multi Agent System".

4.3.3 Implementation of production strategies scheduling

The implementation of production planning processes helps to analyse different strategies. The workflow scheduling strategies influence the order of the steps done in the production process. They have to be predefined and combined with an order that has to be fulfilled. An order consists of a certain set of products and due dates. A dispatcher is responsible for the order and has to decide which steps have to be realized next. This decision is based on workflow scheduling strategies.

The decision making process is an optimization problem. It is difficult to get a clear answer without testing various parameters. It is important to have various optimization methods to get different results to compare. Depending on the optimization goal the test results of the methods differ from each other. In case of production planning the number of finished products and the machine utilization can play an important role. Therefore various workflow scheduling strategies are defined, which influence the simulation flow. Thus a certain number of alternative results and several conflicting performance criteria can be analysed.
To provide a simulation environment that allows changing the strategies without having too much effort, the strategies have to be decoupled. figure 18 shows such an environment where the scheduling strategy is a separated component.
During the implementation part of this thesis eight various workflow scheduling strategies for two different applications (SAW, ACIN) have been implemented. Further strategies can easily be added to the project. They just have to be registered in the input parameter file and the dispatcher has to know about them.

4.3.4 Priority Scheduling

Manufacturing Companies often have to prioritize their customers' orders. These processes are often done daily or even several times per day. This reorganization of the scheduling order does always depend on the production situation. The dispatcher, who selects the next production steps, has to calculate the priority index of each order. This index can include a lot of different data, for example resource information, holding costs or tardiness penalties. For the classification of the priority rules, order information and an index are used as the criteria. Rates, like the earliest due date or shortest machine function time, are easy to classify but there are sometimes some more complex calculations that have to be done. For example a classification of the "importance" of a user, or the balanced satisfaction for all costumer, can also influence the decision making process. Kemppainen [34] therefore defines the following classification matrix:

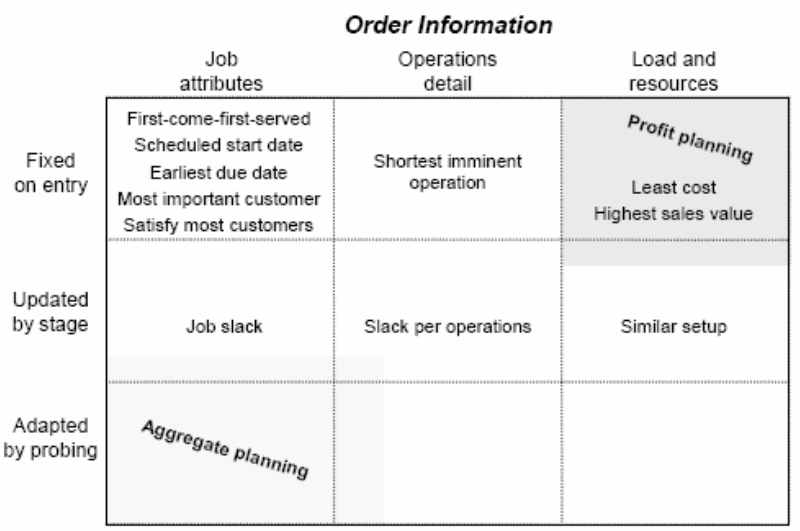


figure 19: classification of the dispatch priority rules (slack = time left from current time to due date)

The matrix shows that the main focus of the priority rules on the upper-right position in the matrix is on profit making. These factors just analyse the financial rates. In contrast the rules in the lower left corner use the information of jobs, operations and production load and delivery. Depending on the different rates and factors, the complexity and the effect of the various priority rules differ from each other.

4.4 Simulator (Prototype)

The simulator is the main part of the practical work of this diploma thesis. At the beginning there is a listing of the requirements that should be realized. Afterwards the existing simulator is analysed and a design overview of the whole application is given.
The next sections of the chapter include technical information about the application itself. First the incremental way of the implementation of the tool is explained. Before going into the details of the agent communication and the general agent class, the simulation architecture is described.
The prototype is developed for two different target audiences. The two different prototypes are compared in the next subchapters. At the end of the chapter and for the proof of concept, a data analysis of simulation results is presented.

4.4.1 Requirements

Goal of the demonstrator was to simulate an automated assembly workshop system. It helps to analyse the affects of different workflow scheduling strategies. Therefore a Manufacturing Agent Simulation Tool (MAST) based on MAS is used. It is an effective way to simulate and test workflow scheduling strategies. The assembly processes were defined in workload packages which include a certain number of products. Therefore example scenarios with a set of different products were defined. The structure of such a product is displayed as a product tree with different (sub) products and machine functions needed in order to assemble the product.
To fulfil a workload package, the existing simulation tool, developed by Rockwell Automation, is extended by a set of coordination components. To allow different research

results the input parameters of the simulation are flexible and different production strategies are available.

4.4.2 Existing Simulator

The existing simulator is called MAST and has been developed by Rockwell Automation Research Center situated in Prague, Czech Republic. It is programmed in Java and based on the JADE platform. The tool simulates a material handling system using basic components like conveyor belts, diverters or manufacturing cells. The combination of these components in a simulation GUI allows configuring the layout of different material handling systems. MAST is used to simulate various manufacturing tasks, for example the transportation of products between different machines. Therefore the agents within the simulation have to communicate with each other and have to be coordinated. In MAST, the agents exchange messages for coordination.

Another important ability of the tool is the failure detection and recovery. By adding various simulation failures, like stopping a conveyor belt, the reaction of the agents and system can be monitored. The analysis of these situations can help to increase the efficiency of such material handling systems.

The existing simulator was extended during the work on this thesis. A work order scheduling system and a test management component were added. The simulation agents were extended and a new agent was added. The next section gives an overview of the project design.

4.4.3 Design

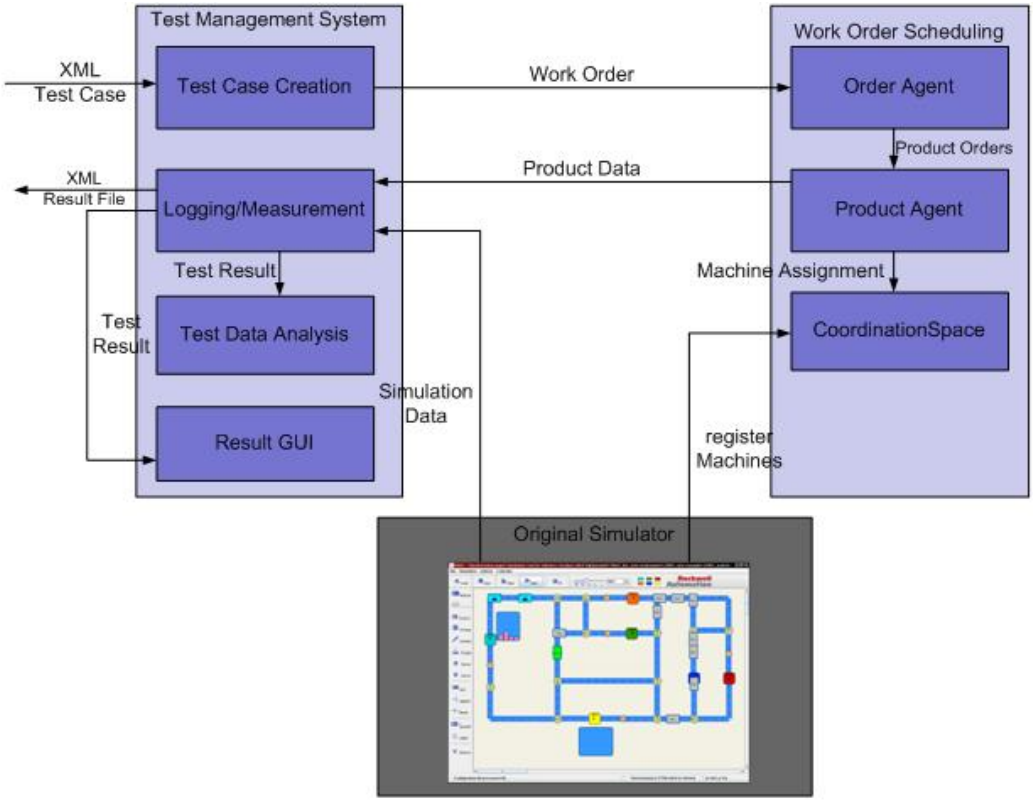


figure 20: application design

The demonstrator consists of three main parts:

- Test Management System
- Work Order Scheduling
- Simulator

a.) Test Management System

The test management system basically is responsible for reading test cases out of a test case file, for forwarding the test cases to the work order scheduling system and for the measurement of the simulation results. The test management system consists of the following four parts:

- Test Case creation
- Logging/Measurement
- Test data Analysis
- Result GUI

Test Case creation

The test cases are defined in files. Therefore an interface is implemented which reads the data of an XML file. The test case file includes different input parameters.
The input parameters set the simulation into a defined state. The following parameters can be set:

- Workload packages à list of products to be produced, including product type and due dates for all products
- Strategy à strategy used to fulfil the order
- Transport Speed à regulates the speed of the whole simulation
- Set of Palettes à number ob palettes on the simulation
- Inventory à use an inventory with predefined products inside
- Shift à duration of a shift

All input parameters, except the workload package, set the simulation in a certain state. The workload package is forwarded to the work order scheduling system, where the fulfilling of the workload is organized.

Logging/Measurement

To get test results from the simulator the system has to log different events/states that occur in the simulation. Therefore a scoring system is needed which defines which events are important for the measurement.
Another advantage of logging is the reproduction of a certain simulation state when an unexpected failure occurs.

Test data analysis
After getting a lot of test data from the simulation, the data has to be analysed. This is important to be able to compare different results and to get the knowledge of how different input parameters affect the results. Depending on which person the results are presented to, different information are important. Therefore algorithms are used to get a value for comparison of the huge amount of data.

Result GUI
To present the result of the test data analysis the values have to be presented in an understandable way. Therefore a scoring system and ranking mechanism is implemented to show the current test result compared to other test results.

b.) Work Order Scheduling

The work order scheduling component is the main part of the system. It is responsible for fulfilling different tasks. Therefore two Agents have the responsibility to control the production and react on different events happening in the simulation:

- ProductPlanAgent (à OrderAgent)
- Product Agent

ProductPlanAgent
The Product Plan Agent is responsible for a whole order. For each order, including *n* products plans, a Product Plan Agent is registered and controls the production. Every product plan is represented as a Product Agent.
By contrast to a Product Agent the Product Plan Agent is not communicating with the simulation and does not receive messages from the simulation agents.

Product Agent
The Product Agent is the most important part of the extensions to the Rockwell simulator. It gets a product plan as an ArrayList and is responsible for the whole production of one product. Therefore it communicates with the simulation agents and decides which step has to be done next. So it is the most intelligent agent in the system.
For a detailed description of the important methods in the Product Agent see chapter 4.1.2.

CoordinationSpace
The Coordination Space [35] [36] is a container where all the agents are registered. To get the reference of a certain agent the identifier of the agent has to be known.

c.) Simulator

The simulator is a development of Rockwell which can be modified in any way. For our project the simulation was extended by a further simulation agent and a set of other agents. A Sorting Machine is responsible for controlling the order of products arriving at a destination. Therefore it is placed in front of assembly machines where it can send products into a waiting loop if necessary.

The application is implemented in an incremental process, as described in the next subchapter, where step by step components and agents have been added and modified.

4.4.4 Incremental Software Process for MAS

The incremental development is a software process [37] that, in contrast to the waterfall model [38] or the V-model [39] [40], does not assume that the analysis process has already been finished before starting with the implementation. The advantage is that the implementation can be realized incrementally and a faster reaction on changing or added requirements is possible. Sub parts of the system can be used although other parts are still in the implementation phase.
These abilities provide a good background for the implementation of MAS. As soon as a group of coherent agents are completed they can be tested and do not have to wait for the whole agent system. In the further development of the system, agents or components can be added as long as the whole system is implemented. Another important ability of the incremental process for agent-based systems is the easy reaction on changing requirements. As soon as an agent has to fulfil another function or has to be extended it can be modified or exchanged.

The SAW project is also implemented in an incremental way. The programming process in this thesis is divided into five iterations: Simulator, Simulator extended, Work Order Scheduling, Test management and visualization.

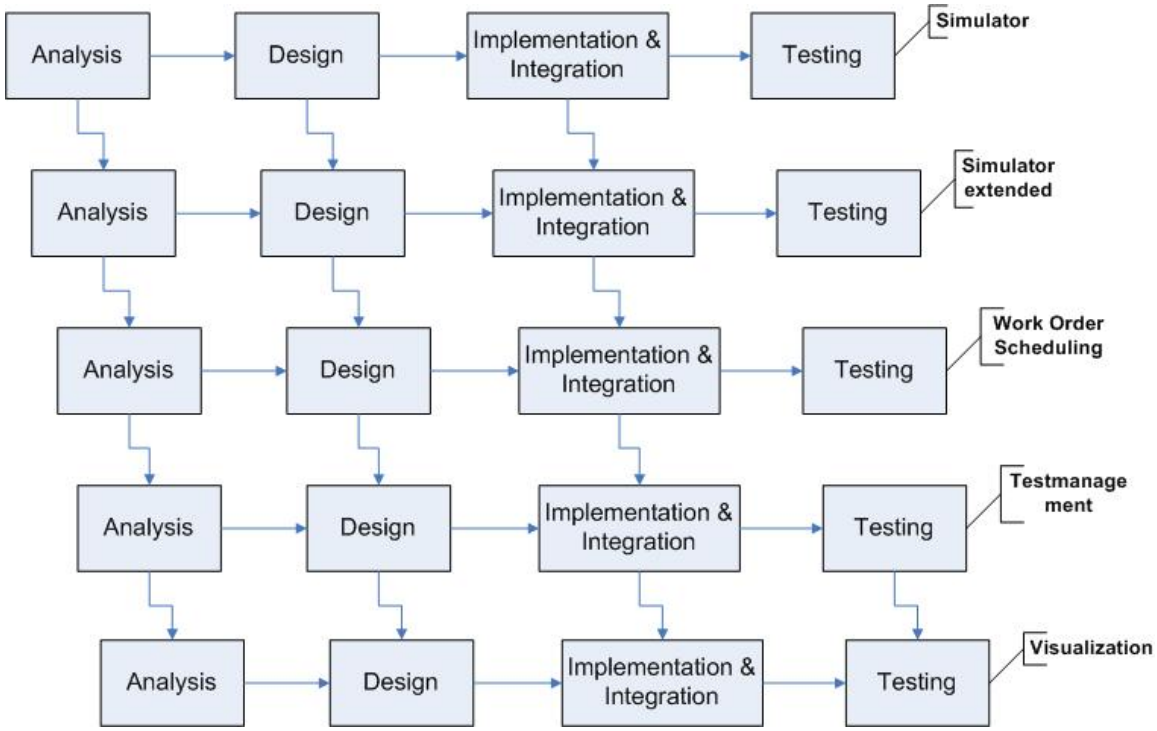


figure 21: incremental software process (SAW project)

Simulator
The first iteration includes the simulator developed by Rockwell Automation. At the beginning the system is tested and analysed. The requirements are defined and compared with the given implementation. The comparison showed that an agent, which is responsible for the order of products arriving at an assembly machine, is missing. Besides the new agent, the existing ones have to be extended by various methods. Therefore some extensions and changes have to be done in the next iteration.

Simulator extended
The first iteration showed that the simulator has to be extended and modified to fulfil all requirements. Therefore methods are modified or added to the existing agents. For example the docking station agent now can have two different roles: inventory or assembly. The agent library is also extended by a new sort of agent. The sorting machine controls the shuttles going to an assembly machine to keep defined product in order. After the implementation of the new or modified components the whole system is tested.

Work Order Scheduling
When the implementation of the simulator is finished the next component is added. To control the workflow the simulation is extended by coordination agents.
The first component is a *Product Agent* (PA). It is responsible for the manufacturing process of one product. To map a whole order, including a certain number of products, the agent system is extended by another coordination agent. The *ProductPlanAgent* (OrderAgent) gets an order and registers a PA for each product.

Test management
After finishing the coordination component the test management is implemented. The test management consists of the test case interface and the measurement part. The test case files are using XML format. An interface reads the file and forwards them to the work order scheduling part. The test management interface is separated from the simulator and the work order scheduling part.

Visualization
At the end the visualization, like the improvement of the production process, is added. The visualization is partly connected with the measurement component as well as directly with the simulation. Beyond the different interfaces to represent the simulation results, a data analysis with defined algorithms calculates the various measurements.

a.) Benefits
The incremental software engineering process has the big advantage that sub parts of the system can already be used although the whole system has not been designed. During the thesis the application was presented to the “end-user” at the CEBIT. Meanwhile the system was extended by new workshop scheduling strategies or data analysis algorithms. These changes do not affect the simulation and are added easily.
For future steps the system can be extended easily by components and agents of any kind.

b.) Limitations

One problem during the whole implementation phase is the difficulty to change or extend the existing architecture. This limitation does occur mainly if the development process already has an advanced status. To avoid such problems the requirements are clearly defined before starting with the design.

4.4.5 Simulation Architecture

The simulator provides transparent demonstration, redundant transportation paths and machines with overlapping capabilities based on MAS. An editor allows configuring the transportation network in different ways. The demonstrator is implemented in JAVA and uses the open source JADE (Java Agent Development Framework) agent platform as agent development and runtime environment. The software simulator is implemented parallel to the physical simulator of the ACIN laboratory. The simulator consists of four different parts:

- The simulator is based on *the library of agent classes* which represent the basic material handling components. Each agent type has individual abilities which are autonomously controlled by supervised manufacturing equipment and by the coordination with other agents. The agent library includes a docking station (machine), a conveyor belt, a sensor, a crossing and a storage agent. Further it is extended by another agent, the sorting machine, which is responsible for sorting problems within product plans.
- The simulator is controlled by *the simulation engine* which provides the different behaviours, like transport steps, of the demonstrator.
- The *Graphical User Interface (GUI)* provides a visualization of the agent actions within the simulation. The agent decisions, like the routing of a shuttle, can be observed by the user.
- The communication between the agents and the simulation engine is controlled by *the control interface*. It allows the agents to react on various states within the simulation. For example the routing is influenced by conveyor belt failures.

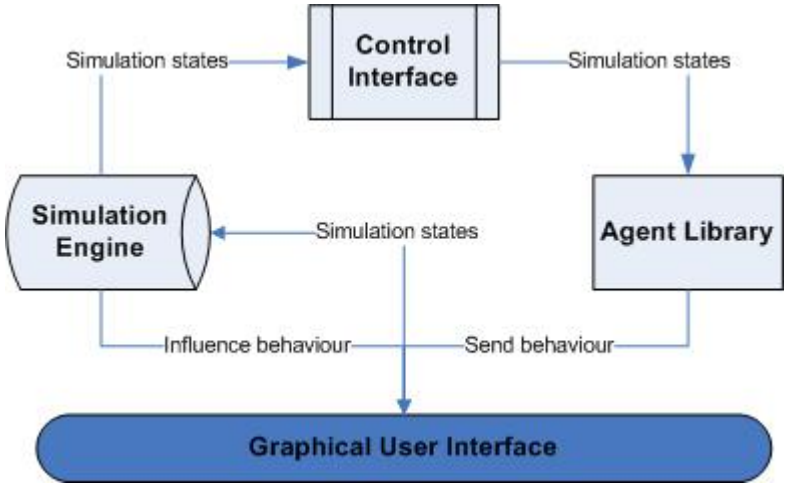


figure 22: Simulator Architecture

Figure 22 displays the interaction between the different components of the simulator. As soon as the state within the simulation changes, the simulation engine informs the agents using the control interface.
The communication between the agents is realized by sending and receiving messages based on ACL. The next section deals with that message language.

4.4.6 Communication

The communication between the agents is realized by means of ACL. It is a communication protocol working on the application layer. The protocol is used to exchange information. Within the message a clear separation between the content and the purpose of the message is given. So the protocol can be used to request for a certain service as well as to send a reaction on a demand. Table 2 displays the different message types.

Purpose	**Description**	**Content**
INFORM	informs about a certain assumption	Information
QUERY-IF	query about a certain assumption	Information
QUERY-REF	query for a certain object	Expression
CFP	invocation for a proposal	Specific information for a proposal
PROPOSE	offer a proposal	Proposal
ACCEPT-PROPOSAL	a certain proposal is accepted	Proposal ID
REJECT-PROPOSAL	a certain proposal is not accepted	Proposal ID
REQUEST	request to fulfill a certain activity	Specification of an activity
SUBSCRIBE	Register for a certain source of information	reference

Table 2: different message types in ACL [41]

The content and the purpose of the messages guarantee that both, the sender and the receiver, have the same expectation of the concern of the message. Therefore the reaction on the message should be associated with senders' message.
The message consists of a header and the content. The header includes the message type and the ID of the sender and receiver. Additionally it can include a closer description of the used language and encoding schema. If the agents do not understand the content of the message, a standardized description about the vocabulary can be added. This is realized by the usage of an ontology.

Initialization of an ACL message

The ACL message is initialized by the jade.lang.acl.ACLMessage class. The message can include various numbers of parameters. Which parameters are needed precisely, does depend on the situation. Each message has to include the "performative" element, which defines the type of message (e.g. INFORM or REQUEST), and informs about the sender

and the receiver. The whole list of the message parameters and their description can be found in the FIPA ACL specification [42].
listing 4 shows the way an ACL message is created in the SAW project.

```
1 public String sendMessage(int performative, Object receiver, String
2 content){
3
4     // creating ACL message
5     ACLMessage msg = new ACLMessage(performative);
6     msg.setSender(this.getAID());
7     msg.addReceiver(getAID(receiver));
8     msg.setOntology("material-handling");
9     msg.setLanguage("XML");
10    msg.setContent(content);
11    String convID = getUniqueConversationID();
12    msg.setConversationId(convID);
13    msg.setReplyWith(convID);
14    …
15    send(msg);
16    return convID;
17}
```

listing 4: creation of an ACL message (SAW project)

The “performative” parameter (integer) in the ACLMessage class defines the type of message (e.g. 7 = INFORM). Subsequently the sender and the receiver are defined. The sender is the agent which creates the message. The getAID() method returns the unique agent identification.
Line eight and nine are needed for the interpretation of the content which has to be converted in XML. The most important part of the message is defined in line ten, where the content is added to the message. The following two lines create and set a unique conversation ID for the agent. Another interesting line is the thirteenth, where the receiver for the reply message is defined.
After the creation of the message, it is sent by the *send()* method.

Message content

The following example is a message used in the SAW project. It is sent when a shuttle with a product arrives at its destination and the product is removed from the simulation. Therefore a “removedWorkPiece” message is sent by the destination machine.

```
1 (INFORM
2 :sender  ( agent-identifier :name DS5@ClemensPC:1099/JADE  :addresses
3 (sequence http://ClemensPC:7778/acc ))
4 :receiver  (set ( agent-identifier :name
billy_medium21@ClemensPC:1099/JADE  5 :addresses (sequence
```

```
http://ClemensPC:7778/acc )) )
6 :content  "<removeWorkPiece>
7 <workPiece ID=\"Shuttle5\" source=\"DS2\" destination=\"DS5\" />
8 </removeWorkPiece>"
9 :reply-with  DS5@ClemensPC:1099/JADE-13  :language  XML  :ontology
10 material-handling  :conversation-id  DS5@ClemensPC:1099/JADE-13 )
```

listing 5: received message from the Product Agent

The first line describes the message type, which is an INFORM. The second and third lines include the information about the sender. The first element (agent-identifier) defines the name of the agent (DS5) that sends the message and the second element (addresses) is the address of the agents' container (http://ClemensPC:7778/acc).
The fourth and fifth lines include the information about the receiver. They can be interpreted like the lines for the sender.
What follows next is the content of the message. After reading the content the receiver knows that a product has been removed from the Shuttle5 at the machine DS5. With the help of the last two lines the receiver agent (billy_medium21) knows that it has to answer to DS5 (reply-with).

For more information about the ACL specification see:
http://www.fipa.org/specs/fipa00061/index.html

4.4.7 MAST Agent

The *MASTAgent* is the general agent in the SAW project. It was implemented by Rockwell Automation and is an extension of the jade.core.Agent class. Each other agent in the system is extended by it. It includes the basic attributes and methods of an agent.

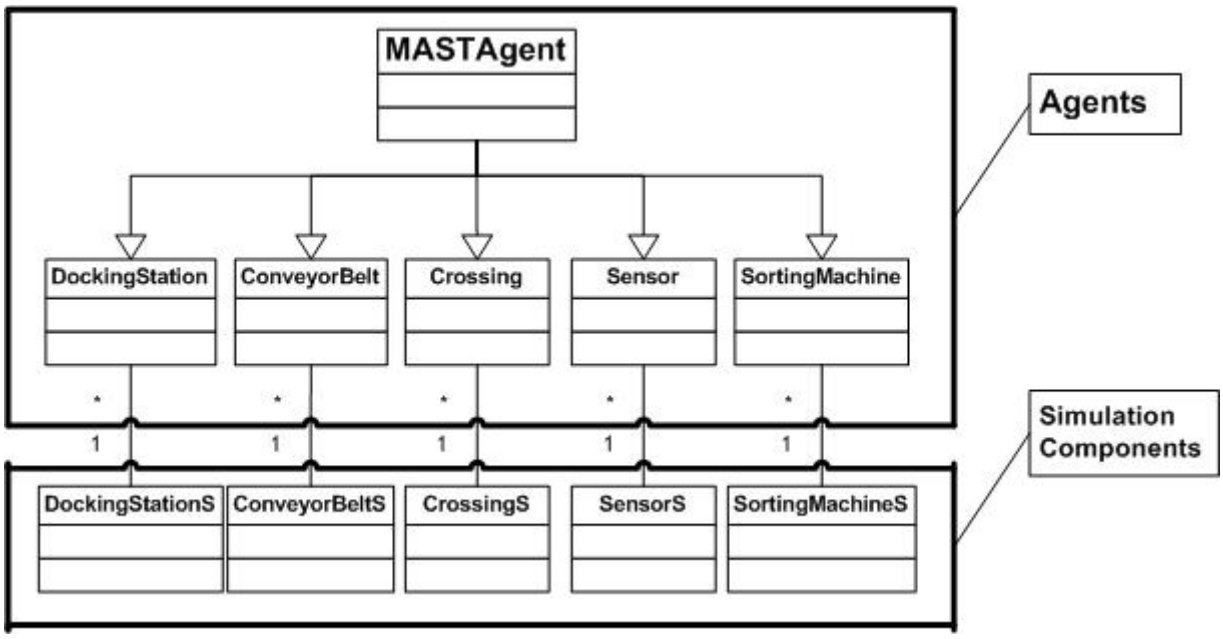


figure 23: agents and their component classes

Attributes
The most important attributes are listed here:

- name (String) : name of the agent
- services (LinkedList) : list of services of the agent
- thisAgent (MASTAgent) : reference to the agent
- thisAgentAID (AID) : Agent Identifier, unique name
- subscribeList (SubscribeList) : list of subscribed agents
- guiComponent (ComponentS) : pointer to the component in the simulation

Methods
The basic methods of all agents are defined in the MASTAgent. The following list includes some of the important methods:

- *getUniqueConversationID()*: The method generates a unique conversation ID, consisting of the agent name, platform name and a counter. It is used for the communication with other agents.

- *informSubscribedAgents()*: The method informs all agents, which are subscribed to the current one, about an event.

- *getSubscribedInfo()*: A list of subscribed agents is returned.

- *getAID()*: The method returns the unique agent identifier.

- *sendMessage()*: The general method to send a message of a given type of message to a certain receiver. Depending on the input parameter a message can be sent to one or more receivers.
  Input parameter
  - performative (int), the integer value defines the type of message (e.g., 7 = INFORM)
  - receiver (Object or Object[]), agent identifier of the message receiver(s)
  - content (String), content of the message

- *sendReply()*: The general method to send a reply message with a given type. The MASTAgent has two different sendReply() methods. One includes content and the other not.
  Input parameter
  - originalMessage (ACLMessage), message the was received and is replied
  - performative (int), type of message
  - content (String), content that is sent back – optional

- *sendInform()*: Sends a INFORM message either with a given conversationID or with a new one. The method invokes the sendMessage() method described above.
  Input parameter
  - receiver (Object), receiver of the message
  - content (String), content that is sent back
  - convID (String), conversation ID – optional

- *sendRefuse(), sendCancel(), sendAgree(), sendRequest(), sendSubscribe(), sendInformRef(), sendNotUnderstood()*: All these methods work the same way as the sendInform() method does.

- *handleInform()*: The method is invoked if the agent receives an INFORM message.
  Input parameter
  - msg (ACLMessage), message that has been received

- *handleNotUnderstood(), handleSubscribe(), handleCancel()*: The methods work the same way as the handleInform() method does.

4.4.8 Load Balancing

Load balancing in manufacturing systems is a method to divide work between two or more redundant machines. Machines are called redundant if they have the same abilities, which mean that they can execute the same functions. This parallel production possibility can help to increase the productivity within a production system. Additionally if a machine has a failure the production flow can be rerouted to another machine.
In the SAW project load balancing controls the machine utilization of the assembly machines. Each product has a list of destinations where it needs to go to. If the list consists of more than one machine entry, it would mean that there are two or more machines which are redundant. For the decision, which assembly machine will be the destination, two factors have to be considered.
First the product tree of the product is analysed. If the product has a sibling and this sibling has already been sent to a machine or is waiting there, the product has to be sent to the same machine. If this case does not occur the input queue of each assembly machine has to be analysed. The virtual input queue is a list of products that should arrive at a particular. To guarantee maximized machine utilization the machine function times of the product in the list have to be summed. The product is sent to the machine with the lowest overall value.
To optimize the load balancing, the calculation of the overall machine function time can be extended. During the decision making process the sibling products of the current product, which have not been sent to a machine yet, can be included into the calculation. This calculation would provide a forecast about the expected machine utilization.

$$U = \sum_{i=1}^{n} Pm_i + \sum_{i=1}^{m} Sm_i$$

U … utilization
Pm … machine function of products in the input queue
Sm … machine function of sibling products that has not been arrived

listing 6: formal for calculating the machine utilization

4.5 Proof of Concept: CEBIT versus SAW

The two applications were implemented for two different target audiences. Therefore they differ in their requirements but both are based on the same Multi Agent System. The main difference between the two applications is the user. The CEBIT project is developed for an "end-user" at the CEBIT exhibition and the SAW project is for researchers.
Besides these differences, the used strategies are various. These strategies and the machine utilization have the highest priority during the data analysis.

4.5.1 CEBIT Project

This version was presented at the CEBIT 2008. It helps the user to understand the impact of tactical decisions in small but efficient automated production systems. The users could act as dispatcher deciding which strategy is used to fulfil a certain order. An order consists of a package of products. Depending on the complexity (very low, low, medium, high, very high) of the package the number of products and their complexity (simple, medium, high) differs. To get a feeling of how strategies and parameters influence the simulation run, a scoring system is implemented. It shows the user the results compared to other combinations of a strategy and parameters. To calculate these results a simple data analysis is used.
As mentioned before the project is designed for "end users" at the CEBIT. That is why the visualization is one of the main parts. The user interacts with the application over several interfaces. These interfaces have to be designed in an easily understandable way, because the user should not need to spend a lot of time to study the screen before knowing what she/he has to do next. figure 24 shows the input parameter interface which was used at the CEBIT:

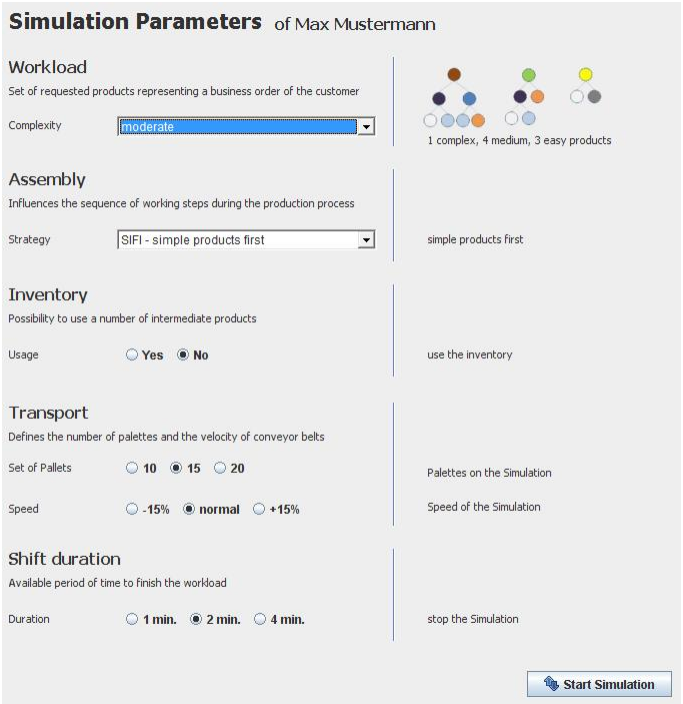


figure 24: Input parameter Interface (SAW Project)

Most important for the simulation results is the choice of the workflow scheduling strategy. For the SAW project four different strategies are implemented:

- SIFI (Simple First)
- COFI (Complex First)
- MODI (Most different)
- MOFI (Most finished)

SIFI (Simple First)
The simple first strategy analyses the workload package and sorts the products by their complexity. The simple products get the highest priority and will be produced first. The most complex products get the lowest priority and will be produced as last.

COFI (Complex First)
This strategy works the same way as the SIFI, but with inverted priority rating. Complex products will be produced first and the simplest one as last.

MODI (Most different)
Goal of the MODI strategy is that at the end of the shift the outcome consists of as many different products as possible. Therefore the strategy starts with the product having the highest quantity.

MOFI (Most finished)
MOFI is a strategy that has the most efficient throughput, which means as many products as possible are finished at the end of the shift. Therefore a bottleneck analysis compares the machine function with the transport time in order to maximize the machine's usage, which means to keep the input queue filled.

4.5.2 SAW Project

The SAW (**S**imulation of **A**ssembly **W**orkshops) project is a research tool for simulation, measurement and data analysis. It is used to study dynamic scheduling strategies with the attention on parallel machine scheduling. This means that *n* tasks depending on different constraints, like capacity, are accomplished on *m* machines. The parallel arrangement of machines provides a better overall system throughput and an alternative production flow if a machine failure occurs.
In contrast to the CEBIT project it is not implemented for an "end user". Therefore visualization, like the process status, is not a key factor. The application is used to test the effects of different workflow scheduling strategies and input parameters. Especially the simulation throughput and the machine utilization are analysed. The input parameters differ from the ones in the CEBIT project. The workload parameter is extended. It has a set of 40 products where each product has a due date given in seconds.

To test the efficiency of the simulator a case study with 600 test cases is done. The test cases are a composition of the different input parameters. The results showed that the choice of the strategy has a significant impact on the test rates.
Strategies can also be defined as dispatching rules, which means sequencing the tasks that have to be done next. These rules are given in the input parameters of the test cases. Therefore four different strategies are defined:

- First Come First Serve (FCFS)
- Critical Ratio (CR)
- Earliest Due Date (EDD)
- Shortest Processing Time (SPT)

First Come First Serve (FCFS)
The FCFS strategy is the simplest where the products are produced in the sequence as they are in the input parameter.

Critical Ratio (CR)
The critical ratio strategy is expressed as the slack time (R_t) divided through the total processing time (P_t). The slack time is the time left from current time (time=0) to the due date. The total processing is calculated by summing all machine function times. This value is normally used to indicate the degree of urgency of a job. The Critical ratio rule then gives the highest priority to the job with the smallest critical ratio.

Calculating the remaining processing time
To calculate the processing time of a product, the whole product tree is worked through. Every product has a list of sub products and every product has a list of machine functions.

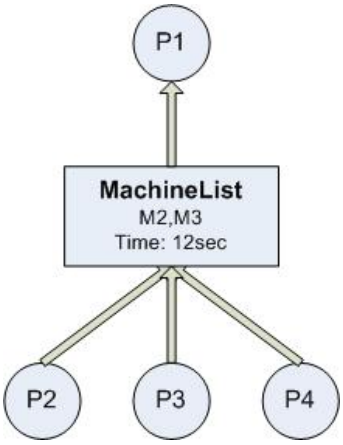


figure 25: product (P1) consisting of 3 sub products (P2,P3,P4)

Figure 25 shows such a scenario. The product P1 has three sub products which have a machine function list consisting of two elements (M2, M3). These two elements give the simulation the possibility to choose between (redundant) two machines to produce P1.

To calculate the processing time of P1, one sub product is chosen. Then one element of the function list is taken (time – 12sec). To complete the calculation the unload time (3sec) has to be added x-times, where x is the number of childrens-1.

```
processing time P1: M2 + unload + unload à 12 + 3 + 3 = 18
```

listing 7: formal for calculating the processing time of one (sub-) subproduct

Calculating the ratio

The slack time is needed first to get the ratio value. Therefore the due date and the current time is converted into milliseconds.

slack time = due date – current time

Finally the slack time is divided through the remaining processing time of the product.

ratio = slack time / remaining processing time

Ranking

The products (jobs) are ranked by the ratio value where the product with the lowest ratio is produced first.

Earliest Due Date (EDD)

The Earliest Due Date strategy depends on the due date of each product. The product list is sorted by the due dates of each product which is the time when the production should be finished. The product with the earliest due date will get the highest priority and is produced first.

Shortest Processing Time (SPT)

The shortest processing time strategy depends on the processing time, which is the sum of all machine functions needed to produce a curtain product, of each product. Therefore the processing time of all products within an order are calculated. Afterwards the product list is sorted. The product with the shortest processing time is produced first and the one with the highest is the last to get on the simulation.

4.5.3 Example: Simulation Approach SAW

The combination of all input parameter resulted in a total number of 600 test cases. Each test case consist of a workflow scheduling strategy (FCFS, CR, EDD, SPT), a certain number of pallets on the simulation (5,10, 15, or 20) and a workload of 40 orders. An order includes a product type that has to be built and a due date (in seconds). To simplify the simulation process, 3 products (simply, medium and complex) were predefined. They differ in their complexity which means the number (sub-) products and machine functions. The duration of the simulation run is set by the shift time. Each test case has 10 minutes (600 seconds) as shift time. The value was chosen to ensure that all 40 orders in each test case can be fulfilled. The Input parameters, like described before, are defined in an XML file. An interface reads the information and saves them. An example test case file can be seen in listing 8.

```
<inputparameters id="6">
        <workload>
```

```
            <order orderID="1" product="billy_low" dueDate="113"/>
            <order orderID="2" product="billy_complex"
dueDate="135"/>
            …
            <order orderID="39" product="billy_low" dueDate="190"/>
            <order orderID="40" product="billy_low" dueDate="410"/>
        </workload>
        <strategy type="EDD"/>
        <transport>
            <setofpallets number="15"/>
        </transport>
        <shift time="600"/>
</inputparameters>
```

listing 8: Input parameters for ACIN Testmanagement

Results

The goal of the test management system was to get an overview of how the different sets of input parameters influence the simulation. Therefore two kinds of measurements were carried out: A.) How do the number of pallets and the chosen workflow scheduling strategy influences *the number of finished products*; B.) The *machine utilization* depending on the strategy that has been used.

A. Number of finished products

This measurement counts the finished products during a test case in a certain time, the shift time. It measures the effectiveness of the workflow scheduling strategy depending on the input parameters. The results shown in the figure 26, point out that the number of pallets on the simulation is an important factor for the effectiveness.

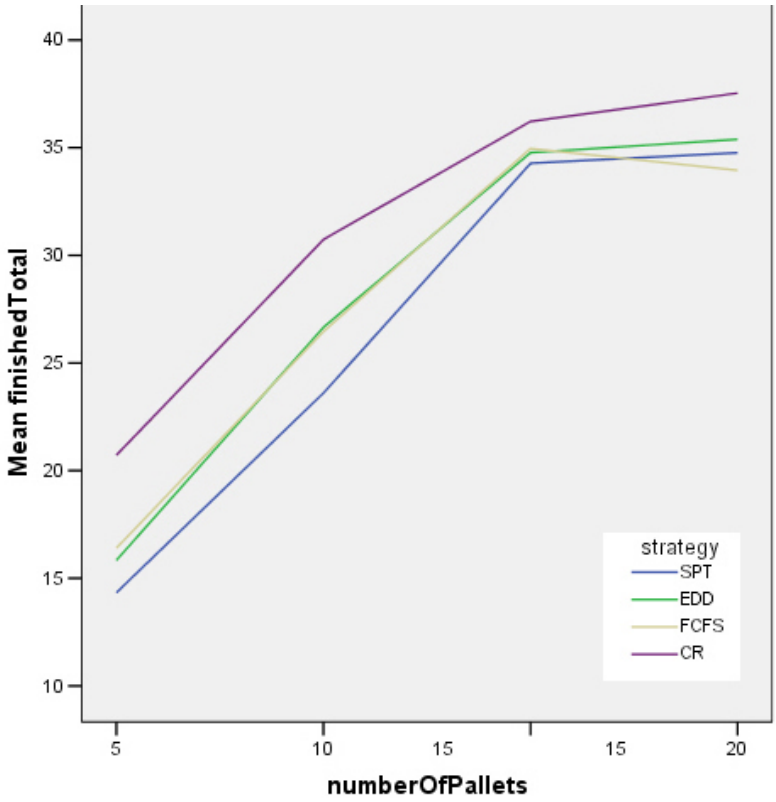


figure 26: number of finished product in contrast to number of pallets

The analysis focuses on the number of pallets used to transport the (sub-) products and the different strategies. The workloads consist of 40 chosen products (simply, medium, complex). The measurement shows that the Critical Ratio strategy results in the highest number of finished products. So it can be classified as the most efficient strategy. Generally the increase of pallets used to transport (sub-) products leads to an improvement of the overall throughput. An exception is the First Come First Serve (FCFS) strategy where the average number of finished products reduces by 2% when the pallets increase from 15 to 20. The reason for that drop of effectiveness is that during the FCFS only a limited number of pallets are needed, in contrast to the other strategies which utilize the maximal number. This means that the FCFS strategy has a better usage if the number of pallets available is low.

B. Machines Utilization Rate

In a real production system the machine utilization is an important factor, because using machine capacity is always linked with costs. The optimization of the machine utilization would result in a reduction of operation costs.

The machine utilization (MU) is defined as the actual total machine time (T_m) used, divided through the total effective manufacturing time (T_{eff}).

$$MU = \frac{\sum_{i=1}^{n} T_{mi}}{T_{eff}}$$

listing 9: formula for calculating the machine utilization

T_{eff} is the remaining available time in a certain shift minus the ramp up time (RU) and minus the cool down phase (CD). RU is the time needed to prepare the production system (simulation) for the first production step and CD is the time to send all remaining product parts back to the inventory before the shift ends.

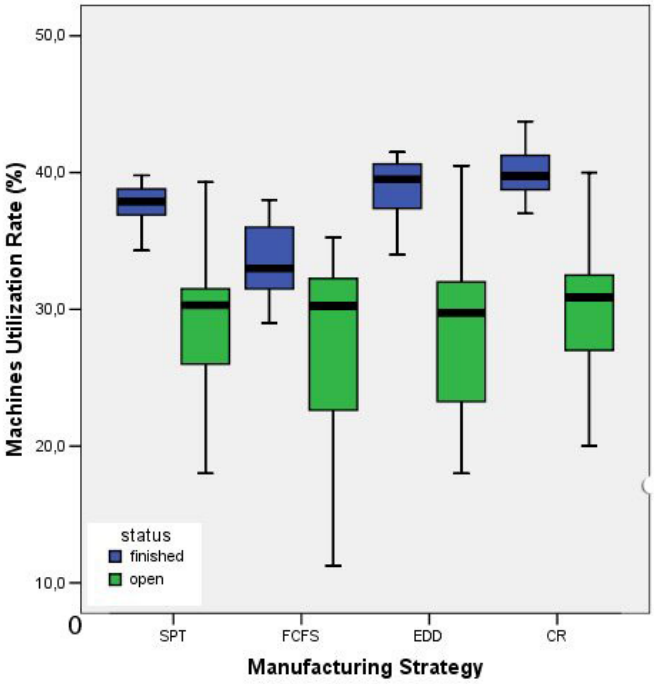


figure 27: Machine Utilization Rate Level for 600 test cases

The test results show that the workflow scheduling strategy choice has a significant impact on the total machine utilization as shown in figure 27. This assumption is valid for finished workloads especially. These workloads are such ones, where all work orders have been finished during the shift time. Like for the number of finished products the CR was the most efficient strategy with 39,88% (see Table 3 for details).

	Machine Utilization Rate (%)							
	SPT		**FCFS**		**EDD**		**CR**	
Status	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
Finished	37,53	0,31	33,36	1,02	38,95	0,47	39,88	0,06
Open	28,56	7,77	25,29	7,05	27,71	5,29	28,79	5,12

Table 3: Comparison of Scheduling Strategy and Machine Utilization Rate

Table 3 shows that the FCFS has the lowest machine utilization with 33,36% (mean). Compared to the FCFS strategy the CR increases the rate by 20%.

5. Results

This section gives an overview of the SAW project results. Therefore the problem statements of this thesis are analysed. The main analysis is focused on the coordination of the agents and the used patterns. Furthermore the proof of concept and its results are discussed.

5.1 Multi Agent System for production planning

The main goal of the design process of this thesis is the development of a coordination component for an agent-based production planning system. Thus the implementation is used to be presented to end-users and is extended by further researchers the whole project is divided into two different scenarios. One is the implementation of the CEBIT application, which has been finished. The other scenario represents the testing tool for the optimization of the workshop scheduling strategies, which will be extended by further researchers. Both scenarios have the same basic application including a MAS, coordination component and test management.

5.1.1 CEBIT Scenario

The CEBIT application was successfully presented at the CEBIT exhibition. The user is able to act as dispatcher in the production system simulator. The simulator got a lot of attention and the visitors seemed to be interested how MAS can support the simulation of manufacturing systems.

During the exhibition some problems occurred which had not been detected before. One problem was the missing of an endless run mode of the simulation during the exhibition. These problems were fixed during the exhibition period.

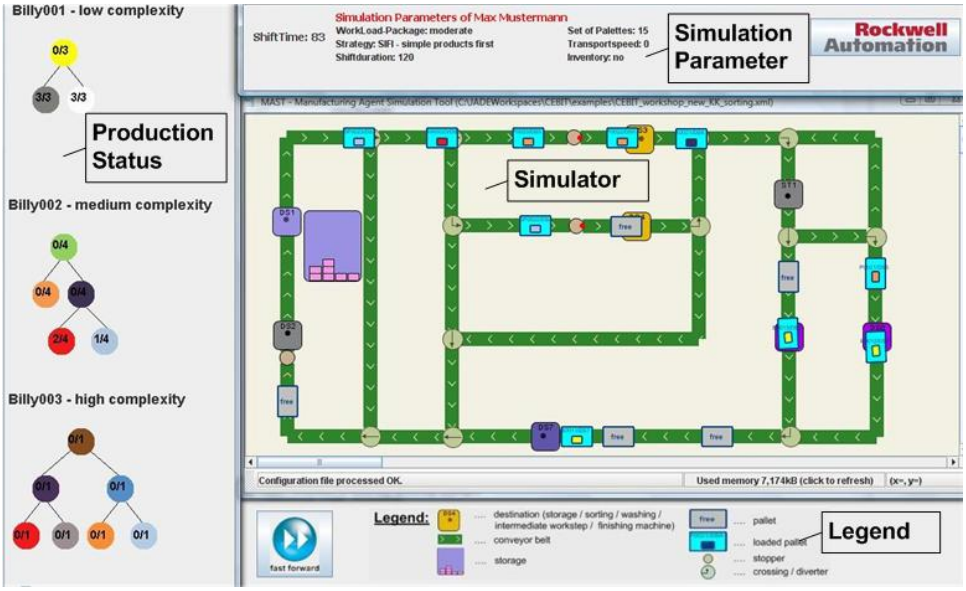


figure 28: CEBIT Application during simulation run

5.1.2 SAW Scenario

The SAW application is an extension of the CEBIT program. It helps to analyse the impact of different workshop scheduling strategies and input parameters on the overall system

performance, i.e. the number of finished products and machine utilization during a certain shift. The first analysis could be done soon after the CEBIT because there were just a few modifications to be implemented.
Currently the application is being extended by other students. The planned extensions are described in section 6.5 “Further steps”.

5.2 Coordination of agents

The coordination of the agent-based production planning system is the main goal of this thesis. The production planning system is based on a simulator developed by Rockwell Automation which already includes a set of simulation agents. This agent system is modified and extended by a coordination component that monitors the agents’ tasks and activities. This coordination component represents the interface between the business layer and the workshop layer. The business layer is managed by a dispatcher and the workshop layer consists of the various simulation agents.
The coordination component is described in a Master/Slaver pattern which is shown in figure 29. The dispatcher (*Order Agent*), located in the first layer, acts as Master and delegates tasks to a certain number of Slaves depending on the number of products in the order. The Slaves are defined as *Product Agents* (PA). These agents in turn act as Masters and use the simulation agents (*Resource Agents* and *Transport Agents*) from the third layer as Slaves to fulfil their tasks.

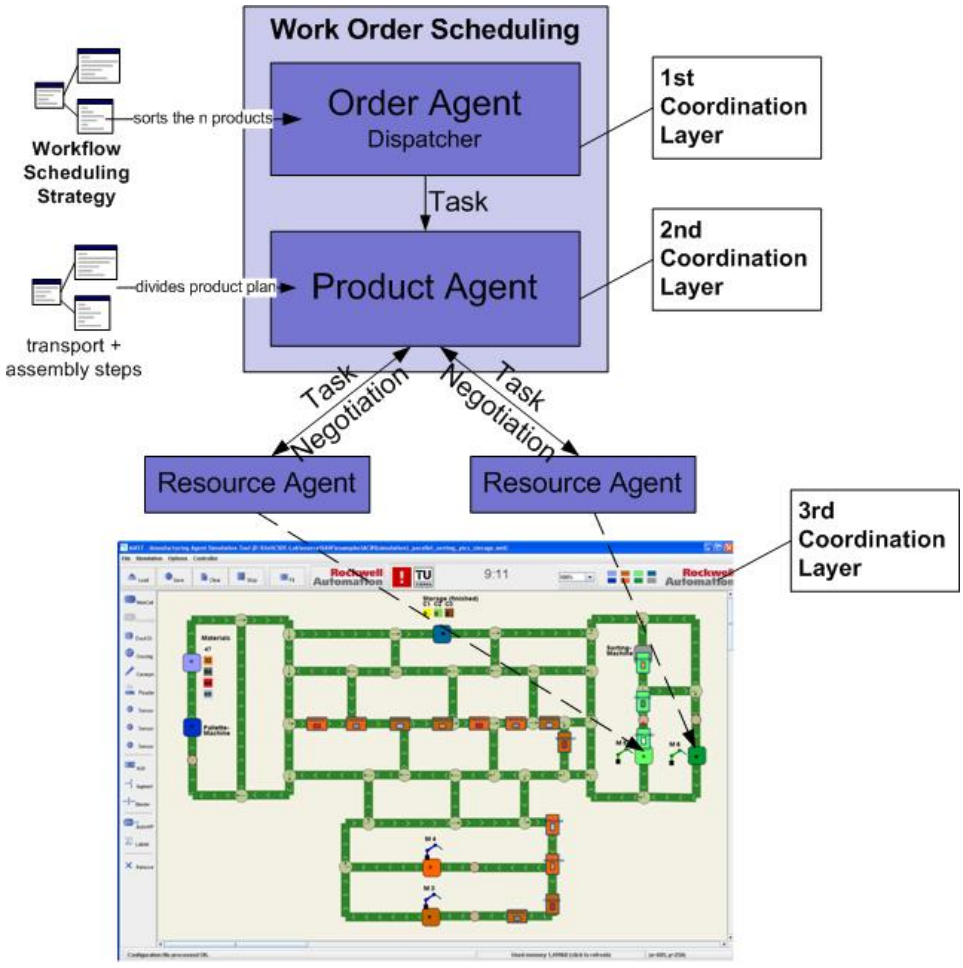


figure 29: Coordination layer model of the project

Figure 29 shows the layer model of the production automation system used in the project. The order agent in the first layer is responsible for the incoming business orders from the costumer, monitoring and guiding a single product through the simulator. The business orders represent guidelines for the arrangement of product sequences depending on the selected workshop scheduling strategy. After the sorting mechanism the dispatcher registers n agents (PA) for each product and forwards the product, as a defined so-called product plan, to these agents.
These n PAs operate in the second layer of the coordination model and represent the interface between the dispatcher and the simulation agents. They analyse the product and divide the product plan in more detailed working steps, i.e. transport and assembly steps. These working steps are delegated to Resource Agents (RAs). This decision making process includes a very important negotiation and allocation sequence between the PA and the RAs. The choice which RA will get the working steps is done using the principal of an auction pattern. The Auction pattern provides the possibility of offering a good or a service to other participators. In this case the PA sends an announcement message, including an identifier and the machine function, to the RAs. The RAs offering the required function send back a message containing the estimated processing time of the machine function plus the estimated time needed for the transportation to the machine. The PA picks the RA with the lowest overall machine function time and delegates the task to it. During the simulation processes the product agents of the second layer can influence the simulator and monitor the events and states within the simulator.
The third layer is the simulator, where the simulation agents communicate with each other as well as with the agent of the layer above. The agents can be classified into Resource Agents, which adopted delegated tasks by the PAs, and Transport Agents, which have to fulfil various transport steps (e.g. sending palettes). The simulation agents implemented by Rockwell Automation already include a self coordinated behaviour. This coordination functions have been modified and adjusted to the needed requirements. The agents have to fulfil the various tasks getting from the agents in the second layer and have to report about the different status, capacities, failures and the measurements.

5.3 Coordination Patterns

The developed coordination pattern shown in figure 29 represents the interface between the business layer and the workshop layer. The business layer is responsible for incoming orders and divides these orders into more detailed working steps. These processes are realised in layer 1 and 2 of the patterns. The agents of the second layer forward the details working steps to the agents on the third layer. These agents are operating in the workshop layer and are represented in the implemented simulator.

Besides the developed coordination pattern for the interface between the business layer and the workshop layer a literature study is done to analyse the existing simulator. This analysis shall give an overview of the used coordination patterns based on the defined by Deugo, Weiss and Kendall [19] for the agent coordination. The research shows that the coordination techniques within the simulator are based on two patterns. One is the Blackboard pattern and the other one is the Master/Slave pattern. The first mentioned

technique is responsible for the message exchange between the agents; the other technique regulates the task distribution between the various agents.

5.3.1 Neglected Coordination Patterns

Besides the coordination patterns used in the application, Deugo, Weiss and Kendall [19] defined three further ones, which may be useful to be integrated into the project:

- Market Marker Pattern
- Negotiating Agents Pattern
- Meeting Pattern

The *Market Marker pattern* simulates a market where a seller tries to sell goods or services to one or more buyers. In the MAST, used for the project, the dispatcher does not "buy" the agent's ability from the highest bidder in the system. As described in the section 4.2.2 the agents delegate tasks to others and have to wait until an agent is free to fulfil the task. The *Market Marker Pattern* is one subject that will be added into the project (see 6.5).

The *Negotiating Agents Pattern* describes the cooperation of a certain number of agents. An Initiator forwards its problem to the other agents. These agents take on the role of Critics, which check if there would be a conflict with their own tasks. If there is a problem an alternative solution has to be found. The MAST tool does just fulfil this ability in a reduced way. Such coordination between the agents can be found within the simulation agents. For example the routing is such a problem solving process. If a conveyor belt fails, the shuttles search for an alternative route to the destination. Therefore the simulation agents, particularly the crossings and conveyor belts, tell each other if they are reachable from the shuttle's position.
Such an alternative solution search is not used during the communication of the coordination agent, Product Agent (PA), and the simulation agents. The PA just forwards the tasks to the agents and does not react if any failures occur or the capacity of all simulation agents is reached already.

The *Meeting Pattern* provides an environment where agents can meet to communicate with each other. The agents within the project do not have the possibility to register to an independent environment to solve coordination problems. The communication between the individuals is public and each agent that wants to receive certain information, can subscribe to it. There is not any Supervisor which controls the subscription of the agents.

5.4 SAW prototype compared to existing solutions

The SAW project is one MAS approach to implement a production planning and modelling simulator. In the literature there are several other solutions for such a complex and distributed problem. Within this thesis there are two approaches compared with the SAW project.

5.4.1 SAW prototype

The SAW prototype implemented during this project shows that MAS is a possible solution to implement production planning systems. The flexibility and the possibility to provide the various agents with artificial intelligence allow configuring agents in different ways as well as simulating a whole manufacturing system. The development during this project is the first step into an agent coordinated system which allows simulating production planning processes and analysing workshop scheduling strategies. The SAW prototype is taken over by other students who extend and optimise the agent system.
As a literature research showed, the SAW approach is similar to existing solutions. Two other solutions for a manufacturing system based on an agent system are described in the next sections.

5.4.2 ProPlanT

The ProPlanT [12] is a multi agent based prototype for production planning. It is implemented to simulate production processes of TV transmitters manufactured by Tesla TV. The system consists of a certain number of agents that are classified into three groups. The first group includes so called order agents (OA). They are responsible for one order and how it can be fulfilled. The other group consists of project management agents (PMA) which are account for "distributing the work to the best possible executive PA agents" [43]. The last group includes the already mentioned PAs (production agents). These agents are responsible to carry out the manufacturing processes at the shop floor level.

This short description of the ProPlanT prototype shows that the coordination framework of the agents is based on the same agent types like in the SAW project. Table 6 shows a comparison of the roles at the ProPlanT and the SAW project.

Agent Role ProPlanT	**Agent Role SAW**	**Description**
Project Planning Agent (PPA)	Order Agent – (OA)	responsible for a whole order (*n* products)
Project Management Agent (PMA)	Product Agent (PA)	monitoring of production processes
Production Agent (PA)	simulation agents	carry out manufacturing processes

Table 4: agent roles of ProPlanT prototype and SAW prototype

To read more about the ProPlanT prototype see Lazanský[43] et al. and Marík et al.[12].

5.4.3 Framework structure of Lim and Zhang

Lim and Zhang [13] define an agent framework structure for an agent-based manufacturing system. The framework is more complex than the one of the SAW project. It consists of four different manager agents, which can have a certain number of subagents. The agents can be classified as execution agents, which carry out procedures and make decisions, and information agents, which provide information. An overview of Lim and Zhang's framework can be seen in figure 30.

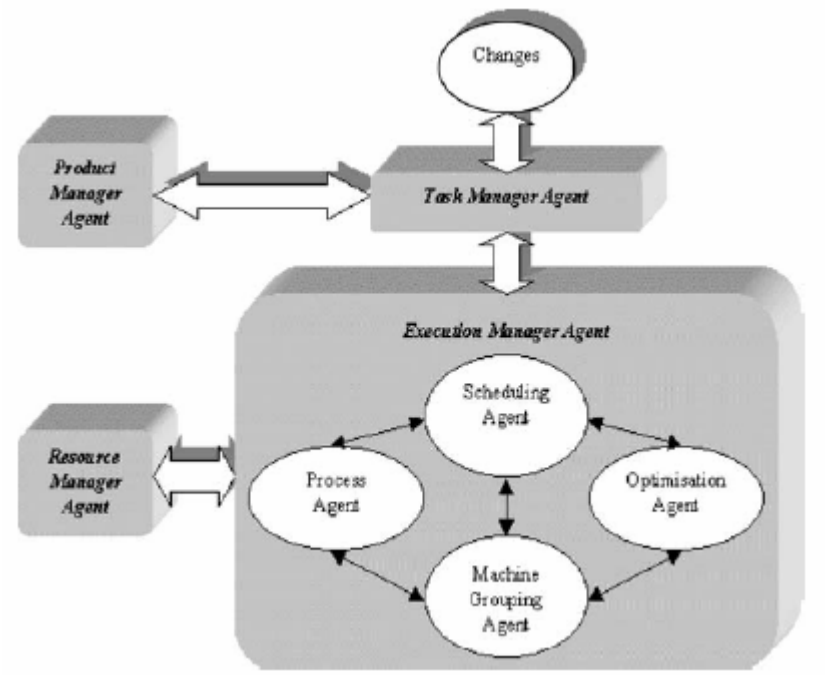


figure 30: Lim and Zhang agent framework overview [13]

The product manager agent (PMA) "represents all the physical products that are currently and previously manufactured in the shop floor" [13]. The products include various parameters, where the product identifier has to be unique.
The resource manager agents (RMA) are physical resources like, machines or conveyor belts. Task manager agents (TMA) are the interface to the user. They represent an interaction channel between the user and the other agents.
The execution manager agent (EMA) is responsible to monitor the agents producing a certain product. This agent has subagents which have different roles and abilities. If changes within a product plan occur or a new product is added to the simulation the EMA sends a message to the process agent (PA). The PA has to check if the manufacturing processes are available. The machine grouping agent is responsible to generate alternative machine groupings based on the process sequences provided by the PA. The scheduling agent (SA) provides a certain number of scheduling options and the optimisation agent (OA) evaluates and optimises the scheduling options.

Compared to the SAW framework this agent coordination is much more complex. But there are also some similarities between the two approaches. Table 5 shows the comparison.

Lim and Zhang framework	**SAW framework**	**Description**
Task Manager Agent (TMA)	not yet implemented	interface to user
Resource Manager Agent (RMA)	simulation agents	Existing resources in the production system
Product Manager Agent (PMA)	not yet implemented	Manage product information and inventory

Execution Manager Agent (EMA)	Product Agent (PA)	scheduler

Table 5: agent roles of Lim and Zhang framework and SAW framework

To read more about this framework see Lim and Zhang [13].

5.4.4 Workflow Scheduling Strategies/ Proof of concept

One aim of the thesis is to test different predefined workflow scheduling strategies on a digital simulator. This is done by a proof of concept where the agents have to fulfil a certain set of tasks under different conditions. This proof of concept shall in one hand demonstrate that the implemented coordination component is able to coordinate a complex and distributed agent-based system. On the other hand it helps to compare the various workshop scheduling strategies with each other.

The analyses show that the workflow scheduling strategies have a very important impact on the production system. The first part of the analysis focuses on the number of finished products in a certain shift, depending on the selected strategy and the number of pallets on the simulation. The second part of the analysis compares the machine utilization during the usage of different strategies. Therefore four strategies are defined: *First Come First Served* (FCFS), *Critical Ratio* (CR), *Earliest Due Date* (EDD) and *Shortest Processing Time* (SPT). For further descriptions to these strategies see 4.5**.**

During the test runs for the calculation of the number of finished products, the CR strategy exposed to be the most efficient one. The tests are done with 5, 10, 15 and 20 pallets on the simulation. Regardless to the number of pallets, the CR always gets the highest rates. As Table 6 shows, the number of finished products usually grows with an increase of the number of pallets. An exception is the EDD strategy, where the value drops (2%) if the number of pallets is increased from 15 to 20. This happens because the EDD does only use a limited number of pallets, in contrast to the other strategies which do always utilize the maximum number of pallets to deliver. These results show that the EDD is better for a manufacturing system with less available pallets.

	Number of finished products							
	SPT		**FCFS**		**EDD**		**CR**	
Pallets	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
5	15,83	1,21	16,40	1,14	14,33	1,33	20,60	1,67
10	26,66	5,95	26,46	5,59	23,22	5,70	30,73	4,31
15	34,77	10,27	34,96	6,53	32,58	7,59	36,22	4,75
20	36,03	7,06	33,95	8,91	34,76	5,92	37,54	3,95

Table 6: test results for number of finished products (mean - % of finished workloads, STD - standard deviation)

The second analysis focuses on the machine utilization. Again, the test cases show that the choice of the workflow scheduling strategy has an important impact on the test results. Like in the calculation for the number of finished products, the CR strategy is the most efficient. As table 5 shows the FCFS has the lowest machine utilization.

	Machine Utilization Rate (%)							
	SPT		FCFS		EDD		CR	
Status	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
Finished	37,53	0,31	33,36	1,02	38,95	0,47	39,88	0,06
Open	28,56	7,77	25,29	7,05	27,71	5,29	28,79	5,12

Table 7: Comparison of Scheduling Strategy and Machine Utilization Rate

6. Discussion and Conclusion

6.1 Effectiveness of MAS for Production Planning

The research for this thesis shows that MAS are an effective solution for modular, decentralized, complex and time varying systems, like production planning. The complex production planning and modelling techniques can be simulated by agents, which negotiate and collaborate with each other to solve given tasks. This coordination within the agent community is similar to the situation within a factory where employees interact with each other or with machines. These employees and machines can be substituted by agents, which have different abilities defined in their behaviours. The information used to produce certain products is divided into various working steps and delegated to the agents. Lazanský et al. [43] define the following advantages of using MAS for production planning:

- A clear separation of collaboration and problem solving knowledge
- Transparent organisation of knowledge and data enabling easy maintenance
- Highly dynamic behaviour with minimal communication traffic achieved through the subscription mechanism

6.2 Limitation using MAS

Besides the advantages and the effectiveness of MAS there are also some disadvantages and problems [32]:

- The agent-based problem solving mechanism is not always the optimal way. This may result in computationally unstable, which means that a useful solution is not found in a given computational time.
- MAS can not be used for physical problems that can not be divided into sub-problems or sub-objectives.
- Agent-based systems consume a lot of resources especially for monitoring and support. Implementing, testing and modifying the system is usually expensive and consumes a lot of time.

Another limitation that occurs during the implementation phase of this thesis is the huge number of messages sent by the agents. For each changed event or state a message is sent and received from several agents. This message overload is a performance problem and can be improved by analyzing the messages and filtering the ones which really affect the system. This work has already been done in the current version of the SAW project.

6.3 Extensibility of MAS

During the implementation phase of this thesis the easy extensibility of MAS was one major advantage for further extension of the application. The agent system was extended by new agents and components without much effort. The new system parts were easily added and integrated. An agent-based application allows working with incremental

software development processes which means that the system can be used without finishing all requirements.
Besides adding new agents, further coordination patterns improve the system performance. One example is the Auction Pattern (or Market Maker Pattern) which provides a dynamic dispatching function to support the load balancing during the simulation run.

6.4 Fault tolerance

The simulation runs showed that the agent system already includes a certain fault tolerance. The coordination component implemented during the thesis provides a monitoring function represented by the dispatcher. If this component fail or the communication between the simulation agents and the coordination agents breaks down, the agents within the simulation still fulfil theirs tasks. The simulation agents can work without the coordination agents for a certain period. This self organized mechanism guarantees that the simulation does not fail without the coordination component.
The fault tolerance is one part of the system which can be optimized by further work.

6.5 Further Steps

After finishing the practical part of the thesis, the project was passed to other students who extend the system. The following section will describe the extensions that already have been realised or will be.

6.5.1 Shop layout

The first step of the extension was a new shop layout. It is more complex than the one in the previous SAW test cases. The figure 31 shows the new layout. Besides the visual changes, the performance was improved by speeding up the simulation.

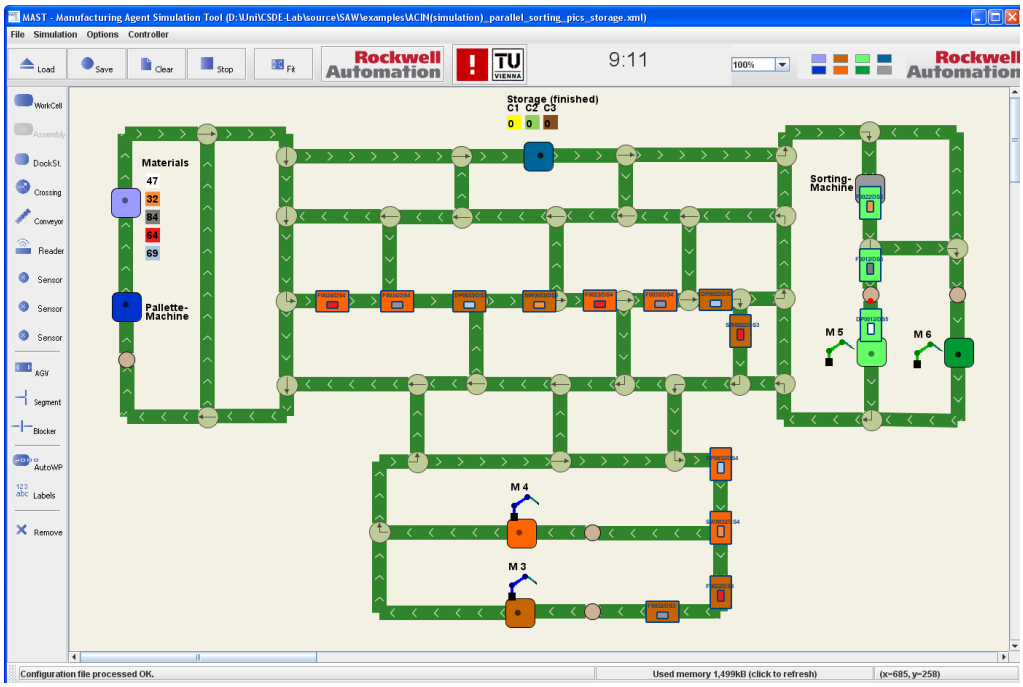


figure 31: new shop layout

6.5.2 Input parameters

The input parameters are extended by adding failure scenarios. As in a "real" manufacturing plant, failures can disturb the production processes. So they also have to be included in the implemented simulation. Such failures help to know how the simulation and especially the agents would react on them.

```
<failures id="1">
   <failure>
      <machinename>B50</machinename>
      <trigger type="time">503.49</trigger>
      <timetoresolve>80.69</timetoresolve>
    </failure>
</failures>
```

listing10: failure entry in the input parameter file

6.5.3 Workflow scheduling strategies

The strategies are extended by adding the expected transport time into the algorithm. This helps to have a better comparison of the strategy's performance.
The following new strategies can be added to the project:

- *Average processing time*: The value is calculated by dividing the sum of all needed operation times through the quantity of the operations.
- *Relative processing time*: The calculation is done by the division of the next operation time through the remaining operations.
- *Number of operations*
- *Short imminent operation:* Operations which have to be done next

Additionally a Java interface allows any user to define workflow scheduling strategies without having knowledge of the implementation of the simulator.

6.5.4 Integration of Naiad

The Naiad [44] correlated event processor provides basic services for event processing and analysing. Therefore a general agent has to be implemented. This agent is subscribed to all message types and agents. It is the interface between the agents regulating the manufacturing process and the Naiad application. As soon as a message is sent, the general agent forwards it to the Naiad component. This improves the message flow by having a central control system. Therefore the messages have to be extended by a timestamp. The main goal of a combination of SAW and Naiad is the possibility to define external analysis with the help of the agent's messages.
For the communication with the Naiad tool a special format for the exchange protocol has to be defined.

6.5.5 Dynamic dispatching

The communication process between the coordination agents and the simulation agents can be improved by implementing the Auction pattern (see Market Maker Pattern 2.4.7). The Auction pattern provides the possibility of offering a good or a service to other participators. Within the application a coordination agent would send a message about all needed machine functions to all simulation agents. The simulation agents that provide the function would answer with the processing time plus the waiting time. Additionally the expected transportation time from the starting point to the machine would be added. To realize such an auction for machine functions the coordination agent has to know about the position of all the transport shuttles, it already has occupied to avoid double bookings.

7. References

[1] R. A. LeMaster, "Automated Production Systems", http://www.utm.edu/departments/engin/lemaster/Auto%20Prod%20Sys/Lecture%2001.pdf , 24.05.2008, foil 23, 2001.

[2] R. H. Erich Gamma, Ralph Johnson, John Vlissides, Design Patterns. Elements of Reusable Object-Oriented Software.: Addison-Wesley Longman, 1995.

[3] L. L. Nwana H. S., Jennings N. R., "Software Agents: An Overview", Knowledge Engineering Review, vol. 11, pp. 205-244, 1996.

[4] P. Maes, "Agents that Reduce Work and Information Overload", Communications of the ACM, pp. 31-40, 1994.

[5] Z. R. Brenner Walter, Wittig Hartmut, Intelligente Softwareagenten: Springer, 1998.

[6] R. J. Durfee Edmund, "Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples", in 13th International Workshop on DAI, 1994, pp. 94 - 104.

[7] E. H. Durfee, "Distributed problem solving and planning ", in Mutli-agents systems and applications Springer, 2001

[8] K. P. Sycara, "Multiagent Systems", in American Association for Artificial Intelligence, 1998, pp. 79-89.

[9] J. Ferber, Multi-Agent Systems: Addison-Wesley, 1999.

[10] L. L. Nwana H. S., Jennings N. R., "Co-ordination in software agent systems", British Telecom Technical Journal, pp. 79-88, 1996.

[11] S. O. Lazanský J., Marik V., Pechoucek M., "Application of the multi-agent approach in production planning and modelling", Engineering Applications of Artificial Intelligence, pp. 369–376, 2001.

[12] V. Marík, Pechoucek, M., Hazdra, T., Stepánkova, O.,, "ProPlanT - multi-agent system for production planning", in XVth European Meeting on Cybernetics and Systems Research, Vienna, 1998, pp. 725-730.

[13] Z. Z. Lim M.K., "A multi-agent based manufacturing control strategy for responsive manufacturing", Journal of Materials Processing Technology, pp. 379-384, 2003.

[14] Nwana H. S., "Negotiation Strategies: An Overview", BT Laboratories internal report 1994.

[15] Jennings, "Coordination Techniques for Distributed Artificial Intelligence", in Foundations of Distributed Artificial Intelligence: John Wiley, 1996, pp. 187-210.

[16] S. M. P. Huhns M., "CKBS-94 Tutorial: Distributed Artificial Intelligence for Information Systems", 1994.

[17] M. J. Bussmann S., "A Negotiation Framework for Co-operating Agents", in Proc CKBS-SIG, 1992, pp. 1-17.

[18] C. Alexander, The Timeless Way of Building: Oxford University Press, 1979.

[19] W. Deugo, Kendall, "Reusable Patterns for Agent Coordination", in Coordination of Internet Agents: Models, Technologies, and Applications: Springer, 2001.

[20] S. G. Eilebrecht Karl, Patterns kompakt, 2 ed.: Elsevier, 2007.

[21] Hayes-Roth, "A Blackboard Architecture for Control", Artificial Intelligence, pp. 251-321, 1985.

[22] L. Aridor, "Agent Design Patterns: Elements of Agent Application Design", IEEE, 1998, pp. 108-115.

[23] P. Wong, et al., "Concordia: An Infrastructure for Collaborating Mobile Agents", in Proceedings of the First International Workshop on Mobile Agents, 1997, pp. 86-97.

[24] Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Transactions on Computers, pp. 357-366, 1980.

[25] C. Steinmetz, et al, "Bid Evaluation and Selection in the MAGNET Automated Contracting System", in Agent Mediated Electronic Commerce Springer, 1999, pp. 105-125.

[26] N. Carriero, and Gelernter, D., How to Write Parallel Programs: A First Course: Massachusetts Institute of Technology, 1990.

[27] N. Griffeth, and Velthuijsen, H., "Reasoning about goals to resolve conflicts", in Proceedings of the International Conference on Intelligent and Cooperating Information Systems, 1993, pp. 197-204.

[28] P. Vrba, "MAST: Manufacturing Agent Simulation Tool", in IEEE Conference, 2003, pp. 282-287.

[29] Telecom-Italia, "Java Agent Development Framework." vol. 2008, 2008.

[30] FIPA, "The Foundation for Intelligent Physical Agents", http://www.fipa.org/, 2008.

[31] B. F. Andreas Drexl, Hans-Otto Günther, Hartmut Stadtler und Horst Tempelmeier, "Konzeptionelle Grundlagen kapazitätsorientierter PPS-Systeme", Zeitschrift für betriebswirtschaftliche Forschung 46, pp. 1022–1045, 1994.

[32] C. S. Caridi Maria, "Multi-agent systems in production planning and control: an overview", in Production Planning & Control: Taylor & Francis, 2004, pp. 107-118.

[33] S. M. Mönch L., Zimmermann J., Habenicht I, "The FABMAS multi-agent-system prototype for production control of water fabs: design, implementation and performance assessment", in Production Planning & Control: Taylor & Friends, 2006, pp. 701-716.

[34] K. Kemppainen, "Priority Scheduling Revisited - Dominant Rules, Open Protocols and Integrated Order Management ": HSE Print, 2005.

[35] E. Kühn, Virtual Shared Memory for Distributed Architecture: Nova Science Publishers, 2001.

[36] R. M. M. Mor, J. Riemer, "Using Space-Based Computing for More Efficient Group Coordination and Monitoring in an Event-Based Work Management System", in ARES Wien, 2006, pp. 1116-1123.

[37] W. D. Moser T., Biffl S., Merdan M., Vrba P., "Simulation of Workflow Scheduling Strategies", in ICARCV, 2008, pp. 1-6.

[38] W. Royce, "Managing the Development in Large Software Systems", in Proceedings of the 9th international conference on Software Engineering, 1987, pp. 328-338.

[39] A.-P. D. Bröhl, W., Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden: Oldenburg, 1993.

[40] G. Müller-Ettrich, Objektorientierte Prozeßmodelle: UML einsehen mit OTTC, V-Modell, Objectory. Bonn: Addison-Wesley, 1999.

[41] M. c. S. Tanenbaum Andrew, Distributed Systems: Prentice Hall International, 2003.

[42] FIPA, "FIPA ACL Message Structure Specification", www.fipa.org/specs/fipa00061, 2002.

[43] S. O. Lazanský J., Marik V., Pechoucek M., "Application of the multi-agent approach in production planning and modelling", Engineering Applications of Artificial Intelligence, pp. 369–376, 2001.

[44] R. V. Szabolcs Rozsnyai, Josef Schiefer, Alexander Schatten, "Event Cloud - Searching for Correlated Business Events", in Proceedings of the 4th IEEE Conference on Enterprise Computing, E-Commerce and E-Services (EEE ' 07): IEEE, 2007, pp. 409-420.

Appendix

A.1. inSensor() – Docking Station

```
1. Input:   wpS : WorkPieces  //represents the incoming shuttle
2.
3. Variable:      this : DockingStationS  //represents the current
docking 4. station
5.
6. //BEGIN
7. IF (wpS.getDestination() != this.getName())
8.    à send Shuttle through
9. ELSE
10.   // Order and Sequence in work
11.   setOrderInWork(((Shuttle)wpS).getOrderID());
12.   setSequenceInWork(((Shuttle)wpS).getSequenceID());
13.
14.   //checks if shuttle transports a product
15.   IF (((Shuttle)wpS).getProductIDonShuttle() == “”)
16.         //if current docking station is the inventory
17.         IF (wpS.getDestination() != inventoryID)
18.               //get the storageList (List of products waiting to get
19.               out of the inventor)
20.               ArrayList<ProductPlan> productList
21.                                    this.getStorageList();
22.               IF (productList.size() > 0)
23.                     à take first product from the list, put it on
the 24.                        shuttle and send it to its destination
25.         ELSE
26.               à let free shuttle pass
27.
28.   ELSE  //Shuttle with a product arrived
29.         //get productID, take fourth part of getProductOnShuttle()
30.         //getProductOnShuttle() = OrderID/SequenceID/Number/ProductID
31.         String p1 = wpS.getProductOnShuttle();
32.         à set go through time depending on the machinefunction
33.         this.removeWorkPiece();
34.         à set Shuttle and docking station as free
35.
36.//END PROCEDURE
```

A.2. workPieceWait() – SortingMachine

```
1. Input: productID : string  //ID of the product incoming product is
2.                            dependent
3. Variable:      machineList : ArrayList<MachineListElement>
4.                productsToMachine : ArrayList<String>    //list of
5.                            product went to a curtain assembly machine
6. //BEGIN
7. i < 0
8. j < 0
9.
10. machineList = this.getAgent().getMachineList();
11.
12. //REPEAT (i) //check every assembly machine belonging to sorting
13.                machine
14.   //REPEAT (j)  //check all products went to a curtain assembly
machine
15.         IF (productsToMachine[j] == productID)
16.               return false;
17.         END IF
18.   j = j+1;
19.   UNTIL j == productsToMachine.size()
20. i = i+1;
21. UNTIL i == machineList.size()
22. return true;
23. //END PROCEDURE
```

A.3. checkProductOrder() – Sorting Machine

```
1. Input:   wpS : WorkPieceS  //shuttle arrived at machine
2.
3. Variable:      checker : Boolean //needed to know if a conditions is
4.                                  kept
5.          productID : String //ID of the incoming product
6.          machineList : ArrayList<MachineListElement>
7.          siblingList : ArrayList<String> //list of the sibling
products
8.                                          of the incoming
9.
10. //BEGIN
11. i < 0
12. j < 0
13. k < 0
14.
15. checker = false;
16. machineList = this.getAgent().getMachineList();
17. siblingList = wpS.getProductPlan().getSiblingList();
18.
19. //check if a sibling of incoming product waits at an assembly machine
20. //REPEAT (j) //go through all sibling products
21.   //REPEAT (i)  //go through all assembly machine belonging to
sorting 22.                  machine
23.         //REPEAT (k)  //go through all products want to the machine
24.               //check if sibling product is already sent to a machine
25.               IF (machineList[i].getArrivalProductList[k] ==
26.                           siblingList[j])
27.               //send shuttle to machine where sibling waits and
remove
28.                     this.sendWorkPiece(machineList[i].getID());
29.                     checker = true;
30.                     //add product to list of product sent to machine
+ 31.                  remove product from the list of expected products
32.               END IF
33.
34.         k = k+1;
35.         UNTIL k == machineList[i].getArrivalProductList().size()
36.   i = i+1;
37.   UNTIL i == machineList.size()
38. j = j+1;
39. UNTIL j == siblingList.size()
40.
41. //check if an assembly machine is free
42. IF (checker == false)
43.   //REPEAT (i) //go through all assembly machine belonging to sorting
44.               machine
45.         //checks if list of products expected at the machine is
46.         empty
47.         IF (machineList[i].getArrivalProducts().size() == 0)
48.               //send shuttle to free machine + add product to list of
49.               expected products
50.               checker = true;
51.         END IF
52.   i = i+1;
53.   UNTIL i == machineList.size()
54. END IF
55. IF (checker == false)
56.   //send shuttle to waiting round
57. //END PROCEDURE
```

A.4. checkHierarchy() – Product Agent

```
1. Input:   ds : DockingStationS    //machine where shuttle arrived
2.          productid : String      //ID of the arriving product
3.
4. Variable:      productHierachy : ArrayList<ProductPlan> //product plan
5.          parent : ProductPlan //parent product of the arriving one
6.          subProducts : ArrayList<ProductPlan> //subproducts of the
7.                              parent == sibling products of arriving one
8.          finish : boolean //flag to check certain conditions
9.
10. //BEGIN
11. i < 0
12. j < 0
13.
14. //iterate through the product plan
15. //REPEAT (i)
16.   IF (productHierachy[i] == productid)
17.         parent = productHierachy[i].getSuccessorProduct();
18.
19.         IF (parent != null)
20.               subProducts = parent.getConsistsof();
21.               finish = true;
22.
23.               //REPEAT (j)
24.                     // check if all child products already arrived
25.                     IF (subProducts[j].getFinished == false)
26.                           finish = false;
27.                     END IF
28.               UNTIL j == subProducts.size()
29.
30.               //check if all childproducts of parent are arrived AND
if 31. parent product did not arrived at destination AND is not on the
way to 32. destination
33.               IF (finish == false &&
34.                     parent.getFinished() == false &&
35.                     parent.OnTheWay() == false)
36.                     //logging stuff
37.                     //send product (parent) with arriving shuttle to
38.                     destination
39.                     getSendPalettWithProduct(parent, ds);
40.               ELSE
41.                     //send arriving shuttle to inventory
42.         ELSE
43.               //send arriving shuttle to inventory
44.   END IF
45.
46. UNTIL i == productHierachy.size()
47.
48. //logging stuff
49. //END PROCEDURE
```