IFT

A Multiagent Design Methodology for the Manufacturing Execution System Domain

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Dipl.-Ing. Solmaz Mansour-Duschet
Registration Number 0526321

to the Faculty of Mechanical and Industrial Engineering

at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Burkhard Kittl

The dissertation has been reviewed by:

Ao.Univ.Prof.Dr. Wolfgang Kastner	Univ.Prof.Dr. Alfred Taudes

Vienna, 27th October, 2020

Solmaz Mansour-Duschet

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Solmaz Mansour-Duschet
Nußberggasse 42, 1190 Vienna

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.

Wien, 27. Oktober 2020

Solmaz Mansour-Duschet

Acknowledgements

First, I would like to thank Prof. Burkhard Kittl, who guided me and taught me to question everything with courage and an open mind. He is a good friend and my most valuable critic.
I am grateful to Prof. Kastner for sharing his valuable time and experience with me.
To Shahin Mahmody, I want to express my gratitude for turning my ideas and visions into executable code. I learned a lot from him and enjoyed working with him.

I am grateful for the support and friendship of my friends Amina Wieser, Sara Mansour Fallah, Claudia Hesse and Christina Schön.

My dear father Majid Mansour Fallah, taught me that books can be remarkable friends. He introduced me to technical science, and never hesitated to invest in me.
My beloved mother Sima Naghshdoust, left the familiar for the uncertain abroad only to offer her daughter a brighter future. She is my role model of a confident and independent woman in so many ways.
I am deeply grateful for having you both as parents.

I am grateful for the unconditional love and support my sisters, Sara and Sarina Mansour Fallah gave me.

I also want to thank my beloved husband Simon Duschet, who helped me overcome any doubt. May our future be as bright and joyful as our past was.

Abstract

While technological inventions and progress are driving the linkage of cyber-physical-production-systems (CPPS), there is still room for a suitable control structure. Multiagent system (MAS) are proper candidates since they establish optimization through competition and flexibility. Although MAS in manufacturing are not new, they still lack momentum.

One reason is the lack of specified design methodologies meeting the demands of the manufacturing domain. The manufacturing domain is mainly driven by mechanical or industrial engineers as stakeholders, with no prior MAS knowledge. Although numerous MAS design methodologies exist, none address mechanical or industrial engineers as target designers.

Among others, this doctoral thesis addresses this issue, as it is built on two pillars. The first pillar of this doctoral thesis is the design of a MAS as a process control for the integrated tool life cycle of the TU Wien Pilotfabrik. The TU Wien Pilotfabrik is a demonstrator factory offering students and scientists a state-of-the-art production environment. The tool cycle combines the tool disposition, the tool supply, and the tool use. The tool cycle was chosen as demonstration process since it manages physical and digital entities and encompasses different production layers. Its biggest challenge is the highly heterogeneous hard- and software landscape, causing missing data exchanges and a lack of interfaces. Although it can be assumed, that the tools have a high impact on the overall production cost and thus a high saving potential, the true tooling cost can only be estimated. According to Sharit and Elhence cutting tools cause 25% of all operation costs [SE89]. As groundwork for this thesis, all linkage issues of the tool cycle in the TU Wien Pilotfabrik were resolved and published as the “integrated tool lifecycle” [FTP19]. Besides providing a process control, the MAS maps the true tooling cost for each order and aims to minimize it if possible, by negotiating agents finding a beneficial order sequence.

During the first attempts of designing such a MAS, the lack of feasible design methodologies for the manufacturing domain was discovered. The second pillar of this doctoral thesis addresses this issue by developing a multiagent **Design Methodology for the Manufacturing Execution System Domain (DM-MESD)** especially addresses mechanical and industrial engineers. The DM-MESD offers the designer the possibility to evaluate if the design methodology is suitable for the desired application in advance. This will be achieved by a defined investigation area, which serves as gauge for other applications. Comparing the application with the investigation area resolves, if the DM-MESD can be used for a specific application. The presented design methodology consists of six phases. The DM-MESD focuses on building services based on existing functions and tasks. It uses textual and structural analysis methods to elaborate those tasks and functions to describe existing services and elaborate new services if possible. It is a pure design methodology with no implementation tools or suggestions. This thesis examines the DM-MESD on the integrated tool cycle. By doing so the desired MAS is being designed and the use of the DM-MESD can be exemplified. The feasibility of DM-MESD and its resulting MAS design will be evaluated by a proof of concept.

Preface

Since the preface stands at the beginning of a complex discussion, it shall ease the reader's first step into a perhaps new topic. The reader might find aid in understanding some of the perspectives presented here by knowing the author's background. Solmaz Mansour-Duschet (born Mansour Fallah) studied industrial engineering at the TU Wien. While studying, she was employed as an application and sales engineer at MMF Engineering, a distributor for industrial equipment. Before, she worked as a Launch-Team member at the General Motors powertrain plant in Vienna. During her time as a PhD student and as employee of the Institute of Production Engineering and Photonics Technologies of the TU WIEN, she engaged with a broad range of topics such as, Manufacturing Execution System (MES), Service Oriented Architecture (SOA) and Multiagent Systems (MAS). In this context the Solmaz Mansour-Duschet acquired humble skills in programming (Java), Web Services, OPC-UA, ontologies and of course SOA and MAS. The reader might notice during this thesis, that the professional strength of the author originates from her knowledge as an industrial engineer, and not as a computer scientist. With respect to this, she pursues a solid scientific work about MAS in the manufacturing domain, from the perspective of an industrial engineer, instead of aiming to provide a wider, but inaccurate perspective as a beginner in programming.

As humans we always risk encountering unfamiliar topics, which cannot be fully mastered in one's lifetime, but it is the unknown that excites the curious the most. Future challenges will call for the cooperation of professionals from different fields. Following this spirit, the author does not try to address the multi-agent paradigm in the manner or quality of a computer scientist instead; she aims to widen the horizon of industrial and mechanical engineers, such as herself, for the multi-agent paradigm.

Contents

CHAPTER 1

Introduction

Competition has a history of leading individuals and firms to their optimum. This is also valid with respect to global competition forcing the domestic producing industry to refine in any possible manner. In 2017, Austria achieved a GDP of 329,18 billion Euros, of which 62,20 billion Euros have been realized directly by the mining and goods producing industry, which makes 18,9% of the overall GDP [Sta17, p. 26]. In 2016, Germany realized a GDP of 2.822,2 billion Euros of which the goods producing sector has contributed 722,3 billion Euros, 25,6% of the overall GDP [DD17, p. 330].

Since 1990 a trend of companies relocating their productions to other countries has become apparent [Kin12, p. 703]. This relocation is not mainly pursued for shortening transportation distances to a fast-growing market, or for operating direct investment (foreign direct investment). Economist Steffen Kinkel explained in his work, that the dominating single motive for relocating the production abroad is still the reduction of labor cost [Kin12, p. 705]. Konings et al. have depicted the substitution of originated jobs by foreigner jobs for multinational enterprises in the European Union (EU) and Central and Eastern European Countries (CEEC). The relocation trend was mainly visible in the manufacturing sector, but against common belief it took place between European countries [KM06, p. 281].

In order to inhibit the relocation trend, reducing the labor cost can be addressed by a higher level of automation. However, a higher level of automation is most profitable in mass production, and not yet suitable for serial or small-sized productions. Besides major investment, a higher-level automation always engenders long ramp-up times, a high sensibility to changes and unsuitable to meet the increasing product variety. The increasing product variety is not new in the manufacturing industry (see [MSF96] and [FRU99]). It is driven by new material or technological benefits or legal regulation, but the major part is caused by customer demands for specifications [ElM09, p. 43]. While a higher level of automation could prevent the relocation of domestic production, it compromises production flexibility and is for now difficult to realize for production with high product variety. Therefore, in 2011 the German Federal Ministry of Education and Research (BMBF) founded the research and development initiative “Industrie 4.0“(I4.0), globally known as Smart Factory or Smart Manufacturing [Gro13, p. 67].

This pre-announced fourth industrial revolution shall equip the domestic industry and its competitiveness for the advent of the Internet of Things (IoT) in the sector. The Internet of Things will enable more flexibility by raising the amount of cross-linked entities through the Internet, while achieving a higher level of automation. IoT owes its genesis to multiple factors,

such as, decreasing prices for central processing units (CPU) and memories, which made Internet connected devices increasingly affordable. At a point, the Internet became more and more the Internet of things, than of humans. Industrie 4.0's strategy is to build upon IoT to enforce the linkage of objects, knowledge, processes and humans, in order to optimize the manufacturing to the extent that a lot size of one becomes profitable [Gro13, p. 23].

The disadvantages of a higher level of automation shall be overcome by production entities capable of being reassembled in a production network as currently needed. Those production entities could be interlinked in a network (IoT, or Ethernet), either by transmitting their information or offering their service to others. In this manner a restructuring of those entities should result in a restructuring of the production itself.
To realize this vision in the manufacturing domain, two requirements must be fulfilled: Melding of the physical with the digital entity, and a flexible control structure to enable the restructuring of entities. The first requirement ensures, that the digital twin reflects the state of the physical entity in all stages. I4.0 utilizes Cyber-Physical-Systems (CPS) to fuse the physical entity with its digital representation [Gro13, p. 5]. CPS are embedded systems integrating computation to physical entities and processes [Lee08]. In their later work Lee, Jay and Bagheri defined CPS as a "technology capable of transforming and managing between the physical assets and the computational capabilities of a connected system" [LBK15, p. 18]. CPS in the production domain called Cyber Physical Production System (CPPS) have already been acknowledged as a core technology of I4.0. CPPS are expected to be the digital twin of their physical asset, and to integrate features, such as robustness, self-organization, self-maintenance, remote diagnosis etc. [Mon14, p. 11].
The second requirement is a control structure enabling flexibility. A control structure is a framework defining form, artifacts, content of the communication. A broadly used control structure in the manufacturing domain is the hierarchical control structure. The Figure 1.1 illustrates the rigid hierarchical control structure. The lowest layer in this figure contains all value adding actions and the therefore required CPPS. The highest layer reflects the global decision-making unit, the Enterprise Production System (ERP) layer. The layer beneath, the Manufacturing Execution System (MES) layer controls the short-term production scheduling and its process control. To exemplify the adverse effect of such hierarchical control structures, figure 1.1 illustrates the report and command flow triggered by an occurring event. Since the CPPS can not decide for themselves, they report and obey orders received from their assigned decision unit of a higher layer (Cluster A, B, C). Due to its rigid command line, such a communication structure is complex to maintain and reacts inflexible to unforeseeable circumstances. However, hierarchical production control realizes an overall optimum, which can hardly be outperformed, when working as planned. Such a production control reacts highly sensitive to unforeseen circumstances, such as machine downtime, or disturbances in the supply chain. An alternative to the overall scheduled optimum is be a local optimized schedule limited to specific CPPSs and managed by them. By taking the decision-making units down, a flatter and therefore more flexible control structure can be achieved. Self-containing CPPSs could cooperate in a flexible manner. This possibility raises the question: Which control structure should frame the cooperation between those CPPS?

Using multiagent system (MAS) seems to a suitable solution for managing a production in a flexible manner. Thereby the services of one or more CPPS would be offered and managed by agents. Such agents react to environmental changes, act, negotiate, cooperate, and compete. Such a control structure reduces the complexity of a system by its natural encapsulation mechanism, as each (CPPS's) service group will be represented by a specific agent type. Figure 1.2 illustrates as a scheme how the before introduced example would change by introducing a multiagent

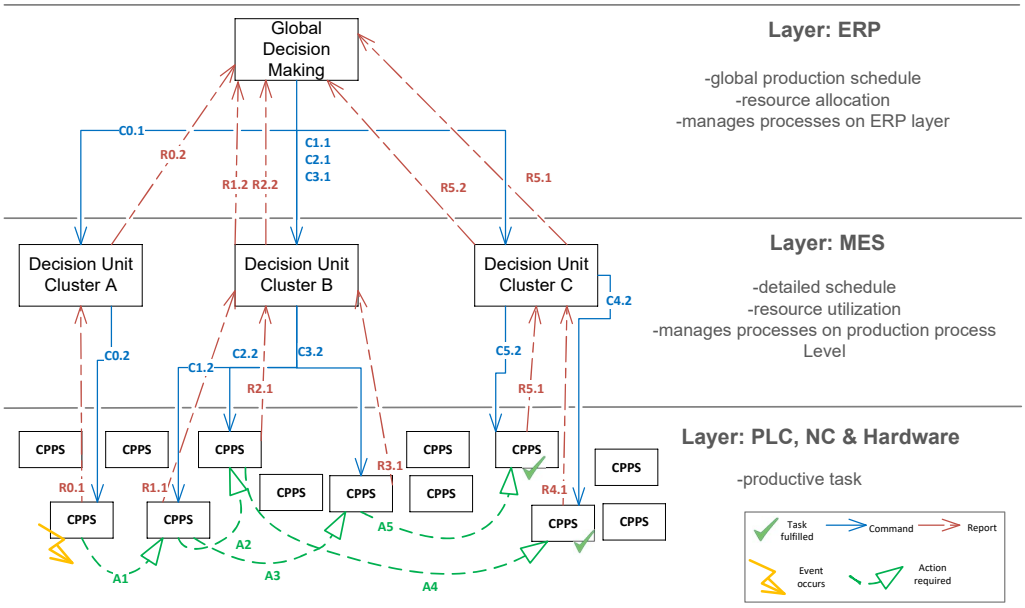

Figure 1.1: Scheme of a possible message traffic in a hierarchical control structure

system on the MES layer. Each agent would be capable of cooperating with other agents and decide according to its own goals. The true advantage will be the direct interaction between the agents, which substitutes the hierarchical command from the higher layer.

Advantages such as reducing the complexity of a system, or encapsulation cannot be achieved, if each and every CPPS is represented by an interface agent. Implementing an interface agent on top of each CPPS does not reduce the complexity of the system, nor does it contribute to its modularity or flexibility. An interface agent offers only a digital one-to-one representative, without any higher service. Interface agents would take the decision unit down to a layer, which might require resolving real-time and safety issues. Instead of interface agents, agents can be modeled to represent higher services of clustered CPPS to employ MES functions, such as controlling or scheduling.

The MAS paradigm does not enjoy a wide popularity in the industry. Since investments are high, and ensuring the throughput is crucial, the idea of letting software entities decide autonomously, based on their Belief, Desire, and Intention (BDI) is not popular. Beside others this is one reason, why MAS, despite their benefits, are not well accepted in the manufacturing domain. According Leitão et al. the scientific projects SOCRADES, IMC-AESOP and ARUM suggest, that the combination of CPS with a collaborative automation paradigm, such as Service Oriented Architecture (SOA) and MAS will resolve some of those issues [LCK16]. The authors propose a strategy to use MAS on the higher enterprise layer for the orchestration and managing task, while SOA shall address the interoperability issues of MAS. SOA is a process driven communication paradigm, which not technology driven. Its service-offering is not bound to a specific protocol or technology, and could in its simplest form be accomplished by the exchange of text (see [JS05] and [BNP+08]). Leitão et al. further point out that the combination of CPS with other technologies, such as MASs, SOAs, IoT, and cloud computing, is the sought innovation. Table 1.1 compares the expected CPPS characteristic with SOA and MAS capabilities to support those innovations. It summarizes both MAS's weakness, such as real-time control and predictability,

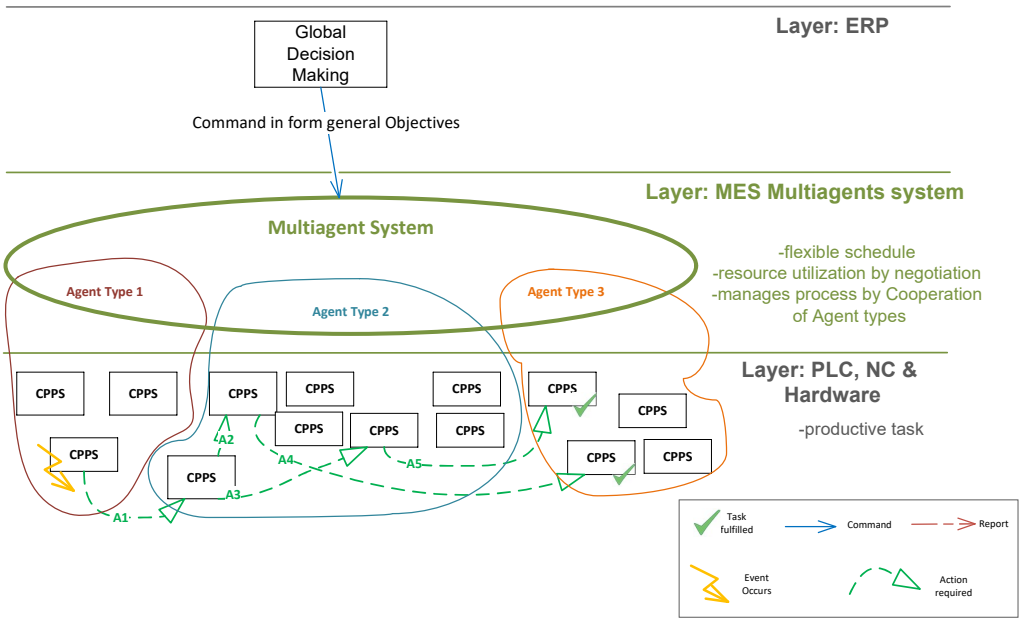

Figure 1.2: Previous scheme with a multiagent system on the MES layer managing and scheduling the production entities

which can be partially repeal by a combination with CPPS and SOA. MAS in manufacturing struggle with interoperability issues since this domain has a highly heterogeneous hard- and software landscape. In this respect, SOA can support MAS by enabling a vertical and horizontal integration of MAS through OPC-UA or Webservices (WS). An additional benefit of MAS is the availability of an interaction standard by the Foundation for Intelligent Physical Agents (FIPA). The FIPA specification defines the communication by the Agent Communication Language (ACL) and eases implementation efforts by offering a set of communication protocols as guidance. Agent Integrated Development Environment (IDE), such as JADE have implemented the FIPA specification. However, the triumphant procession of MAS in manufacturing has not come yet, although those technologies and specifications exist.

Leitão et al. illustrated the delayed MAS advent in manufacturing by assigning technology readiness level (TRL) to the MAS applications [LKR+16]. The authors provided a list of MAS projects enrolled from 1995 until 2015 sorted by ISA 95 levels. Figure 1.3 on the left side visualizes this list. The remastered data in this figure 1.3 reveals that MAS applications were mostly used on the second ISA 95 level "supervisory control" with 16 projects out of 33 projects. The right side of figure 1.3 lists the scopes of those sixteen projects, of which the majority dealt with manufacturing or process control. Especially MAS with self-learning and self-optimization capabilities benefit as a manufacturing control. MAS are also more likely to be utilized as manufacturing control due to their location on a higher layer, where their advantages, such as self-optimization and self-organization are not blurred by their lack of real-time capability. MAS applications with the highest maturity scored a TRL of nine, such as the Daimler Chrysler Project of Bussmann and Schild [BS01]. While most MAS Projects encountered a TRL of six or seven, only projects limited to research laboratories reached a technology readiness level between seven and nine. Although each application of MAS contributes to their familiarity, explicit efforts must be made to ease the MAS application in manufacturing.

Expected CPPS Characteristics [Mon14]	SOA in relation [RBM08] [VMS+14]	MAS in relation [LKR+16] [VMS+14]
Self-Organization	**Very Supportive** SOA based technologies enable offering of devices, services or other entities	**Very Supportive** known for autonomic behavior, can provide self-optimization and self-adaption
Self-Repair	**Supportive** offer the infrastructure and integration for MAS	**Very Supportive** self-healing and self-configuration capability inspired by biological system and swarms
Efficiency	**Very Supportive** enabling a network of services	**Very Supportive** due to the interaction among the individual agents an their self-optimization capability
Transparency "ability to unravel and quantify uncertainties to determine estimation of its capability and readiness" [LL13]	**Very Supportive**	**Very Supportive**
Safety	**Supportive**	**Weak**
Remote Diagnosis	**Very Supportive**	**Supportive**
Real-time Control	**Very Supportive** due to OPC-UA over Time-Sensitive Networking (TSN)	**Weak** due to unsuitable high-speed interactions between individual agents
Predictability	**Very Supportive**	**Weak**

Table 1.1: Comparison of CPPS with SOA and MAS Characteristics

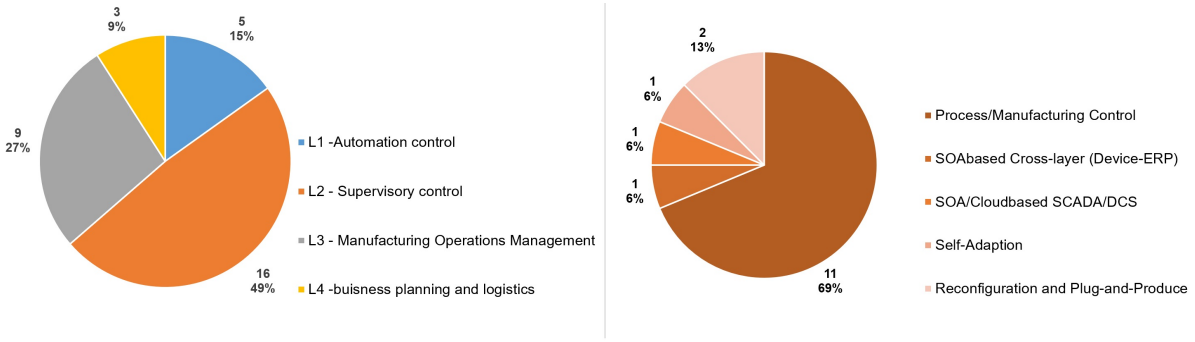

Figure 1.3: **Left**: Realized MAS projects sorted according ISA 95 levels;**Right**: Realized MAS projects at the level 2 of ISA 95 sorted by their Project Scope, Original Data from [LKR+16] analyzed and remastered for this illustration

Vrba et al. suggested that four MAS challenges must be overcome to enable a broad application of MAS outside large R&D departments and laboratories [VMS+14]. The authors conceptualized their work for energy systems. However, the four MAS challenges are also valid in manufacturing:

Resolving the emergent behavior: Designed systems can react in an unexpected manner. This emergent behavior might be inefficient or not, however, it appears alarming for the conservative manufacturing sector.

Solving interoperability issues: A successful interoperability on the syntactic and on the semantic level is required. Syntactic integration can be realized by the common agent communication languages (ACL) of the FIPA specification. A proper ontology for the application domain is necessary. For a successful vertical integration, the combination of MAS with SOA concepts and existing implementation such as Web-Services (WS), OPC-UA, RESTful, ESB will be essential.

Execution on field devices: The use of Java-based agents on field devices still holds deployment challenges.

Lack of proper design methodologies and architectures: MAS design methodologies lack awareness in the manufacturing domain. Common design methodology addresses computer scientists or designers with MAS knowledge. Those methodologies rather focus on the decision-making algorithms or role assigning mechanisms, than realizing an agent architecture that considers industrial restrictions.

1.1 Motivation

Considering the advantages of MAS as a control structure and the available technologies and standards, one question must be raised: Why are MAS still missing approval in the manufacturing domain? The MAS paradigm might not be familiar enough to the manufacturing community. The MAS has its historical source in computer sciences, while the manufacturing domain is driven by industrial and mechanical engineers as stakeholders. Another reason may be the high consulting cost for external professions, especially if the benefit is not certain. Even if a MAS would be feasible for manufacturing process, the stakeholder might not be aware of it, nor can he or she assess the feasibility in advance. Even if the integration of MAS gains acceptance in the producing industry, tools to evaluate the suitability of the MAS paradigm for users with less MAS knowledge do not exist. Vrba et al. pointed out, that the lack of design methods for specific areas might be one reason for the lack of acceptance [VMS+14].

A design methodology is a systematic path towards a solution design offering its user proper methods, concepts, and models along its stages. Design methodologies most commonly consist of at least four stages:

1. A broad description of its application area
2. An analysis phase, mapping the problem area
3. A designing phase, transforming methods or guidelines
4. A development phase

Design methodologies for multiagent systems exist. Some of them have their genesis right at the beginning of MAS or even before, at the beginning of object-oriented programming. All those methodologies source from computer science and require prior MAS- knowledge. The number of design methodologies addressing the manufacturing domain as an application field is small. Among these, the designer is expected to be familiar with common MAS concepts. Beside the expected MAS-Knowledge, modeling aspects and languages common in computer science are also required. These requirements towards the designer restrict the usage of those methodologies in the manufacturing domain. Production processes are complex. Transforming their process control into a MAS process control requires the knowledge of the process stakeholder. A multiagent design methodology especially for industrial and mechanical engineers could address this issue.

Requirements of MAS Design Methodology for the Manufacturing Execution System Domain (DM-MESD)

A common requirement of design methodologies focusses on the model fidelity, which describes the suitability of the modeling concept. The prescriptiveness, the second common requirement, evaluates the user-friendliness, the design methods, and the extent of tools. Those tools can be modeling languages with their own ontology, modeling concepts and notations. Such tools can also be more evolved, such as the CASE (Computer Aided Software engineering) tool. The prescriptiveness is given, if the offered methods guide the designer step by step through the process. This requirement is bound more than the previous one onto the knowledge and capability of the designer. A person with a profound MAS-knowledge might compensate missing descriptions, while a designer with no knowledge depends on the continuous assistance through the designing process. The target designers addressed here are industrial and mechanical engineers with rudimentary knowledge about UML and no knowledge about MAS. This definition raises the demands towards our design methodology.

Besides common requirements the desired design methodology must also meet the specific requirements of the shop floor. Such specific requirements derive from two aspects of the manufacturing domain, the process stakeholders as target designers and the focus on the integrity of the manufacturing processes. Defining mechanical and industrial engineers as target designers limits the modeling tools to those commonly known. The design methodology must also conduct the MAS concept and deploy it intuitively, as the designer has no prior MAS-knowledge. The missing of this prior MAS-knowledge demands exact instructions and guidance throughout the design process. Focusing during the design process on the integrity of the manufacturing processes requires the embedded mechanisms to account for manufacturing restrictions. Such manufacturing restrictions could be task sequences, thresholds, releases, etc. Following domain specific requirements must be considered for designing a multiagent system on the MES layer:

Highly process controlled

While the object-oriented perspective focuses on objects, attributes and methods, the production control requires a process-oriented perspective on the system. A production control focuses rather on process stages than on objects and roles. A process mapped on a MES layer is a multidimensional activity diagram, where the sequence of activities and its constraints are more important, than the entity responsible for it. Those dimensions in this activity diagram arise from the several possible combinations by which the process could be realized. These combinations vary by either the process flow itself or by the responsible actor. It is possible to capture the production process with object-oriented modeling approaches, such as UML, but it is labor intense and complex. This might also

be the reason why there have been attempts to introduce various modeling approaches such as Petri net, or IDEF0 into manufacturing.
Future Task: The design methodology should respect the process driven perspective required in the MES layer and try to manage it by offering a less extensive and complex concept to capture the possible process paths.

Mutual dependency
While object- and agent-oriented approaches uphold characteristics such as autonomy, encapsulation and emerging behavior, those characteristics are problematic for a process control in the manufacturing domain. To produce a good a sequence of tasks depending on one another is required. The autonomy of one entity might impede the next task. On the other hand, autonomy and encapsulation are highly desired, if their negative effects can be prevented.
Future Task: Concepts like autonomy, proactiveness and goal orientation can be incorporated into the production control rather than into a lower layer. To incorporate those concepts, the design methodology must include mechanisms paying special attention to mutual dependencies and encapsulation potentials. Thereby, the most suitable balance between two options must be found. The first option is a system with the narrowest encapsulation setting and the lowest mutual dependency, as each service is divided into fractions. However, such fractions may have no value-adding capacity. The second option is a coarse encapsulation, leading to a higher level of mutual dependency between each service. Such a system is more holonic, as more mature services can be provided. The chain of interaction in such a system is smaller than in the first mentioned system. But this system lacks robustness, as the mutual dependency between the entities and their roles is high. A proper design methodology should give the designer clear rules or guidelines for clustering service fractions.

Sensitive System A program error handling mechanism is based on a combination of error catching and concurrency mechanisms, restoring a safe state. A manufacturing execution system has mechanisms to catch errors and safe states to which it can return, but since it has numerous real-life interfaces, this procedure is more complex and time consuming. A production control in such error cases is inert and rigid since it takes time and human effort to restore it to a safe state. This characteristic makes the system sensitive to unforeseen errors.
Future Task: A MAS design must cope with this issue by only implementing the agents on a higher MES layer leaving a sufficient buffering zone between the MAS and the error handling mechanism. Taking the sensitivity of the system into account, a multiagent system as production control cannot be implemented purely as interface agents. The task is to provide an encapsulated service of one or more production assets to a multiagent system, not to lash the production asset to its interface agents as a digital one-to-one representative. An interface agent error or exception would effect the asset directly and must be prevented. An agent on higher MES layer offering only the encapsulated services, leaves enough buffer for the system to cope with an error or denial, as only successful trades and interactions will be transmitted. This design choice ensures that an agent breakdown will not cause any harm to the production asset or its availability.

Highly heterogenic hard and software structure Manufacturing overlaps traditionally a heterogeneous landscape of technologies. In the industry, a shop floor is being conceptualized as a holistic design in its beginnings. After several years, parts of the shop floor need adaptions. This partial modification also raises the heterogeneity of technologies on

the shop floor. A multiagent system on the MES layer is effected by this heterogeneity, although not as strongly as a multiagent system on the lower level.

Future Task: A MAS must either be applied as a platform independent solution or on a platform with multiple interconnection possibilities. The first solution requires a high programming effort. Each production asset and its agent must be conceptualized individually and then be integrated into the agent society. This solution is complex and difficult to consider during the design process since the different ways of implementation effect the agents' interactions. The second solution implies that agents will be designed and implemented on a single platform. This solution ensures the design freedom, the agents' interactions, and the feasibility of the implementation. The downside of this solution is, that the interface possibility between each agent and its production resource is limited by the range of interfaces the platform supports. This aspect must be considered during the design phase, as it is possible, that not all services mapped and designed into an agent can be realized due to missing interfaces. Such a MAS platform could be, for example, JADE.

To summarize the requirements for desired MAS design methodology in concrete terms:

Model fidelity
The MAS design methodology for the manufacturing execution system domain should provide an acceptable model fidelity. Its model fidelity will be evaluated through the resulting MAS's capability to maintain the core process.

Prescriptiveness
Prescriptiveness describes the quality of assistance that the design methodology offers. This continuous assistance appears as a guiding procedure, in which the results of the previous step are the inputs of the next step. Further, the design methods are described precisely, including their operation, their initial state and their final output. The designer shall at no time be forced to improvise or guess the next step.

Feasibility for industrial and mechanical engineers
Since industrial and mechanical engineers are the target group, the design methodology must correspond to their common knowledge. While programmers mainly share object-oriented perspectives on systems and approaches, mechanical and industrial engineers are trained to use a process-oriented perspective. Tools and methods offered by the design methodology should ease the transition from a process-oriented perspective to an object-oriented perspective. Considering the target designer, the modeling tools should be limited to combination of textual description and structured analysis. Further the design methodology should exclude any implementation decisions or communication protocols. Such tasks are best handled by programmers with MAS experience.

Capability to model agent-oriented systems in the MES layer
Since the designer has no prior MAS-knowledge, the design methodology must include MAS concepts embedded in its design methods, such as encapsulation, autonomy, etc. It should further respect the specific demands of a process on the MES layer.

1.2 Presented approach

Although the benefits of multiagent systems (MAS) as a process control are apparent, MAS are rare in the industrial practice. One reason is the lack of design methodologies specified for mechanical or industrial engineers as target designers and adapted to their capabilities.

This doctoral thesis is built on two pillars. The first pillar is the utilization of multiagent system (MAS) as manufacturing process control. This approach uses the tool cycle of the TU Wien Pilotfabrik, a small size production. The TU Wien Pilotfabrik is a demonstration factory offering students and scientists a production environment with state-of-the-art technologies. The MAS maps the true tooling cost for each order as over time the orders arrive online. The true tooling cost, which is caused by the tool supply and consumption, can now be recorded and assigned to each order and taking the current shop floor state into account. Beside that, the MAS tries to minimize the real tooling cost, if possible through agents negotiating a beneficial online order sequence. The sequencing is addressed from a process control perspective, and not as an optimization issue. Therefore, the success of a process is preferable to an optimal order sequence. The MAS operates on the MES-layer and manages already released shop orders. Since successful tool supply and machining have the highest priority, sequencing the machining orders is only possible to a limited extent and with respect to the situation on the shop floor at the time of order release.

The tool cycle (also called tool lifecycle) was chosen as demonstration process, since it manages physical and digital entities encompassing different production layers. Its functions are spread over three production layers with individual time demands. The tool cycle touches services on the ERP layer, as well as services on the MES layer. The tool cycle exchanges data between different layers and time frames and covers a wide range of platforms and interfaces. It therefore reflects the highly heterogeneous hard- and software landscape found in the manufacturing domain. A wide range of standards exist that define tool components and tools, such as the ISO 13399, which defines the exchange data, or the DIN 4000, which defines the technical properties. Those standards benefit the full digital representation of cutting tools.

Despite those existing standards and the opportunities offered by them, the tool cycle has not seen many innovations in the last years. The biggest challenge that the tool cycle faces is its highly heterogeneous hard- and software landscape, causing missing data exchanges and a lack of interfaces. Although it can be assumed, that the tools have a high impact on the overall production cost and thus a high saving potential, the true tooling cost can only be estimated. According to Sharit and Elhence cutting tools cause 25% of all operation costs [SE89]. Gray et al revealed in 1993, by reviewing industrial data, that tooling accounts for 25 up to 30% of variable and fixed production cost [GSS93]. New sources, or sources detailing the cost caused by missing tools or delayed set-up times could not be found.

This missing information can be explained by the high effort it would take to trace individual tools throughout the process. Therefore, only expensive customized cutting tools are likely to be monitored. In any other case, enough replacement tools are being stocked next to the machine tool to prevent idle times. The main incentive for using this strategy is that the tool cost is not factored directly into the production. The low per-piece-cost of standard cutting tool components supports this strategy. Before the tool cycle at the TU Wien Pilotfabrik can be used as a demonstration process, its data interruptions must be resolved.

As groundwork, for this thesis, all linkage issues of the tool cycle in the TU Wien Pilotfabrik were resolved. The results were published during the CIRP ICME in Italy as the “integrated tool lifecycle” [FTP19]. The integrated tool cycle developed in the pilot factory at the Vienna University of Technology allows the tool costs to be determined for each order, while considering the situation on the shop floor at the time of order release. Chapter two will present the integrated tool cycle, which serves as application domain for the presented multiagent design methodology. The designed MAS captures the possible tooling cost and aims to sequence the orders to achieve a lower tool cost. This sequence shall be achieved by utilizing price-based negotiations between the agents, which decide independently whether to accept or deny an offer.

During the first attempts to design such a multiagent system for the integrated tool cycle, the lack of feasible design methodologies for the manufacturing domain was discovered.

The second pillar of this thesis is the development of a multiagent design methodology, which meets the specific manufacturing requirements and supports accomplishing the first pillar. The **Design Methodology for the Manufacturing Execution System Domain (DM-MESD)** presented in this doctoral thesis, especially addresses mechanical and industrial engineers. It offers the designer, in contrast to already existing design methodologies, the possibility to evaluate if the design methodology is suitable for the desired application in advance. For this purpose, the DM-MESD defines an investigation area, which serves as a gauge for other applications. Comparing the application process with the investigation area resolves if the DM-MESD can be used for a specific application. The design methodology presented here consists of six phases, beginning with the evaluation phase. The DM-MESD builds services out of existing functions and tasks and creates new services from scratch. While existing services cover the original process, new services introduced by the methodology extend the functionality of the process. The DM-MESD uses textual and structural analysis methods to elaborate existing tasks and functions from which existing and new services can be built. In its clustering phase the DM-MESD combines services into service clusters based on three clustering rules. The resulting service clusters are the foundation for the future agent types. In the transformation phase, the last phase of the DM-MESD, those service clusters are transformed into agent types, by defining roles, interactions and multiplicities. DM-MESD is a pure designing methodology without any implementation mechanisms or preferences. It does not specify communication protocols, nor does it suggest any ways of implementation. Since the target designers are mechanical and industrial engineers with no prior MAS-knowledge the DM-MESD suggests no way of implementation. Nonetheless, implementation decisions are best reserved for programmers with MAS experience. DM-MESD is a platform independent design methodology, it does not favor any programming language, or any agent implementation strategy. Although the MAS presented here is implemented in JADE, the final MAS design, could be implemented on any platform. The resulting MAS design is described in a textual and structural manner, leaving the programmer freedom in his or her choices for implementation. The presentation of the DM-MESD is divided into two parts. First the methodology itself and then its application on the integrated tool cycle is presented. Using the DM-MESD on the integrated tool cycle has two objectives. The first is demonstrating how to apply the DM-MESD step by step, and the second it offers an insight into the methodology's feasibility and the results, that can be achieved by applying it. Both aspects will later serve the proof of concept.

The final MAS is a flexible process control reflecting the current situation on the shop floor and online scheduling the released machining orders in the best possible manner. It thereby aims to minimize the tooling cost and therefore the production cost of each machining order. This management of released machining orders is achieved by the MachineAgent offering the OrderAgents a production price and throughput time for each available position in the machining queue. If there is a chance of a sequence with lower tool costs, the MachineAgent starts a new offering process. In the new offering process, it tries to buy back the already sold positions in the machining queue, by either offering the owners a lower production price or a shorter throughput time. Thereby the price reduction consists of half the tool costs saved with the new sequence. Since each of the agents is designed to achieve its individual goals, this renewed offer process may or may not be successful. The OrderAgent decides whether to accept an offer or not, based on its individual parameters, such as the maximum order execution time, the maximum cost, and the time factor. If, for example, the magazine is currently loaded with cutting tools required for OrderAgent B and their tool lifetime is sufficient to machine the order, the MachineAgent

offers OrderAgent A a price reduction, if it gives up its machining queue position in favor of OrderAgent B. The MachineAgent however only revises its offer, if the current circumstances of the shop floor allow it. Those circumstances include the remaining lifetime at the machine tool, the remaining lifetime currently available at the storage, the time required to supply and commission the cutting tools to the machine and the time left before the setup must be started. Those circumstances are random process parameters, which reflect the reality on the shop floor and must be considered by a successful production process control. If all those circumstances are in favor of rescheduling the offer, the MachineAgent proposes such offers to OrderAgent A and B and awaits their affirmative answer before proceeding. OrderAgent A could also refuse the offer, although a lower production cost (tooling cost) was offered. On the one hand this design choice primarily reflects the reality in which orders are processed according to their priority while the expenses are secondary. On the other hand, this design choice expresses the freedom of decision that the OrderAgent has, since its primary goal does not necessarily coincide with the MachineAgent.

The final MAS design for the integrated tool cycle is simulated using JADE and will not be directly implemented into the TU Wien Pilotfabrik. Although JADE compatible interfaces for each machine and asset are possible, it would exceed the scope of this thesis, and require a sufficient recovery strategy. Such a recovery strategy includes proven mechanisms to return to a safe and correct state from which the system can be restarted again. The presented approach uses a system simulation in JADE without any backup recovery mechanism.

1.3 Investigation area

Since manufacturing is a broad term, confinement is required. This confinement is achieved by describing the investigation area of this thesis. Defining an investigation area, serves the comparability of this work and the definition of its extent. Further, the investigation area clarifies the target domain of the presented design methodology for the manufacturing execution system domain (DM-MESD). The investigation area reflects the defined requirement of the DM-MESD. At the same time, it serves as a gauge for other applications of the DM-MESD. Comparing the application with the investigation area resolves if the DM-MESD can be used for a specific application. The survey of design methodologies will show that only a few existing methodologies have a defined application domain. Among those, no design methodology defines it in a comparable preciseness.

With respect to the production process itself, the focus lies on **discrete manufacturing** (see figure 1.4). Other than the process industry, the output of the discrete manufacturing is processed individually and countably. Its process control faces different challenges than the process industry. The production is also differentiated by the produced quantity. On the scale of the production quantity two extreme kinds of production exist, the job-shop and the mass-production. The size of the production addressed in this investigation area ranges between a **job shop and a serial production**. The tool cycle as the chosen demonstrator fits into this investigation area. The application layer of the investigation area is also limited. Although the DM-MESD can be used theoretically on processes from the ERP its field of application is mainly the **MES layer**. The application process should however not be located lower that the MES layer.

1.3.1 Systematic literature study

As a first step, a systematic literature study is conducted in the previously described investigation area. The objective is to gain information on common design methodologies for multiagent

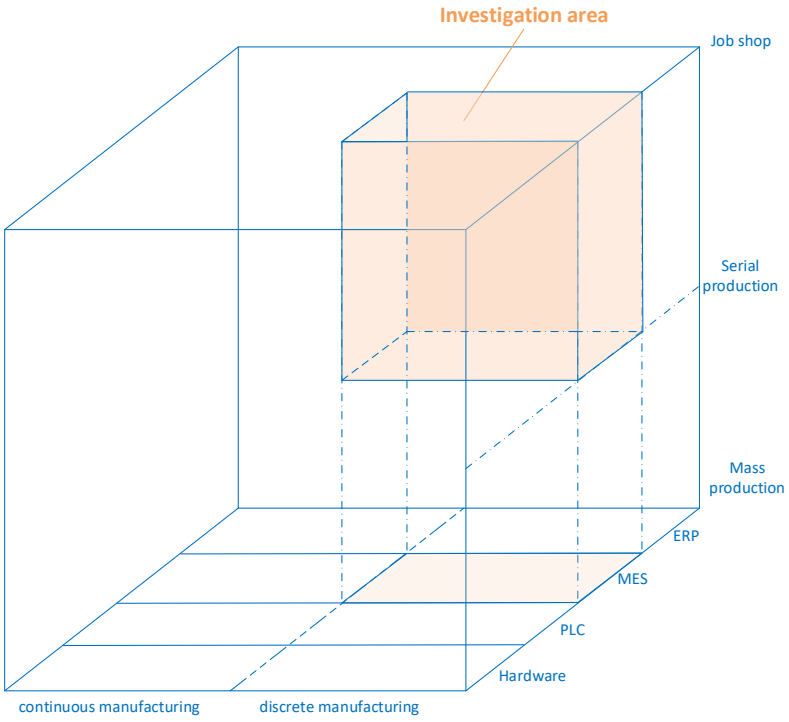

Figure 1.4: Investigation area visualized

systems. Since the scope of this thesis covers three courses, tool cycles, MAS and MAS design methodologies, the systematic literature study must be limited to a process-able amount. This search is not limited to an explicit number of sources, but to the content that fits the investigation area. The first part of the literature search addresses the tool cycle. This search shall define the tool cycle, its core process, its weaknesses, and its open potentials. Due to the closeness of the tool cycle to industrial practice, this search is not limited to scientific publication, but requires the knowledge of process stakeholders and professionals. The result of this search will be presented in section 2.1. The second course addresses MAS. Since the volume of publications in this area is tremendously high, the search must be limited. Beside more general terms and concepts directly bound to the MAS paradigm, the review is limited to MAS applied in domains comparable to the investigation area. This excludes MAS on the device or Programmable Logic Controller (PLC) layer. Additionally, the search is limited by excluding approaches, which are located higher than the MES layer. Among those, MAS approaches using the FIPA Specification will be favored. The remaining literature offers enough volumes and more importantly an active scientific community, which covers all required fields of interest. This search focuses predominantly on the Elsevier's database, Scopus. Since FIPA has been integrated into the Institute of Electrical and Electronics Engineers (IEEE), the search will be extended to the IEEE database. Since MAS source from computer science, the journals of the Association for Computing Machinery (ACM) are included in this search. To cover relevant patents, books, white papers or reports, the same keywords and search mechanism are used on the Google scholar search engine. Possibly doubled literature is not being considered. The above-mentioned restrictions limit the search to the following keywords:

- MAS
- FIPA

- JADE
- MES
- Manufacturing domain
- Manufacturing control

The last course in this search addresses MAS design methodologies. It covers the same journals and databases as in the search for MAS. First a more general search for available MAS design methodologies is pursued. This general search offers an overview of the existing methodology landscape. Based on the results, the methodologies are limited to those widely evolved and used. Among those methodologies, those with a suitable modeling concept for mechanical and industrial engineers and are favored.

- Multiagent design methodology
- Multiagent design method

The comparable work of other researchers is presented in the respective sections of the state-of-the-art chapter.

1.3.2 Evaluation by proof of concept

The last step involves validating the suitability of the presented MAS design methodology for the manufacturing execution system domain (DM-MESD). This validation is achieved in three stages. First it will be examined, if the DM-MESD fulfills all domain specific requirements and all requirements set for a proper MAS design methodology. Then the resulting MAS will be reviewed to conclude if the methodology provided the desired result. The review focuses on the benefits achieved by using the DM-MESD and on the documentation of the final MAS design it provides. The final stage of the review offers a conclusive summary, comparing the DM-MESD with the PASSI and the Gaia methodology.

CHAPTER 2

State of the art

This chapter provides a brief but adequate understanding of the three major topics this thesis examines. First the tool cycle, its issues and its potentials will be addressed. This section describes the role of the tool cycle in production and gives a brief insight about cutting tools, standards, data management and their importance in the manufacturing process. Afterwards, the state-of-the-art tool cycle will be briefly outlined to show the linkage issues it faces. The "integrated tool cycle" developed to solve the linkage issues for the TU Wien Pilotfabrik is presented besides other tool cycle approaches in this section. The second part of this chapter addresses multiagent systems (MAS) and their characteristics in general. In this section, topics such as autonomy, encapsulation, emergent behavior, FIPA Specification, Agent Communication Language (ACL), and JADE will be introduced. This section shall broaden the reader's MAS understanding and to highlight the paradigm's strengths and weaknesses in the manufacturing domain. The third section of this chapter will address multiagent design methodologies. This section introduces terms, such as method and methodology and presents how design methodologies evolved. First, it presents commonly used design methodologies, offering the reader an overview of common software and MAS design methodologies. Then it will review in detail the top down approach Gaia and afterwards the Agent UML based PASSI methodology.

2.1 Tool cycle

2.1.1 Cutting tools

Cutting tools are an important production resource, as they play a major role in a span removing manufacturing process. Cutting tools, the NC-program, and the clamping systems count to group of production resources. A cutting tool used in machine tools is an assembly made of components from the ISO 13399, as figure 2.1 displays. An assembled cutting tool consists of a machine side adapter, a tool holder (optional) and a cutting tool, which is an assembly or a solid tool component. Cutting tools used for CNC manufacturing consist of tool components, offers many tool assembly combinations (see figure 2.2). Each tool component has its own set of data, such as material, dimensions, vendor, cutting conditions, price, name, etc.

One reason why the tool cycle was chosen as a demonstration process is that cutting tools are highly standardized. Although numerous specifications exist, only the two most relevant are introduced here briefly, the DIN 4000 and the ISO 13399. Both standards are important for the data representation of cutting tools.

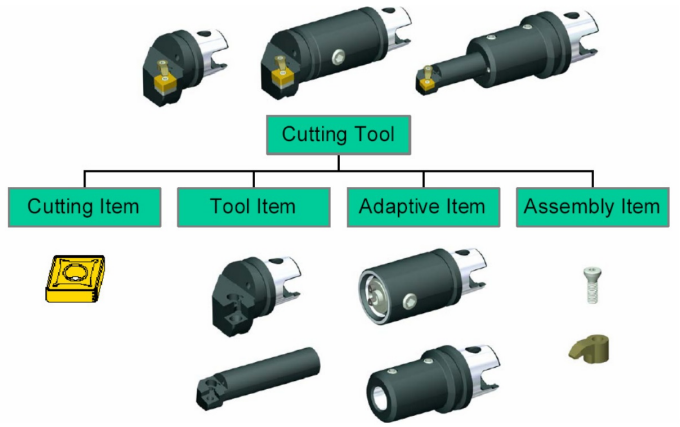

Figure 2.1: Main groups of cutting tool components; Source: ISO 13399

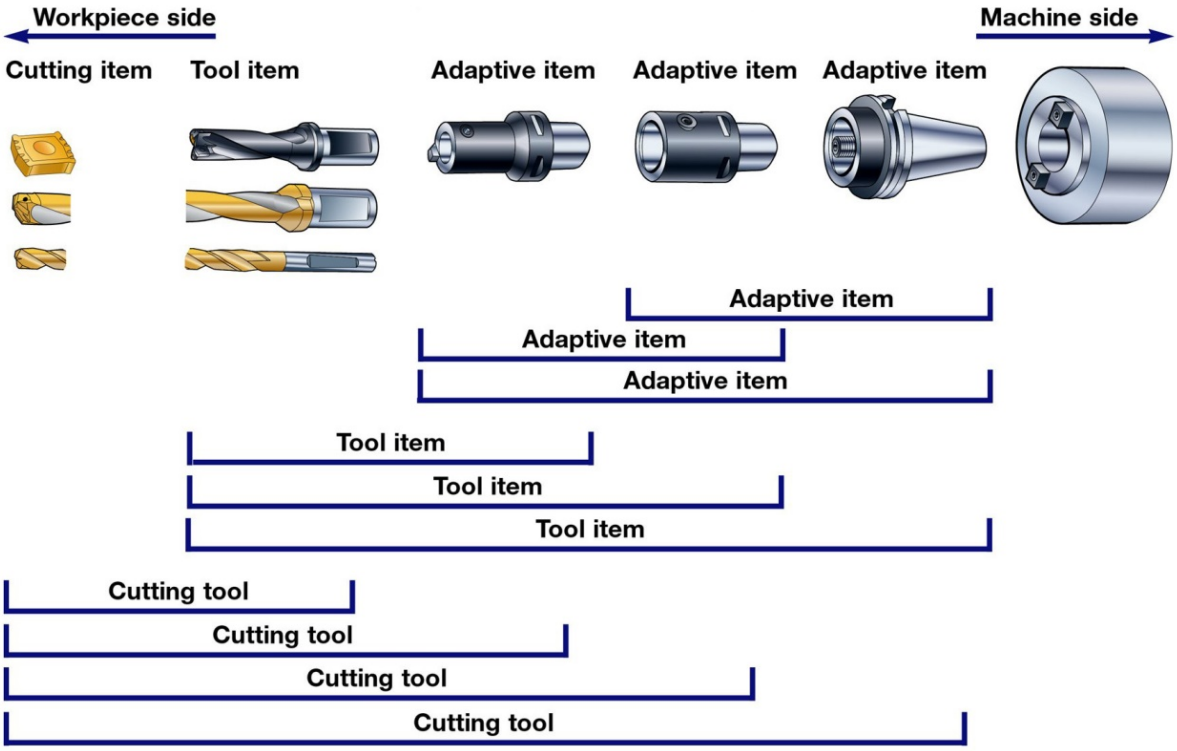

Figure 2.2: Assembly options according to the ISO 13399

The **DIN 4000** (German industrial standard) specifies the tool types, geometrical attributes, material and data properties. The DIN also defines the interface coding for each tool component, so that filter functions can be used for assembling. The interface coding is a string that classifies the interface specification of each component. It includes coupling type, form, dimension, version, and kind. With this interface coding the operator can limit his or her search for fitting components. This interface coding does not only ease the assembly of cutting tools, but it also enables their digital representation.

ISO 13399 is an international standard defining the cutting tool data representation and its exchange. It has its beginnings in the doctoral thesis of Nyqvist [Nyq08]. Its aim is to provide an information model for the computer-interpretable representation and exchange of industrial product data about cutting tools and tool holders. It offers a tool component and data representation free of proprietary format constraints. This data representation enables implementing and sharing product data bases. The data model defined in ISO 13399 uses the EXPRESS modeling language. Its data model can be exchanged either by STEP file or STEP-XML. Based on the ISO 13399 and DIN 4000 tool management software can generate the data set of an assembled tool automatically from the data of its component. This tool data can be exchanged between CAM and ERP systems, easing tool data management, and enabling the manufacturing process simulation in CAM software with the future tool.

2.1.2 Tool management

The tool cycle is embedded into tool management as it covers three of its sub-processes, the tool disposition, the tool supply and the tool use. Tool management is therefore introduced briefly. Tool management is a cross-section process, managing the tool data, as well as the physical cutting tool items. Thomas Geib depicted the tool management as a graphic, which was translated, adapted and digital revised by the author (see figure 2.3) [Gei13]. The graphic shows, that the task range of the tool management spans from tool planning with a long planning period, to tool use with a short planning period. **Tool planning** is located on the ERP layer, while the tool use falls into the machine control's field of action, demanding a faster reaction time. Tool planning starts by defining the range of cutting tools a producing enterprise requires. It is a strategical question, choosing between a small variety of tool types, or a large variety but shorter tool use time during machining. CAM software with manufacturing simulations can determine the future tool use time. Based on this information, the desired tool range and estimate tool requirement is determined Although the tool planning utilizes product data from the CAM, its decisions are yet order-independent.
Another part of the tool management is the **tool logistics**, which manages the tool procurement and inventory on the enterprise level. Tool logistics includes the preceding determination of the tool requirement based on the expected production volume. It also marks the transition from the order-independent to the order-specific, as it considers long term tool requirements based on the expected amount of orders.
Tool disposition manages the tool scheduling, the supply planning and the determination of the short-term tool requirements. It marks where the tool management enters the MES-layer and where, according to Geib, the tool management becomes task-oriented instead of planning-oriented. The tool disposition is also the entry-point of the tool cycle.
Tool supply handles all supporting tasks to deliver the required amount of cutting tools to the machine tool. It includes tool preparation, collecting the cutting tool components, and assembling them, as well as measuring their dimensions or adjustment to preset values (pre-setting). The tool supply also covers the tool commissioning to the machine tool. Tool supply also includes the reworking of the cutting tools, which is an up-following step of the tool use. The tool supply also

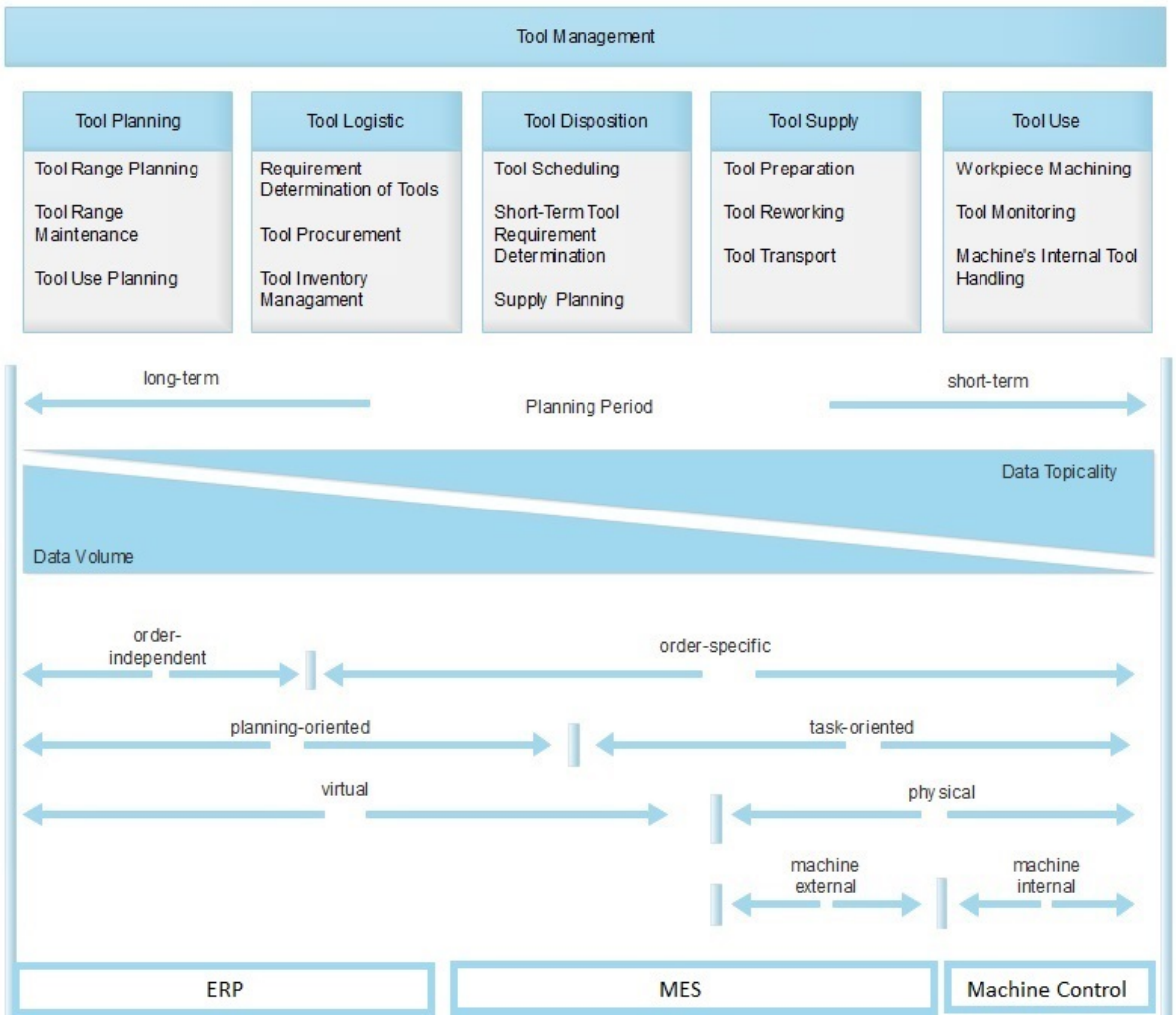

Figure 2.3: Tool management scheme, original source [Gei13], digital revision and translation: author

manages the disposal of tools that can no longer be reworked. Before the tool supply, cutting tools were only considered based on their data representation. This data representation included cutting tool master data. When the tool components are assembled, a new type of cutting tool data is being instantiated, the tool instance data. The tool instance data is a set of data that belongs to a unique cutting tool instance and consists of the tool inventory number (unique resource identifier), its measurements, its remaining lifetime, the critical lifetime limit, etc. This data is only being handled in the MES and on the machine control layer. Some data points of the tool instance data are dynamic data, which means that they are periodically updated, such as the remaining lifetime, or its measurements. The tool supply is the first sub-process to generate machine tool relevant data and to manage it outside the machine tool (see „machine external„figure2.3).

The **tool use** is the fifth and final sub-process of the tool management. The German language distinguishes between tool use or use of tool as a process or process step (Werkzeugeinsatz) and another word for the tool use (time) as part of the machining time (Werkzeugeingriff). To prevent future confusion, the author refers to the time [minute] in which the cutting tool is removing chips from the workpiece as **tool use time**. Tool use takes place inside the machine

tool and is limited to this location. It includes the machining, the tool monitoring, and the machines' internal tool handling. It demands the highest data topicality, as it corresponds to the machine control reactions time, which ranges in milliseconds.

While scientific efforts are made to optimize the cutting speed and quality and to equip cutting tools with sensors, the tool management has not received as much scientific attention as it should. In fact, many scientific publications exist, reviewing various cost optimization or efficiency strategies for other production resources than for the order specific tool supply and its costs. This can be reasoned by the shear amount of data and physical items, that must be managed during manufacturing. Two similar assembled tools, have a different data set, as their circumstance of tool use, such as tool wear and measurements vary. Due to this high management effort, producing enterprises do not factor the tool cost directly into the order processing, but rather count them as fixed production costs. This view identifies cutting tools as assets of the machine tool, such as electricity or cooling lubricant. As such assets neither their consumption nor their replacement is being tracked. This view is common in manufacturing. Reasons for this are:

High physical management effort
: A cutting tool consists of several components, which must be handled and tracked.

High data management effort
: As not only the data of assembled components, but also the data of single components must be managed.

Low per piece price
: Standardized cutting inserts and items, broad competition and the low per-piece price at large quantities, lower the purchase price and make a higher stock level affordable.

Information breakpoints during the tool cycle
: There are still data interruptions in the tool cycle, such as between the machine tool and the Tool Management Systems (TMS) or between the storage and the TMS.

Low variety on cutting tools
: The tool range is minimized to standard cutting tools to purchase large numbers at low prices and to save on logistic efforts.

The standardization of tool components not only drives the competition between the vendors. It also lays the base for the Tool Management Systems (TMS). TMS are software solutions to manage the tool management at different phases. The high grade of standardization of cutting tools enables the data exchange between tool management system (TMS) and CAM software or ERP- software. With the interface to the TMS, that a CAM offers, NC-programs can be optimized to shorten processing time, but also to utilize tool types already in storage. By doing so, the variety on cutting tool types can be reduced as well as the per-piece price. Those points might be the reason for the lack of information about costs caused by missing or delayed cutting tools. In fact, only a few scientific publications can be found, which have elaborated the tooling costs for a case study or a best practice setting.

In 2016 Manogharan and his colleagues published an economic model for a new hybrid manufacturing method and its analysis [MWH16]. This method included 2D layer printing and CNC milling. To compare the costs of this hybrid method, the authors built a case study and captured the CNC and the additive manufacturing cost in detail. Figure 2.4 summarizes the partial and total manufacturing cost of the CNC Rapid Prototyping. Their case study shows that tooling costs have a large impact on the operating cost and on the total cost including material. The

CNC-RP stage	Setup time (hrs)	Machining time (hrs)	Number of tool changes	Tooling cost ($)	Tool change time (hrs)	Stage time (hrs)	Operating cost ($)	Total cost including material ($)
Roughing and hogging	0.17	14.04	8	160	1.33	15.37	417.50	1358.25
Finishing		9.38	5	100	0.83	10.21	276.00	
Total CNC-RP	0.17	23.42	13	260	2.17	25.59	698.25	

Figure 2.4: CNC-RP operation and tooling cost for sample parts, source [MWH16]

tooling costs, which reflect only the number of tools worn out for production summed up to 260 $, which is 19,14 % of the total cost. Thereby the authors assumed the average tool lifetime to be 100 minutes of machining time and tooling cost of $20/tool, which fits industrial settings for a small to medium sized production. In their calculation model, the authors counted the tool setup times as part of the machining time. The summed-up tool setup time was 2,17 hours and made up 9,26 % of the overall machining time. The operating cost rate is assumed to be $ 25/hour as the average rate according to the U.S. Bureau of Labor Statistics in 2013. This case study gives a glimpse on the potential cost that could be saved by drawing more optimization efforts into the tool management. International and national specifications opened the way to overcome systematic linkage issues. Those standards enabled today's practice of generating NC programs in a CAM Software, which include enterprise specific tools. With the standardized tool data, a TMS software can generate setup sheets and managing the tool component stock through an interface with the ERP system.

2.1.3 State of the art

A review of the current literature showed that the state of art can be divided into solutions addressing interface issues and those which address the refining of the tool cycle process.

Approaches addressing the interface issues

Wang et al offered in their publication a tool management methodology based on using RFID tags as the driving technology [WNY+09]. The authors saw the tool management process divided into tool preparation and tool use, due to a lack of connectivity. During tool use, RFID tags with a designed encoding shall solve the linkage issue as a bridging technology. The encoding included a tool instance identifier and the tool length and diameter, as part of the corrective geometry. The tool instance identifier is divided into a group number and a tool instance number. According to the authors the integration of a RFID-based tool management system reduced the tool-setting times and prevented possible mistakes.

Denkena et al. utilized a central tool management system in combination with a unique identification number (tool ID) [DKS14]. The authors stated that the production is currently not capable of providing sufficient feedback on tool condition. Like the previous approach, Denkena et al did not store the data required for the tool use in their centralized tool database. The authors used a self-contained data format, called Gentelligent Markup Language (GIML). GIML is an XML based textual representation of product and production-related information. The authors stored the relevant tool data directly on the RFID tag. At this point, their approach differs from the here presented "integrated tool cycle", which only stores the tool instance ID (inventory number) on to RFID tags and requests the rest of the data from the data base. Each individual system along the tool cycle reads the tool ID and requests the updated data from the TMS webservice. Thereby unnecessary data redundancy is prevented.

The team around Hassan introduced in their paper [HK16] an "Integrated Cutting Tool Management and Monitoring" system (ICTMM). ICTMM was designed to exchange cutting tool information between the shop floor and any virtual system. On the machine side this approach utilized the RFID tags to store the tool ID and the tool life estimation. The ICTMM was used as a monitoring function on the shop floor. The ICTMM represented the tool data in a STEP-NC file to a higher layer. This approach covers the highest range of monitoring functionality yet to be reviewed. It compared to the here presented approaches it does not offer monitoring data to a higher layer through a HTTP based format. Nor does this approach offer a direct interface to the machine tool with life-data. The tool lifetime utilized in this approach is only estimated until the RFID tag can be read and evaluated.

Approaches refining the tool cycle process

Schaupp et al. proposed in their publication a framework for approaching the digitalization in tool management [SAM17]. The authors offered three sequential steps for enterprises to evaluate the stage of tool management. Thereby the authors provided an evaluation methodology for stakeholders beginning with individual goal definition, such as high tool availability, and a low tool inventory etc. The second step determined the digitalization levers and their influence on predefined goals. The last step measured the implementation process, by presenting a Readiness-Model. Schaupp et al. pointed out that RFIDs are not used for the track and tracing of individual tools, but rather for storing the remaining tool lifetime. The authors emphasized the lack of a direct interface between the machine tool and the tool database. Therefore, the remaining tool lifetime stored on the RFID cannot be automatically transmitted to the database without further effort. In fact, the remaining tool lifetime is only considered for special tools. This practice is reasoned by the high purchasing cost special tools have. This is where the integrated tool cycle of the TU Wien Pilotfabrik steps in. The integrated tool cycle provides an automatic data entry of the tool lifetime as well as a track and trace function of the RFID tag through the single tool lifecycle stations and represents a digitalized tool management.

Another approach, which refined the tool cycle is the "Digital Twin of a cutting tool" approach by Botkina et al [BHO+18]. The authors presented a digital representation of the cutting tool through all its lifecycle. This digital twin ran through all stages that the physical item is currently passing. To collect the data the authors used the previously developed event-driven line information system architecture (LISA). LISA is an event-based SOA, allowing other applications to render themselves as services. LISA is using the enterprise service bus (ESB). The digital twin of the cutting tool tweeted its data to LISA to store it. Those tweets were detailed and specific messages from a so-called tweeting machine. The tweeting machine was a machine tool sending tweets or real-time data. Tweets were published to the ESB by using LISA architecture. The CNC machining data, such as cutting path, forces, and tool use time, is based on STEP-NC. The tool data tweeted during the machining includes measurements of parts, tool compensations and changes, cutting data, etc., all with STEP-NC contexts. To accomplish the tweeting capability of the machine tool, a CNC/PLC interface publisher, a STEP viewer, and receiver applications have been previously developed. The author emphasized two novel paradigms of their approach, the ISO 13399 compliant data flow and collection, and its usage of open source applications such as LISA, tweeting machine, and ToolMaker (Sandvik tool assembly software). The authors also stated that their implementation poses some challenges for the industry. One challenge is that „modern product lines do not typically have the tweeting machine capability“. As the tweeting machine connectivity has been realized by several modifications to existing controllers and PLC-software connectivity. It is not clear if this proposed connectivity prevails in the industry. The integrated tool cycle of the TU Wien Pilotfabrik utilizes TMS software, which is not only an industrial best practice solution, but is also compliant to the DIN 4000 tool

properties standard and the ISO 13399 tool data representation and exchange specification. The integrated tool cycle presented here aims to solve the linkage issue of the tool cycle to prime it as being a MAS compatible process.

Approaches addressing the scheduling optimization of cutting tools and orders as offline problem

Beside the two mentioned groups of approaches, which both address the tool cycle on the MES layer, there are numerous approaches addressing the scheduling optimization of cutting tools and orders on the ERP-layer. These approaches consider the magazine load or the order sequence as an optimization preemptive problem in planning and scheduling. Mathematical models for tool provision are created and optimized with respect to the desired parameters. Although all these approaches are located in the ERP layer, a few well-known approaches are briefly listed here to clearly differentiate the approach described in this thesis.

Das, Baki and Li addressed the development of a process-planning model for machining cylinder heads [DBL09]. The authors' objective was to plan the magazine load based on production part groups, which have combined design features (DFUs). Each design feature (DFUs,r) processes one or more UMF,o which can be seen as equal to operations. To develop their mathematical planning and scheduling model the authors defined modeling assumptions. Among those assumptions is that each operation is linked to one tool only, and that the tool lifetime is infinite. This approach does not consider randomly filled magazines, nor previously used cutting tools. Their mathematical model offers an optimization of the machine tools internal tool change and the orientation change times.
Hertz et al. published several new families of heuristics for the loading of a tool magazine [HLMS98]. Among their contributions were two new distance functions and two new methods for the tool load problem.
In their paper Baykasoglu and Dereli offered a simulated annealing based heuristic optimization system [BD04]. The authors' objective is to determine the optimal index position of cutting tool types in a tool turret magazine. Different than previous proposals in this area, this approach considers tool duplications and can effectively resolve automatic tool changer index-optimization problems.

Those approaches differ from the approach presented here, as it addresses process controls on the MES layer. On the one hand the goal of this thesis is to create a multiagent system as a process control for the tool cycle. The multiagents system serving as process control tries to schedule orders as an online problem in the best possible sequence to lower the tooling costs. On the other hand, it aims to develop the DM-MESD, a design methodology for multiagent systems specifically for the manufacturing domain. Both the tool cycle as a use case and the methodology are assigned to the MES layer. Therefore, the orders addressed here are already released and can only be sorted to a limited extent, without impeding the manufacturing process. This circumstance is best described as online scheduling, in which the entities that should be scheduled are unknown until they arrive into the system. Krumke, Taudes and Westphal addressed in their work such a non-preemptive scheduling approach [KTW11]. The authors described the concept of online scheduling based on job scheduling on parallel machines.

The MAS designed here as process control shall at any moment in time evaluate the possible tool cost and achieve a beneficial sequence, if the current situation on the shop floor and the free will of the agents allow it. The actual situation on the shop floor is subject to constant changes and is better addressed by a process control approach, rather a scheduling optimization approach. This thesis therefore covers a process control for tool provision and does not address

Order	Qty	NCProgrammID	Status
OF	8	NC12072019	machining
OG	4	NC12072019	preparing
OH	4	NC12072020	open
OI	4	NC12072018	open

Table 2.1: Sequence of released orders

Order	Required toolID							
OF OG	6585289	6585290	6585292	6585298	6585299	6585300	6585304	
OH	6585264	6585289	6585293	6585299	6585300	6585301	6585305	6585306
OI	6585290	6585295	6585298	6585302				

Table 2.2: Compares of required tools for each order

the planning of a job sequence or its optimization. The presented heuristics or mathematical models exclude real life circumstances such as tool breakage, consumption, down times etc.

In order to give the reader an idea of the potential that the MAS can have as a versatile process control of tool costs, a short example is given here. The same example is used in the test scenario IV (see 5.2.5) to examine the MAS capabilities. This example uses four orders of which one (OF) is already being machined and another one (OG) is being prepared for setup (see table 2.1). The order OH was released before the order OI and would traditionally be machined before OI. In the most scheduling optimization approaches the orders are previously known and sorted before they enter the system. The approach presented here, addresses the production as online problem, as arriving orders are unknown in advanced. The MAS process control presented in this thesis will be capable of considering the current state of the shop floor. This current state includes life data from the storage and machine tool such as the magazine load and the remaining tool lifetime or the current job that is being processed.

Table 2.2 displays the required tools for each order. With this information similar tools can be identified. Different than the developed MAS, table 2.2 does not consider the remaining tool lifetime currently available at the machine tool nor does it consider the state at the storage. The current state of the shop floor includes live data from the storage and machine tool such as the magazine load and the remaining tool lifetime or the current job that is being processed. The order OF and OG share the same NC-Program and use therefore the same cutting tools. If the remaining tool life proves to be sufficient neither additional tools nor setup are needed. Determining the net tool requirement demands continuous monitoring of the production and its real-life circumstances. It can not be sufficiently determined by heuristics or mathematical models.

The order sequences with the lowest tooling and therefore the lowest production cost will be found by negotiating agents. A core element of this approach is the machining queue, which is schematically illustrated in figure 2.5. A change in the machining queue is initialized if an order is processed or a new order enters the system and should be scheduled for a different position than the last. However, the sequence does only change if all parties (machine tool and order) agree upon the change. If the order OI can be scheduled prior to OH, the sum of the changed cutting tools would only be five instead of seven, which the original release sequence requires. Table 2.3 shows the tooling costs, which this thesis considered at a cost rate of 10 Euros equal to the machining cost rate. The table illustrates that the new sequence would save 20 Euros in tooling costs. The machining and handling costs of each order do not vary as they depend on

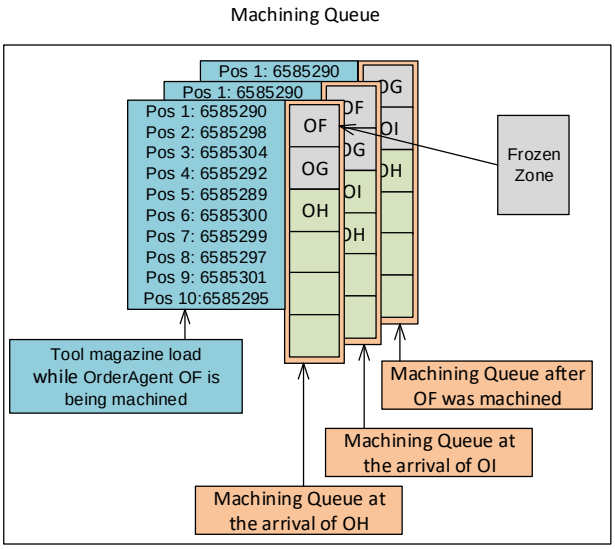

Figure 2.5: Scheme of the flexible machining queue introduce in this thesis; source: own illustration

Comparison of the possible sequences by the tooling costs			
OrderAgent	Tools Qty changed	Tooling Costs	Production Cost
OF	0	0	54,41
OG	0	0	27,21
OH	4	40	64,15
OI	3	30	53,16
OF	0	0	54,41
OG	0	0	27,21
OI	1	10	33,16
OH	4	40	64,15

Table 2.3: Table comparing the original release sequence against the sequence of the online tool cost based scheduling

the production quantity and its machining time. The tooling cost however have (see table 2.3) a strong influence on the overall production price.

The tool cycle in industrial practice

The tool cycle, sometimes also called tool lifecycle, combines three sub processes of the tool management, the tool disposition, the tool supply, and the tool use. Covering all functions of those three processes, the tool cycle includes all necessary functions to guarantee the right number of required tools at the machine tool. Although it plays an important role in production, it has not gained much attention. This might be caused by several data interruptions the tool cycle faces. While the tool cycle in literature is a wholesome process, guiding the cutting tool components and their data from their first demand to their disposal, reality differs. In real world this theoretical tool cycle is not feasible due to the lack of connectivity.

Figure 2.6 illustrates the complete tool cycle. It starts by assessing the production order related tool demand, also called tool requirement. This is classically accomplished based on a tooling list. The tooling list is generated from the CAM program and stored in the setting sheet data. Based on the tooling list the gross tool requirement for the order will be estimated. Theoretically, the

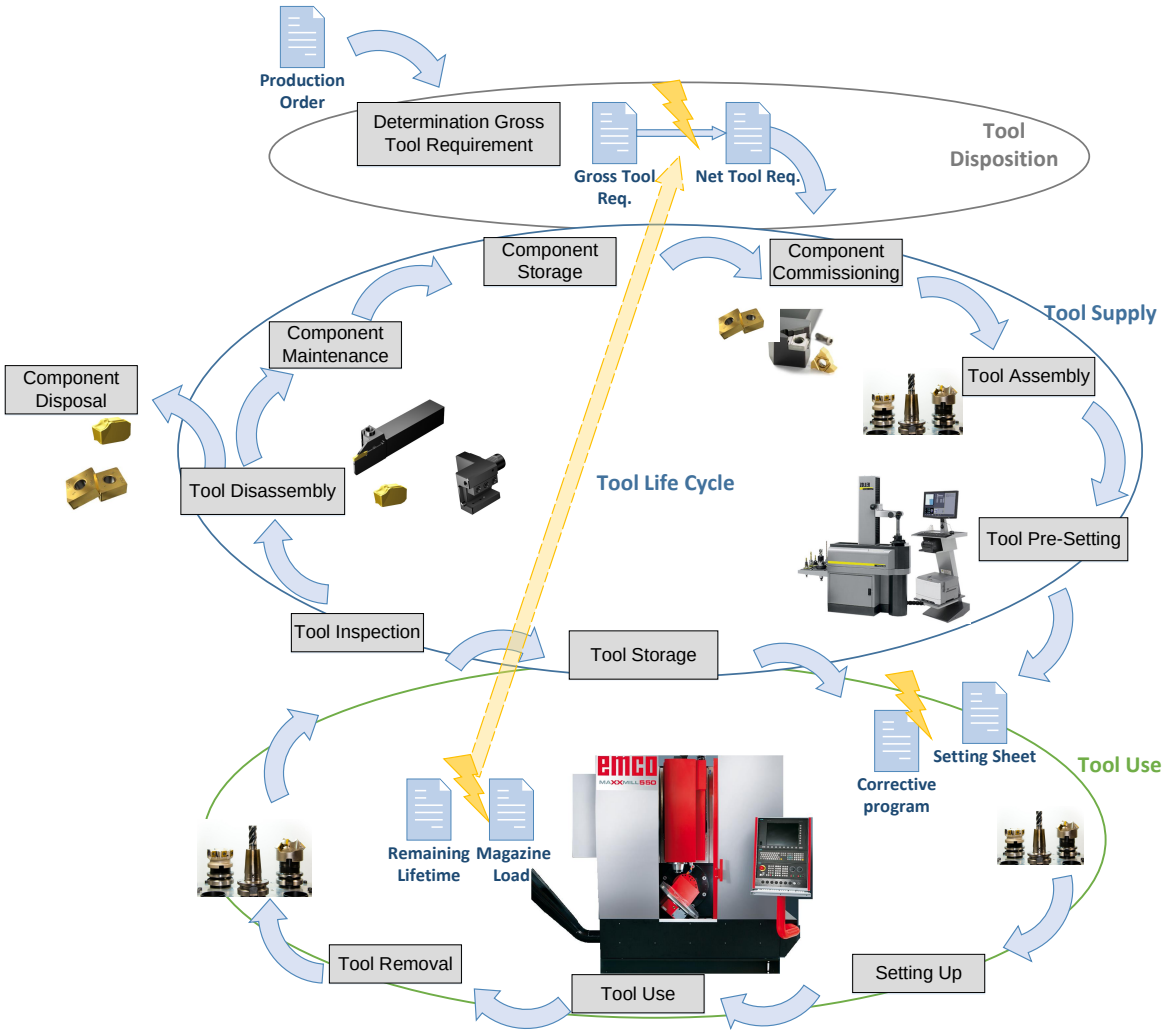

Figure 2.6: Tool Cycle Scheme highlighting data interruptions by flashes; source: own illustration

next step is determining the net tool requirement, which is the sum of required cutting tools for the order minus all tools already available either at the machine tool or in storage. The net tool requirement is therefore always the minimum tool requirement, which is a snapshot of the tools available at the shop floor. Figure 2.6 shows that the first data interruption occurs during determining the net tool requirement. This interruption is caused by the missing linkage to the current magazine load. Some "work around" solutions exist, such as storing a text files of the tool list of the previous order. However, an interface between the TMS and machine tool magazine has not been established yet. Such an interface could exchange the magazine load (tool ID), and inventory number) and the remaining lifetime of the cutting tools at the machine tool. A data exchange exists via Distributed Numerical Control (DNC) between the machine tool and a presetting device or a cell controller with TMS on it. However, those solution are not capable of recalling and storing the current remaining tool lifetime. Based on the net tool requirement, a list of tool IDs and their quantities, the TMS generates a list of required tool components, the picking order.

The components from the picking list are being collected and commissioned by a worker to

be assembled. In some industrial settings, the NC-program refers to cutting tools with preset dimensions. This practice is common if the same tool type is required in a high number. In this case the worker presets the cutting tool dimension to the NC program standard. This makes manually entering the geometrical corrective data into the machine control unnecessary. Small to medium sized productions commonly track and trace their tool instances, since the number of tools on the shop floor is manageable. An inventory number is assigned to each tool instance to store the dynamic tool data, such as remaining tool lifetime, geometrical data etc. From now on, its cyber-physical representation goes through the same changes as the physical tool. To merge the digital representation to its physical item, two common technologies are available, the Data Matrix Code (DMC) or the Radio Frequency Identification Technology (RFID). The DMC is an optical identification technology, which identifies the tool by a two-dimensional code plotted on its tool holder. The surface cleanness can however compromise its reading. RFID tags are commonly mounted into the tool holder. Both technologies join the tool inventory number to the physical tool and to refer to its geometrical corrective data.

After the assembly, the cutting tools are either be measured or preset to a defined measure on the presetting device. A presetting device (in figure 2.6 from Zoller) is an optical measurement device with a TMS terminal. Through the terminal the tool management data base can be accessed and manipulated. The presetting device can also be extended with an RFID read and write terminal. The measured tool geometry is important for the machine tool so it can consider the tools dimensions for its NC-program and prevent collisions during its execution. There are three ways to feed this geometrical corrective data into the machine tool, either by manually inserting it, by storing it on a portable device, or by accessing a corrective program generated by the presetting device. A corrective program is a numerical control code, which includes the measurements of the tool and defines the reference points for the machine control. This artefact can be created by presetting devices with a suitable post processor. After measuring the tools either a corrective program or a setting sheet is generated and commissioned together with the tools to the machine tool. At this point, the second data interruption occurs during the transmission of the dynamic tool data (lifetime and measurements). Although each machine tool can be connected individually with the presetting device through a cell or machine control, this linkage is labor intensive. Besides, this linkage does not enable access the machine tools internal tool data such as the remaining tool lifetime.

When cutting tools and their data arrive at the machine tool the setup can be executed. A magazine is filled by utilizing loading strategy, such as spreading replacement tools over different magazine pockets so, that the time accessing them is shorter. If no magazine strategy is used, the cutting tools will be inserted randomly to the magazine pockets. The machine control assigns the pocket by a Duplo number (internal machine tool number) to the referencing tool type. This Duplo number will be linked to tool identification used in the NC-program. An NC-program could be written identifying a tool by a predefined magazine pocket number. In that case the worker must ensure, that each tool ends up in the right pocket. Another way is flexible pocket coding, in which the cutting tool is identified by a tool name (string or number), which refers to a given name in the NC-program. In that case the cutting tool is assigned to the tool name or type, already stored in the machine control. During the tool use process the machine control counts the tool use time for each tool instance and monitors the machining process.

After dismounting the tool its dynamic tool data is no longer available at the machine tool. At this point the third data interruption occurs. The inventory number and the remaining lifetime can only be transferred manually back to the TMS or by using a RFID tags. Both cases are rarely considered. Therefore, the remaining lifetime of the dismounted tools is not stored. Once dismounted those tools are most likely not be used again to prevent the risk of

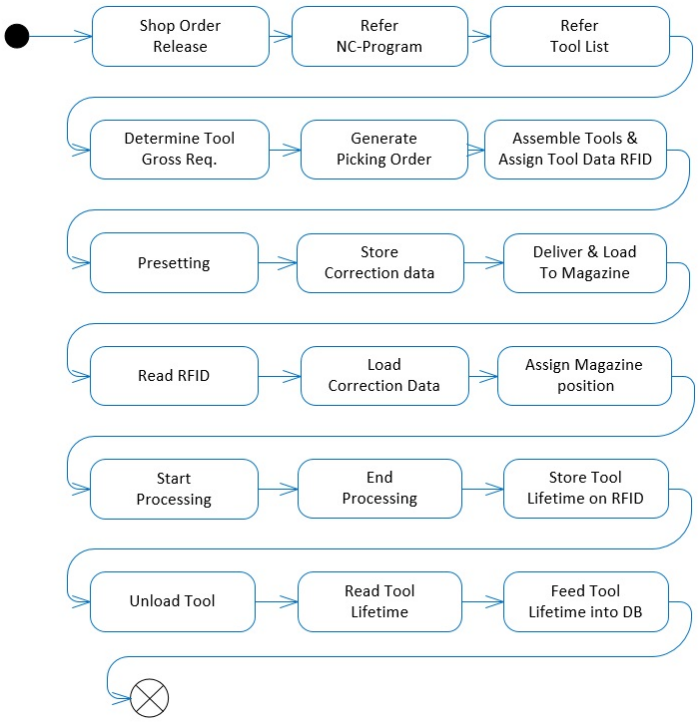

Figure 2.7: Activities of the current state tool cycle; source: [FTP19]

tool breakage. The further use of dismounted tools depends on the company's tool management strategy. Either those tools are considered valuable enough to be tracked and tracked and to fully utilize their potential or they are not. In the first case, dismounted tools are first inspected and then reworked if necessary. In the latter case the dismounted tools disassembled, and the cutting items will be disposed. Figure 2.7 depicts the particular activities of the tool cycle.

Integrated tool cycle in the pilot factory

To resolve the data interruptions requires an interface between the machine tool and the TMS. Through this interface the current magazine load including the remaining lifetime can be accessed and geometrical corrective data can be transmitted. Remote access of the current magazine load enables the determination of the net tool requirement and optimizes the usage of the assembled cutting tools. By resolving this linkage issues realizes the tool cycle as it is mapped in the literature and transforms the real-world process. A digital process transformation is a process restructuration, either by extending its functional capabilities or by optimizing its resource allocation due to newly accessible data. During the groundwork of this doctoral thesis, those technological linkage issues were solved for the tool cycle of the TU Wien Pilotfabrik. The resulting tool cycle is called the integrated too cycle. Its newly accessible data enables determining the net tool requirement and established a continuous process, which could be digitally mapped fully.

Resolving the linkage issues

In this groundwork, a connection between the machine tool and the TMS has been implemented, to remotely exchange the current magazine load and the remaining lifetime of those tools. On the tool management side, setting from the company Zoller has been used, consisting of the TMS Gold software on a host server, a presetting device "Venturio" including a TMS client,

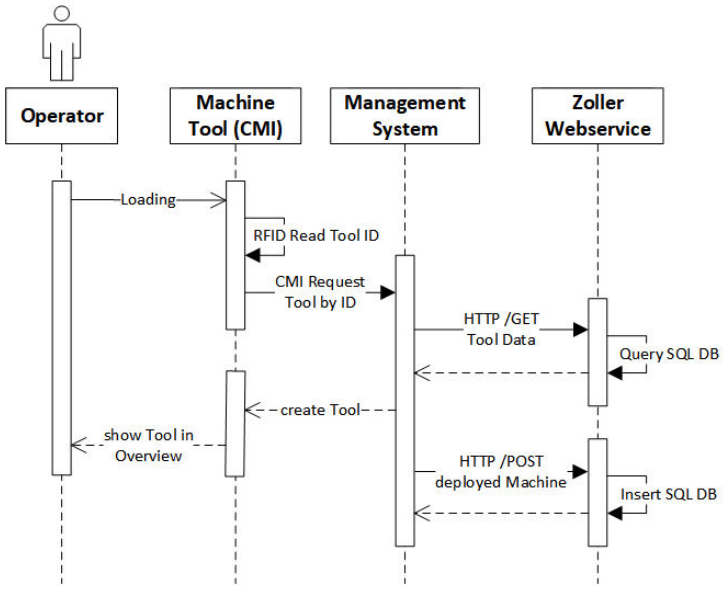

Figure 2.8: Realized communication with proprietary interface; source: [FTP19]

and the Zoller webservice, an HTTP webservice for connecting to the backend TMS database. On the machine tool side, it is was necessary to use some vendor specific interfaces to fill the gap. To en-/decode the data on RFID tags and store them in the tool list, the TDI (Tool Data Information) Ident Connection from Siemens has been used. A reading and writing-terminal has been implemented on the presetting device and on the machine tool. To grant remote access to the machine tool internal tool data "Create MyInterface" by Siemens has been used on top of the machine tool control. As figure 2.8 published by the author and her colleagues [FTP19] shows, the machine tool read using the TDI the RFID tool identifier and requested the tool data from the Management system via CMI. The management system requested the data from the TMS webservice and converted it to the Sinumerik specific tool data format to create the tool in the machine control supported format. The Management system used HTTP to receive the tool instance data from the Zoller webservice. During machining, the CMI interface provided updates of tool lifetime each time a tool is put back into the magazine. This data will be returned to the TMS to be stored. With TDI and CMI on the machine side and the webservice on the TMS side the connectivity issues have been solved. For further details on how the linkage issues have been solved can be found in the CIRP publication [FTP19].

Transformed processed

With the now accessible data, the tool net requirement can be determined, and the tool cycle process can be restructured. Figure 2.9 highlights the process changes in green. The machine tool is now capable of accessing the tool data and the correction data for each tool by reading the tool inventory number from the RFID tag and requesting its data through the CMI interface. This remote information enables now tracking the true tool consumption for each order. After machining the remaining lifetime can be updated in the database in the same way. Determining the tool net requirement optimizes the usage of existing tools as they remaining lifetime could be exploit better. Making the tool net requirement feasible, reduces the number of tools to be assembled and lowers the cost of the tool supply. Now only tools, not available at the shop floor will be assembled. Assembling a smaller number of tools results in a time and cost saving for each tool supply order. In addition, the set-up time is reduced, as well as the necessary magazine load due to the better exploited tool lifetime. All those savings realized by the newly accessible

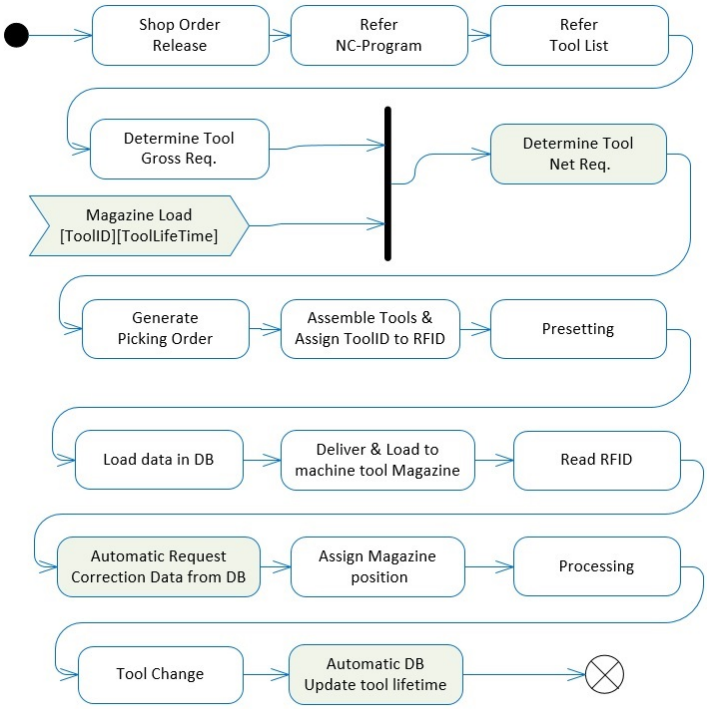

Figure 2.9: Activities of the transformed tool cycle; source: [FTP19]

data result in lower tooling cost. The here realized tool lifecycle develops new opportunities on a higher process level. For example, with the new data orders could be scheduled in an adequate sequence, which minimizes the tooling cost. Of course, this schedule must be managed in a flexible manner considering the current magazine load and react on newly arrived orders. This new possibility to schedule orders in a flexible manner based on the magazine load shall be further explored.
Based on the integrated tool cycle, the author envisaged the use of a multiagent system to schedule flexibly and autonomously orders in a sequence with minimal tooling costs. A multiagent system as flexible scheduler was considered as perfect fit. This schedule shall be achieved by utilizing price-based negotiations between the agents. The first pillar of this doctoral thesis addresses the designing of such a MAS for the integrated tool cycle. During the first design attempts the lack of design methodologies meeting the specific requirements of the manufacturing domain was discovered. The second pillar of this doctoral thesis addresses this issue by developing a MAS **Design Methodology for the Manufacturing Execution System Domain (DM-MESD**), which meets the specific domain requirements. The DM-MESD presented here, especially addresses mechanical and industrial engineers. It offers the designer, in contrast to already existing design methodologies, the possibility to evaluate if the design methodology is suitable for the desired application in advance.

2.2 Multiagent system

2.2.1 Agents and their characteristics

A multiagent system is a society of competing and cooperating agents. The characteristics of a multiagent system can be divided into three layers, the single agent, the agent's interactions, and the structure of the society. First the characteristics of a single agent are described and the

most popular internal agent architecture; Belief, Decision, and Intension (BDI) is introduced briefly. Wooldridge and Jennings defined an agent in 1995 [WJ95] as a software process with following characteristics:

Autonomy
Operates without human intervention and has the sovereignty over its actions and state

Social ability
Interacts with other agents

Reactivity
Perceives its environment and acting upon events

Proactiveness
Takes the initiative

As Giorgini and Henderson stated in their work, that the defining of agents is not straightforward since numerous interpretations of the agents' characteristics do exist [GHS05]. Although the characteristics of agents vary in the literature, a common consensus has been found on autonomy, situatedness, proactivity and sociality. An agent's response to its environment is called its behavior, which can be either proactive or reactive. A proactive agent acts without an external order to do so, while a reactive agent only responds to its environment. According to Giorgini and Henderson, most designed agents are a combination of both agent types.

Proactiveness and autonomy are based on interactions with the environment. A reactive agent only proceeds with its behavior if a change in the environment occurs. Agents sense systems or environmental changes as occurring events. An autonomous agent is independent as it decides how to respond to an incoming communication. Deciding which partial behavior should be proactive or rather reactive, is the designer's challenge. According to Giorgini and Henderson, agents are often compared to objects. An agent perceives its environment by using sensors or interfaces to resources. An agent action is defined as an environmental change or as a change in the agent's state, caused by the agent.

Jennings [Jen00] set encapsulation and situatedness in relation: „an agent is an encapsulated computer system that is situated in some environment and that is capable of flexible autonomous action in that environment in order to meet its design objectives". Some other agent's capabilities, such as mobility, for example cause dissent in the community. There are similarities between objects and agents, and this perspective has been strengthened by the many object-oriented design methodologies adapted for designing multiagent systems. According to Giorgini and Henderson, this combing perspective results in "clever objects" and "objects that can say no". An Agent's decision-making is based on a "perceive-decide-act" cycle. The interaction between agent in an environment is based on the notation of perception and action the agent communication, that has been used. Such a notation is the basis for the negotiation between agents in the form of contract, auction, and request protocols.

The two challenges from the software engineering perspective are firstly that the agent types are not predefined during the designing phase, which requires an iteration. The second challenge is that considering agents instance arriving and leaving is complex.

The best-known agent architecture "Beliefs, Desire and Intention" has been published by Rao and Georgeff in 1995 [RG+95]. Those three concepts are characteristic for a higher-level of abstraction and will be mapped in the design layer. Beliefs reflect the depth of the agent's environmental

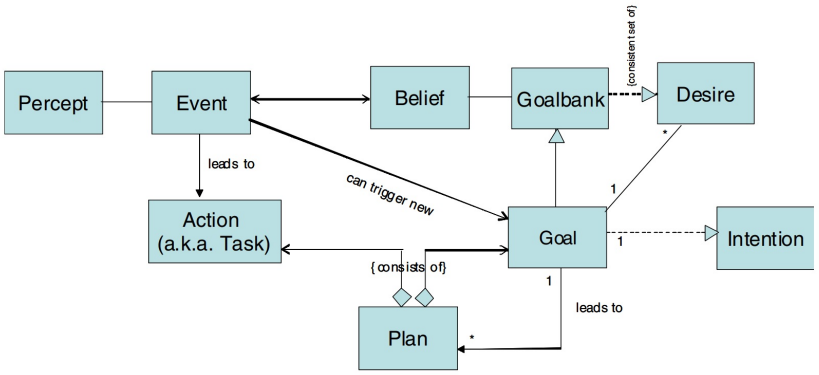

Figure 2.10: BDI model setting the concepts into relation; Source:[HSTD05]

sense, while desires reflect the different objectives, which might include contradictions. Intentions on the other hand display a coherent set of goals with no conflicts. Henderson et al have published a model in 2005 (see figure 2.10) displaying the relationship of BDI concepts [HSTD05]. Each goal leads to at least one plan, which consists of an action and a pursuing goal. An agent senses its environment through occurring events, therefore an event might lead to an action. Events also affect the agent's perception and can have a mutual effect on its beliefs and vice versa. Events also trigger the new establishment or adaption of goals. An agent has a set of goals or objectives. A goal without contradiction is, as the figure shows, an intention, which is inherited from the goal. While the relationship between goals and desires is more complex, a set of goals exist as inheritance of the class goal. Those sets of goals form the agent's desire, as far as they are consistent.

The BDI model describes the internal structure of one agent, not the interaction of an agent society. Therefore, methodologies for agent-oriented design can be divided into those methodologies designing multiagent systems and those designing solitaire agents. Jennings and Wooldridge [WJ95] stated that the **coordination** between agents might be difficult to achieve, due to the following impediment:

Agents actions may interfere
: Two robots reaching for the same workpiece, for instance, will collide

There might be global constraints to be met
: Especially in manufacturing, schedules must be kept and might cause coordination issues

No individual agent has the power over all resources
: This requires a successful coordination of several agents to maintain the required process flow

The interaction of agents utilizes protocols and exchange mechanisms, such as negotiations. Negotiation is seen by Jennings and Wooldridge as a joint decision between at least two parties, where first contradictory demands are made to then find a consensus [WJ95]. There are several negotiation templates inspired by human negotiations, the most prominent are auctions. Auctions are a trade mechanism to exchange goods or services, and can be differentiated into three types, one-sided, two-sided, and continuous auctions. In a one-sided auction, only one-party bids, while the other party accepts or denies the offer. In a two-sided auction an agent places bids to an auctioneer, who requests and then matches those bids to accomplish several trades. And then

there is the continuous auction, which allows bids and requests to arrive over time (and matches both over time). Based on these auction types, several auction protocols have been established. The three most known auction types, the English auction, the continuous double auction (CDA) and contract-net protocol (CNP) are reviewed briefly.

English auction
: is a one-sided auction type, the auctioneer aims to sell at the highest price and requests bids, of which it chooses the best offer.

Continuous double auction (CDA)
: is a mix of the two-sided and continuous auction, bids and requests were continuously made over time and matched by neutral auctioneers according to predefined market rules.

Contract-net protocol (CNP)
: a protocol that assigns tasks to individual nodes. It assumes, that there are several parties capable of fulfilling a task. Parties offering a task are called "manager" and the party requiring the task is called "contractor".
This protocol is initiated by the manager, who announces its task to possible contractors. The contractors are placing their bids and the manager chooses the best bid and sends the chosen one an award message.

Agents possess higher-level characteristics that evolve from plainer characteristics. Scalability, adaptiveness, flexibility, and maintainability for example, are such higher-level characteristics, that root in autonomy and proactiveness. Those characteristics are attributed to a society of agents (MAS). Scalability is a system characteristic, which enables the system to raise or minimize its capacity by raising or reducing its resources. A MAS managing a production schedule could expand its capacity by deploying more agent instances of the same types. Adaptiveness is based on the systems capability for autonomy, and proactiveness, but also on its scalability and reaction to a changing circumstance. An adaptive MAS is capable of coping or contributing to environment and circumstance changes. Incase of a production scheduling for example, scalability describes the system's capability to increase and decrease its capacity. From the throughput perspective, the system is adaptable, if it copes with the raised demand, although it is limited by the production limits. This might be different when it comes to the ability to adapt to a new product, workplan, or manufacturing technology. The flexibility of the system is measured by its range of adaptiveness to expected and unexpected circumstances. The higher the amount of adapting strategies a system has, the more flexible it reacts if one or more changes occur. A system's flexibility is defined by its ability to bend to the changing circumstances by considering and adapting to them. While those characteristics are desired, autonomy and proactiveness are also the fundament of the not desired emergent behavior. Emergent behavior is seen as a wildcard of the system's reactions to unforeseen or emergent events, which cannot be predicted. According to Jennings et al. [Jen00], this unpredictability leads to two key arguments against MAS:

- The patterns and outcomes of the interactions are inherently unpredictable.
- Predicting the behavior of the overall system based on its constituent components is extremely difficult.

The lack of the system's predictability signals, that the whole might be more than the sum of its parts. This means that a system can not be predicted by inspecting the individual parts. As

a further consequence, a bottom up perspective is not descriptive of the system's reaction. This circumstance is supported by the development of top-down design methodologies, such as Adelfe or Gaia. Reasons for this emergent and unpredictable behavior is the combination of two agent characteristics on an individual level (autonomy and proactiveness) and the complexity and mutual interference of the system at runtime. Agents decide at run-time what action to take next for each specific context. Although acquaintances between them are designed and known, neither the sequence in which those interactions might occur, nor their success can be ensured before the run time, which causes unexpected collective behavior [Jen00]. Those numerous factors impede predictions, which are aggravated by the de-coupling and variability. Even if agents make requests, due to their autonomy they might be immediately accepted, modified, or even refused.

Concepts

Agent based approaches are distinguished not only by the above-mentioned characteristics, but also by the concepts they implicate. Such a concept is **encapsulation**, which is the capability for self-containment. According to Shen et al. [SHYN06], encapsulation can be realized either by functional or by physical decomposition. According to the functional decomposition approach, agents can encapsulate functional modules. In manufacturing, such functional modules could be transportation, material handling, scheduling, quality control, etc. This encapsulation approach emphasizes, that the designed agents have no explicit relationship, or interface to their physical entities and assets they represent. This loose relationship of agents with their physical entity enables the integration of agent-approaches into already existing systems and resolves the legacy issues. The physical decomposition approach on the other hand binds the agents as cyber-physical representations to their assets. The agents possess an explicit relationship to their physical entity.

Agents society offer the concept of three **organizational structures**, the hierarchical, federation or the autonomous approach. The hierarchical organization structure is similar to today's manufacturing state. Although it is criticized for its centralized architecture some MAS projects still have realized it, such as the international Holonic Manufacturing System (HMS) project [Dee01]. The federation organizational structure is built on the agents' ability to coordinate, interact, and cooperate. The federation approach has three popular approaches build on either, the facilitator, the broker, or the mediator. The facilitator approach is based on an intermedium (the facilitator) enabling interactions between local and remote agents. The facilitator fulfills two services, the routing of messages to their proper destination and the translation of incoming messages so that the recipient can process them. Such a destination can be a group of agents. The broker approach is like the facilitator approach, but with the difference that the broker is responsible of finding an agent which will complete the task. While the facilitator only forwards the messages to their destination, a broker must find a suitable contractor to fulfill the task. In its role the broker monitors and notifies the process from the request to its task fulfillment. The mediator approach extends the services of the broker as it coordinates and stimulates cooperation between the agents. It also learns from the single agent's behavior and takes it into account for future coordination. Overall, it can be said, that the federation architecture provides a stabile organizational structure coordinating activities and optimizing resource allocation. This agent organizational structure supports scalable MAS designed with the functional decomposition approach. The autonomous agent organization approach differs from both above-mentioned approaches as it does not include any coordination mechanism. This structural approach rests on the individuality and autonomy of each agent instance. An autonomous agent communicates and interacts directly with other agents in its own way, achieving its own goals. An autonomous agent

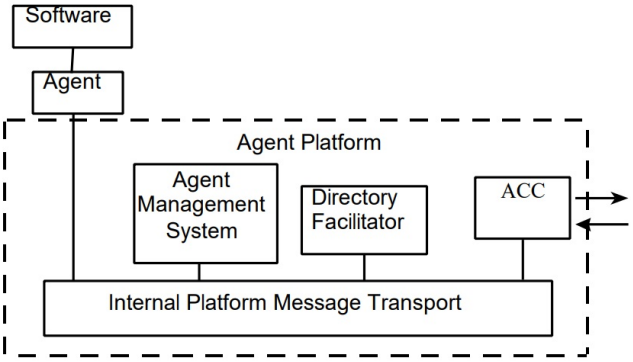

Figure 1 – FIPA reference model of an Agent Platform

Figure 2.11: FIPA specification reference model, source: [BPR99]

possesses knowledge of its environment either through sensors or provided protocols. Beside its high grade of flexibility, scalability, and adaptability the autonomous agent bears the risk of emergent behavior, especially for a high number of agents.

Another concept associated with agents, although it has its origin in the Artificial Intelligence (AI) area, is the concept of **learning**. The capability of a software entity (agent or not) to learn is a complex and charged topic in computer science. Agents with the capability to learn could essentially benefit the manufacturing domain. A learning agent could observe its environment and extend its knowledge to adapt its reactions. The mechanisms that would support this process are not subject of this doctoral thesis as they are issues of computer science.

The next concept offered by agent-based approaches is **simulation**. With emergent behavior and the manufacturing domain in mind, simulations to estimate collective behavior become very important. However, agent-based systems can also be a way to simulate groups of different interests and their mutual influence, as they can be mapped by software agents. Such MAS, for example, one that maps a market with buyers and sellers or any other society, can be used to simulate the collective behavior as an answer to environment changes.

2.2.2 FIPA Specification

The Foundation for Intelligent Physical Agents (FIPA) was founded in 1996 with the aim of providing suitable standards that would ease the development of agents. Since in 2005 FIPA was reincorporated into IEEE as a standards activity. The focus was on specifying the cooperation of different agent platforms, by offering a standardized agent model and communication language. The FIPA Specification offers a reference model of an agent platform (see figure 2.11). This reference model defines key agents necessary for the management of the platform and its agents. The reference model also specifies the agent management content language and ontology [BPR99]. Bellifemine et al. [BCG07] summarized the four core concepts of the FIPA, the agent communication, the FIPA sublayer, the agent management and the abstract architecture as it follows in this subsection:
The agent communication in the FIPA is based on the Agent Communication Language (ACL). The ACL refers to the speech act theory by stating that messages represent speech acts or performatives (communicative acts). The FIPA-ACL includes a set of 22 communicative acts. Each communicative act defines the effects on the attitude of the sender and receiver agent. The approach of communication acts is compatible with the BDI. The FIPA-ACL uses mostly

communications acts to inform, request, agree, not understood, and refuse. The specification states, that to be granted as fully compliant, receiving any FIPA-ACL communicative act, and at least answering it with "not-understood" is required.
The FIPA communication stack is separated into the seven FIPA sublayers [BCG07]. Those sublayers are built within the application layer of the classical OSI or TCP/IP stack.

Transport
: Is the lowest application sublayer protocol, FIPA has message transport protocols for IIOP, WAP and HTTP.

Encoding
: defined message representations for higher-level data structures including XML, String and BitEfficient.

Messaging
: The message structure is independent of encoding to encourage flexibility, key parameters (sender, receiver, message type, time-outs. etc.) if necessary.

Ontology
: The individual terms of the message content reference an application-specific conceptual model or ontology.

Content expression
: The content can be of any form, but guidelines for the use of general logical formulas, predicates, algebraic operations for combining and selecting concepts exist. Examples of logic formula include: "not", "or", "implies", "equiv", etc. Examples of algebraic operators include "any" and "all".

Communicative act
: It implies a simple classification of a message in terms of actions, or performatives. Examples include "inform", "request" and "agree".

Interaction protocol or IP
: It defines several interaction protocols.

Another key concept of the FIPA Specification is the Agent Management. It is a normative framework, in which agents can exist, operate, and be managed. It offers the logical reference model for the creation, registration, location, communication, migration and operation of agents (see [fIPA03]). Its reference model (figure2.12) includes the components building the framework [BCG07], which are :

Agent Platform (AP)
: The Agent Platform is the overall infrastructure in which the agents are deployed. The AP includes the machines, operating systems, FIPA agent management components (agent, DF, AMS, MTS), the agents themselves and any additional support software.

Agent
: An agent is a computational process, that inhabits an AP and offers computational services. Those services can be published as a service description. The design of those services is not subject of FIPA. FIPA only defines the structure and encoding of messages used to exchange information between agents. An agent can be registered at several transport addresses at which it can be contacted.

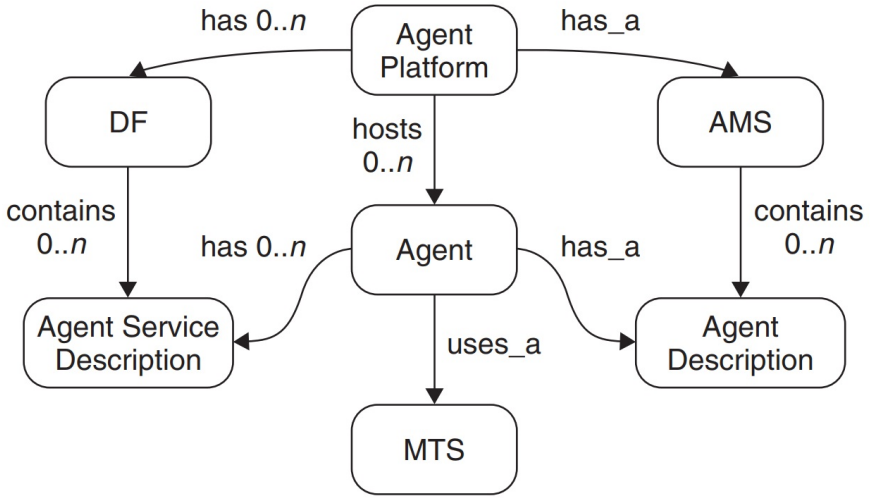

Figure 2.12: Agent Management Ontology from the FIPA specification, source: [BCG07]

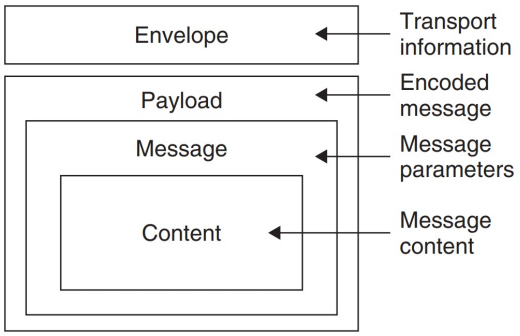

Figure 2.13: Message structure from the FIPA specification, source: [BCG07]

Directory Facilitator (DF)

The DF is an optional component of an AP. Like the yellow pages it provides a maintained and accurate list of all agents at any given time. Any agent wishing to publicize its services to other agents must find an appropriate DF and request the registration of its agent description. The DF fulfills therefore the role of the previously described broker.

Agent Management System (AMS)

The AMS is mandatory as it is responsible for managing the operation of an AP, such as the creation and deletion of agents, and overseeing the migration of agents. Each agent must register with an AMS to obtain an Agent Identifier (AID). The AMS uses the Agent Identifier as a directory of all agents present within the AP and their current state (e.g. active, suspended or waiting). The life of an agent terminates with its deregistration from the AMS.

Message Transport Service (MTS)

The MTS is a service. It transports FIPAACL messages between agents on any given AP and between agents on different APs. The general structure of a FIPA-compliant message is depicted in Figure 2.13. Messages are providing a transport envelope that comprises the set of parameters detailing, for example, to whom the message is to be sent.

Bellifemine [BCG07] summarized the key achievements of FIPA as it follows:

- Specifications supporting inter-agent communication
- Key middleware services
- Highly used agent communication language
- A set of key interaction protocols
- Available FIPA compliant open source tool-kits
- FIPA compliant agent application platforms
- Agent-specific extension of UML, AUML

Several FIPA compliant agent platforms were developed. Three well-known agent platforms are: JADE (`http://jade.tilab.com/`), FIPA-OS (`http://fipa-os.sourceforge.net/`) and ZEUS (`http://more.btexact.com/projects/agents/zeus/`). Due to its popularity, its high range of study material and its vivid community, JADE was chosen in this thesis. In the next subsection gives a short introduction to JADE. Further information about JADE can be found on the website: JADE (`http://jade.tilab.com/`).

2.2.3 JADE

The Java Agent Development Framework (JADE) was developed by Telecom Italia and is an open-source software, with an active community. JADE provides an implementation of the FIPA-ACL. As intended in the FIPA-ACL, JADE also supports XML-based syntax and offers an XML-codec as an add-on. JADE simplifies the development of MAS while it ensures FIPA compliance through a comprehensive set of system services and agents. JADE offers the programmer a FIPA-compliant Agent Platform including includes the AMS, the DF and the ACC (Agent Communication Channel). The communication between agents on the same host is based on Java events, but it also offers multi-domain applications by the starting of several DFs (Directory Facilitator) at once. JADE has an extended library of FIPA interaction protocols ready to be used, and offers a graphical user interface (GUI). Its GUI enables managing and monitoring several agent platforms. The core of the JADE software architecture (see figure 2.14) is the coexisting Java Virtual Machines (VM). Between those VMs the communication is based on Java RMI (Remote Method Invocation). Thereby each VM is a basic container of agents. A container provides a complete run time environment for concurrently executing the included agents. Each agent container is a multithreaded execution environment, which composes one thread for each agent and additional system threads spawned by RMI runtime system for dispatching messages[BPR99]. To model the tasks that an agent shall perform, JADE uses behavior abstractions. JADE offers behavior classes and agents instantiate their behaviors according to the needs. From a programming point of view an agent is an active object with a thread of control. JADE uses a thread-per-agent concurrency model instead of a thread-per-behavior model [BPR99]. So that, two behaviors of the same agent are scheduled cooperatively, although different agents run in the multithreaded environment. The behavior class model of JADE is depicted in figure 2.15, published by [BPR99].

2.2.4 MAS implementations and demonstrators

There multiple ways to classify MAS application in the manufacturing domain. The MAS can be sorted by the agent internal architecture, by their organizational structure, or by the platform environment they use. The MAS applications presented here are limited to those applied in the

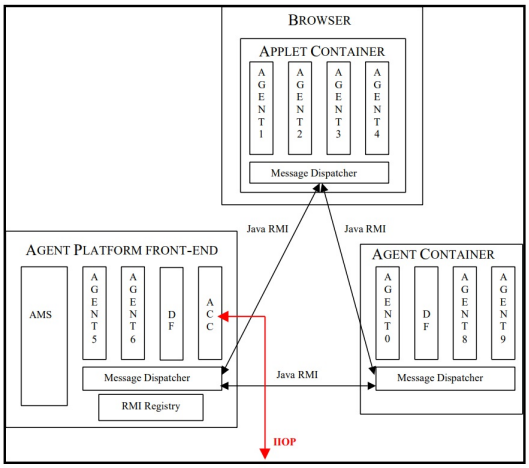

Figure 2.14: Software architecture of JADE, source: [BPR99]

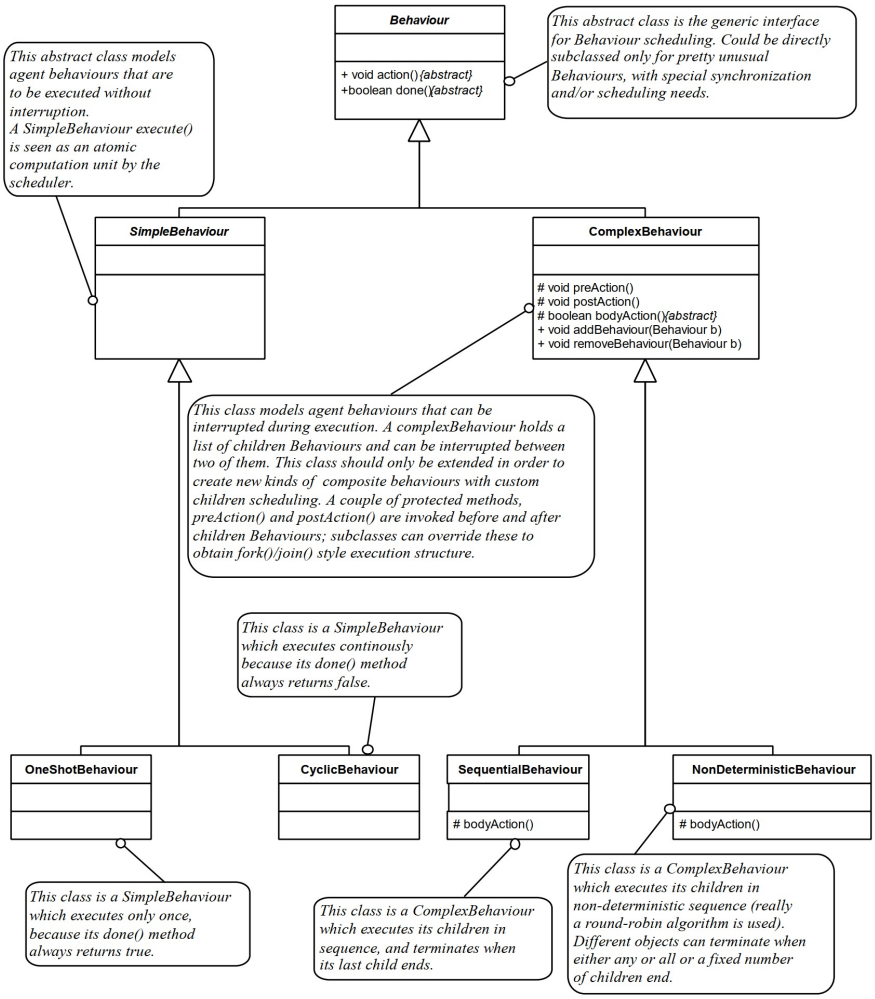

Figure 2.15: JADE behavior class diagram, source: [BPR99]

manufacturing domain. Thereby the focus is set on functional areas relatable to the investigation area, such as production planning and scheduling. In this subsection reviews some of the most fitting projects. Manufacturing planning and scheduling foresees and plans the production demand with the aim of doing so in the most profitable way. In the Gerstner Laboratory EXPlanTech, a production planning MAS has been deployed (see [PVB05]). Its major tasks were data collection, monitoring, and resource allocation by agent negotiations. These negotiations tried to find the best fit for projects with different deadlines and priorities [PM08]. This MAS was designed to include distributed shop floor data. ExPlanTech was successfully integrated into the ERP system of a cast and molds manufacturer. Its range of functions was expanded, so that it could cover some functions of the supply chain.
Another project in this area was developed by Saarstahl AG and DFKI GmbH is the AgentSteel System, a MAS for planning and scheduling the steel production. AgentSteel was fully integrated into the Saarstahl IT infrastructure and utilizes webservices to provide an enterprise external SOA (see [JMMLSF05]). A realization of an enterprise scheduler based on multiagents was developed for Axion-Holding, Russia by Andreev et al. [AIST10]. The authors developed a MAS based on three agent types. The first agent type was the decision maker. Those agents handled and participate in real world negotiations. The second type were the agents of unanimated objects, which represent equipment and orders capable of acting as independent entities. This type of agents had its own objectives and constraints to which it was bound. The last type of agents was called "group" agents. Those agents represent departments or units of the production. The authors combined a set of those group agents to a "swarm". Each swarm representing a production department had its own multiagent scheduler.

Another MAS project, which was deployed in a similar application area to the investigation area, is the MAS designed for the KUZNETSOV corporation [SSS+13]. KUZNETSOV is a large Russian national aircraft jet engines manufacturer, which sought a cooperation with Smart Solutions, Ltd to design a MAS scheduler. Thereby Shpilevoy et al. designed and developed software agents representing the interests of orders, resources, workers, machines, operations, and materials, using relationships between the operations [SSS+13]. The overall system architecture is displayed in figure 2.16, including the client layer, the server side (MAS) and the utilized data storage layer. The core scheduling is handled in the scheduling engine. It includes the agent platform and provides therefore the following functionalities:

- Agent dispatcher
- Messaging services
- Agent lifecycle support
- Agent creation
- Agent termination
- Communication protocols
- Representing the resulting schedule

The MAS responded fast to new events and proactively improved the operation plans. This was realized by optimizing the use of free machine time or worker slots, by the sequence of shifts and reallocations of previously scheduled operations to other resources. This optimization was achieved by changing "unstable equilibriums" for a "stable equilibriums" measured by the bonuses and penalties of a virtual agent market. According to the authors its architecture is very

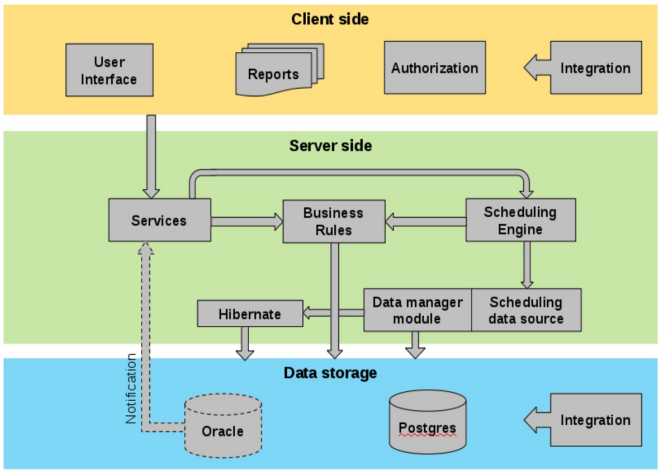

Figure 2.16: System architecture, source: [SSS+13]

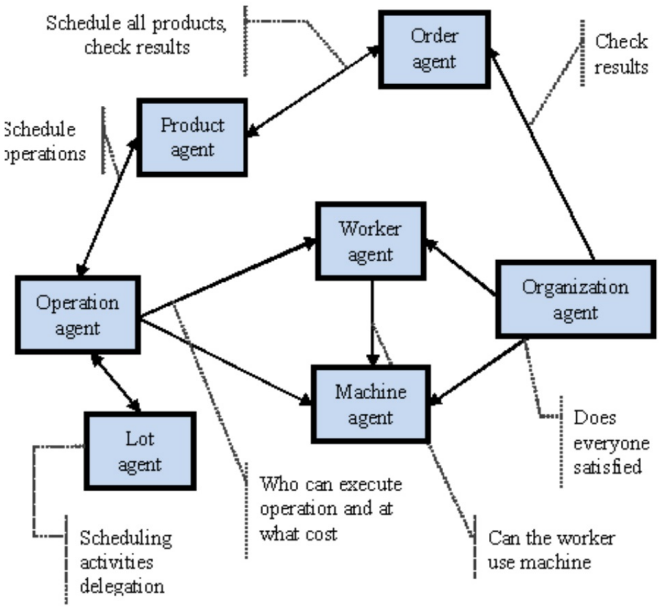

Figure 2.17: Main communication lines between the agents types, source: [SSS+13]

close to the PROSA architecture. The MAS consisted of at least eight agent classes. **Order Agent**, had cost, priority, deadlines and production details as its attributes. This agent aimed to achieve the best possible allocation for itself. The **Organization Agent**, which tried to improve given KPI and monitors and harmonizes the process. The **Worker Agent** tried to get as many jobs as possible, to raise its possible bonuses, achieved for quality and performance. The **Machine Agent** sought an optimal machine utilization. The **Technological Process Agent** coordinated technical operations. The **Technological Operation Agent** worked as a facilitator, with the aim of finding the best possible worker or machine. The **Operations Lot Agent** aimed to improve the efficiency by consolidating operations if possible. The MAS had a class of agents called the **Other Agents**, which function as placeholders for required entity representations, such as materials, instruments, transporter, etc. The communications between those agent classes can be seen in figure 2.17.

Due to its structure and its marketplace strategy this MAS project realized the following

advantages during its first testing:

- An increase of workshop productivity from 10 up to 15%
- A reduction of scheduling efforts
- An increase of the efficiency of resources up to 15%
- A reduction of the time of response to unexpected events
- An increase of the completed orders, within a measured timeframe from 15 up to 30%

Besides those MAS projects, which are clearly located in the manufacturing planning and scheduling domain, there are projects with a wider functional range, such as PABADIS, SOCRADES, and IMC-AESOP. All three projects tackled functions from automation up to enterprise planning and control[LCK16].

2.3 Design methodologies

2.3.1 Common design methodologies for multiagent systems

To give the reader an outline of the common MAS design methodology, figure 2.18 displays Giorgini and Hendersons's graphical overview of design methodologies [GHS05]. It shows the different sources from which design methodologies evolved, focusing (beside Tropos) on methodologies with roots or techniques from the object-oriented approach. Only the commonly known design methodologies for multiagent systems or agent-software are considered. Emergent behavior impedes the system's predictability since inspecting its single parts no longer provides an insight of the system's reaction. It requires top-down design methodologies, such as Adelfe and Gaia to address emerging behavior. Besides those two top-down design approaches, various other design methodologies do exist, which Wooldridge, Jennings and Kenny [WJK00] classified in four groups:

Those using existing object-oriented modeling techniques or methodologies
as their basis seeking to either extend and adapt the models or to define a methodology for their use.

Those with origins in knowledge engineering
Which are based on knowledge methodologies and modeling techniques. Those methodologies are suitable to verify the system structure and function.

Those which include existing formal methods and languages
and provide definition within a framework that supports agents or agents systems.

Those which have been developed completely new for specific agents
such as CASSIOPEIA.

The possibilities to classify design methodologies are vast. One-way of classification is to divide design methodologies into those that use a role concept to map the agents' interactions, such as Tropos, Gaia and PASSI and those that do not. Agent-oriented design methodologies can also be divided into those that consider an agent as a solitary entity with focus on the BDI architecture, and those which prioritize designing an agent society. The latter mimics human

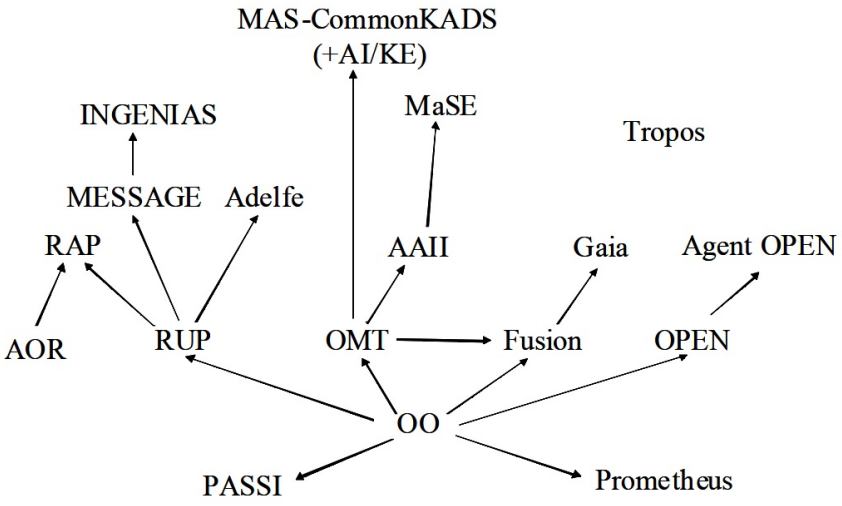

Figure 2.18: Landscape of agent-oriented design methodologies and their influence, source: [GHS05]

organization structures by utilizing roles to design a MAS, such as Tropos, MAS-common KADS etc. One of the core objectives of a MAS design methodology is to define the agents' interactions and cooperation. MAS design methodologies also differ by those which are platform independent and do not consider the implementation (Gaia) and those which offer extended tools and methods to turn models and designs into a code skeleton (PASSI). There are similarities between object-oriented and agent-oriented approaches. It is therefore not surprising that many MAS design methodologies, besides Tropos, have object-oriented roots (see figure 2.18). There are six branches, evolved from the object-oriented paradigm according to Giorgini and Henderson [GHS05]: PASSI, which will be described later in detail, RUP (1999), OMT (Rumbaugh 1991), Fusion (Coleman 1994), OPEN and Prometheus. While some methodologies such as PASSI gained their familiarity with the object-oriented approach by utilizing object modeling language (UML), others like MAS-CommonKADS are rather AI-based yet claim to be influenced by object-oriented approaches.

In 1991 Rumbaugh et al [RBP+91] introduced the **Object Modeling Technique (OMT)**, which is based on object-oriented analysis and design. This methodology produces three models to capture a different perspective:

Object model
: which is a structural description of objects, including identity, relationships, attributes, and operations.

Dynamic model
: which maps the sequence and timing of operations, especially events, sequences of events and states.

Functional model
: Which maps the transformation of values, functions, constraints, and functional dependencies.

From OMT, two branches of design methodologies developed leading to MAS-CommonKADS on the one side and MaSE on the other side. Both branches are also influenced by knowledge-oriented methodologies. The **Multiagent System Engineering (MaSE)** derives roles from

goals [DeL99]. As most MAS design methodologies, MaSE has an analysis and a design phase. During the first phase, goals will be captured by use cases and a goal hierarchy. In the design phase, roles are assigned to agent classes. Then interaction protocols are defined. After that, the internal agent architecture is specified as a last step, the deployment of the agents will be addressed. Beside this structured flow, MaSE does not specify criteria to complete the single steps. For the interaction MaSE offers a protocol specification model, but no method to design the protocols.

MAS-CommonKADS is an extension of the CommonKADS methodology. Both methodologies originated from OMT and extended its techniques to map agent-oriented aspects. MAS-CommonKADS [IGGV97] starts with a conceptualization phase, to capture the requirements by utilizing five modeling phases:

1. **Agent model**
   - Identifies actors through use cases
   - Identifies active objects
   - Assigns entities to agents
   - Offers initial task model to identify functions to be assigned to agents
   - Uses of an agent by other agents
2. **Task model**
   Splitting tasks and mapping goals, inputs, and outputs to fulfill the tasks
3. **Co-ordination model**
   Assigns interaction links by building prototypes and analyses their interaction in the following steps:
   - Prototypes of interaction scenarios
   - Events or messages in a flow diagram
   - Data exchange for each interaction
   - Finalize each interaction in a state-transition diagram
   - Identify the synchronization type for each interaction
4. **Knowledge model**
   Maps knowledge in form of an ontology
5. **Organization model**
   Displays structure of the agent society

MAS-CommonKADS provides several options to identify agents, such as linguistic case analysis, use cases or conceptual analysis. However, it lacks prescriptiveness and guidelines when it comes to implementing the steps. It can therefore be said, that MAS-CommonKADS addresses experienced MAS designers, rather than industrial engineers.

Tropos has no direct link to object-oriented approaches, as it is a system-oriented methodology [BPG+04]. It connects models of single concepts, such as goals, tasks, roles, responsibilities, interactions, and organization to each other, like Prometheus. It begins by listing requirements through modeling goals, tasks, resources, actors, and dependencies. Its next phase is the design phase, in which actors in the system are refined, goal-fulfilling capabilities are identified and those concepts are assigned to agents. Tropos does not offer pattern to identify the above-mentioned

	Dynamic Models	*Structural Overview Models*	*Entity Descriptors*
System Specification	Scenarios	Goals	Functionalities actions & percepts
Architectural Design	(interaction diagrams) Interaction Protocols	(coupling diagram) (agent acquaintance) System Overview	Agents Messages
Detailed Design	Process Diagrams	Agent Overview Capability Overview	Capabilities Plans, Data, Events

Figure 2.19: Classification of Prometheus models, source: [PW02]

concepts. It also lacks prescriptive methods, according to Bussmann, Jennings and Wooldridge [BJW13].

Prometheus [PW02] addresses industry professionals and undergraduate students, with no background in agent technology [PW02]. It has three phases: the system specification, the architectural design, and the detail design. Prometheus uses scenarios instead of roles; besides that, it falls back on concepts, such as goals, functionalities, actions, and perceptions. Its major models are either dynamic, structural overview models or entity descriptors (see figure 2.19). The system specification phase specifies the system by using goals and scenarios as concepts. The next phase is the architectural design phase, starting by identifying agents and capturing the system's overall structure in a diagram. Afterwards, scenarios will be transformed into interaction protocols. The last phase is the detail design, which addresses the internal design of each agent and therefore defines capabilities, data, events, and plans. Prometheus stands out as it provides a detail guidance on how to group functionalities (see [BJW13]):

- **Group**, if functionalities are using the same data
- **Group**, if interaction can be reduced
- **Do not group**, if data should be available for security or privacy reasons
- **Do not group**, if functionalities will be modified by another party

However, it must be stated, that resting the grouping criteria only on data and interaction pairing is a good start, but not enough to capture dependencies in the production. Data for example can be used by different functions, without the necessity of combining them. The authors stated that Prometheus can be used by undergraduate students and industry practitioners, which meets the domain specific requirement. The authors used Prometheus to ease the designing and implementing of multiagent systems in JACK for their computer science students. It utilizes UML with specific extensions developed for the methodology and demands from the designer to have UML modeling skills. Prometheus is however, built upon JACK and therefore not platform independent.

DACS is a design methodology for agent-based production control [BJW13], and shall meet the specific requirement of the manufacturing domain. It addresses mainly control engineers with no or minor knowledge about multiagent systems. Bussmann tested his methodology on two computer science students and then reviewed their feedback (see [BJW13]). Beside the fact that both students did not have any multiagent knowledge before applying the methodology, computer scientists are not our target designers. The DACS methodology has three major steps. The first step is the analysis of decisions, then the identification of agents and finally, the selection of interaction protocols. During the analysis of decisions, necessary control decisions

are identified. By analyzing these decisions, dependencies between decisions can be derived. The author differentiates between effectoric decisions tasks and decision dependencies. Effectoric decisions tasks describe decision tasks that include at least one physical action, and has the following attributes: control interface, trigger (event), decision space and local decision rule. Decision dependencies reflect the mutual influence of decisions made. The analysis phase results in a list of effectoric decisions, a list of decision dependencies and a trigger diagram (Petri Net notation), including dependencies. The second phase identifies agents and designs, by iterating between the agent-based control, and the clustering of decision tasks. An agent is identified and assigned to decision tasks for which only itself is responsible. To find the right partition of decision tasks, Bussmann offers the following procedure (direct quotation [BJW13, p. 145]) :

1. Step
Start with the set of clusters in which each cluster consists of a single decision task.

2. Step
For each subset of clusters, apply every clustering rule to every combination of decision task from the different clusters.

3. Step
Whenever a clustering rule applies, replace the cluster by a new cluster consisting of all the decision tasks from these clusters.

4. Step
Repeat steps 2 and 3 until no more clusters can be combined.

Beside those steps Bussmann offers additional clustering rules, with the aim of finding partitions that would ease the implementation of the decision model. Those clustering rules are (direct quotation [BJW13, p. 147-149]):

Clustering rule 1: Interface cohesion
Decision tasks using the same control interface should be assigned to the same agent.

Clustering rule 2: State cohesion
Decision tasks changing or affecting the state of the same production component should be assigned to the same agent.

Clustering rule 3: High interaction coupling
Decision tasks that are always strongly dependent on each other should be assigned to the same agent

The last step of this phase is the improvement of the decision model and a possible iteration of the first step. The output of this phase is a list of agents.

The last phase of the DACS methodology is the selection of interaction protocols. An interaction is mainly specified in protocols. Such protocols define the types of messages and the sequence of messages. The first step is to classify each dependency by pre-defined criteria. Then those classifications are matched with the library to identify re-use potential. If this is not the case, then customized and specified protocols must be designed. The DACS methodology addresses the production control as application domain. In addition, it also addresses non-MAS specialists as designers, although it mainly focuses on process engineers. This methodology focuses on the process rather than on the objects, which is crucial in managing production processes. Furthermore, it offers clustering rules, although they are a bit vague. On the other hand, the design methodology utilizes Petri net-based modeling notations, which a tester from the book stated to be not a proper fit. The tester recommended UML. This methodology is rating decisions with a physical effect (effectoric decision) higher than other decision tasks. The necessity of design choice is debatable.

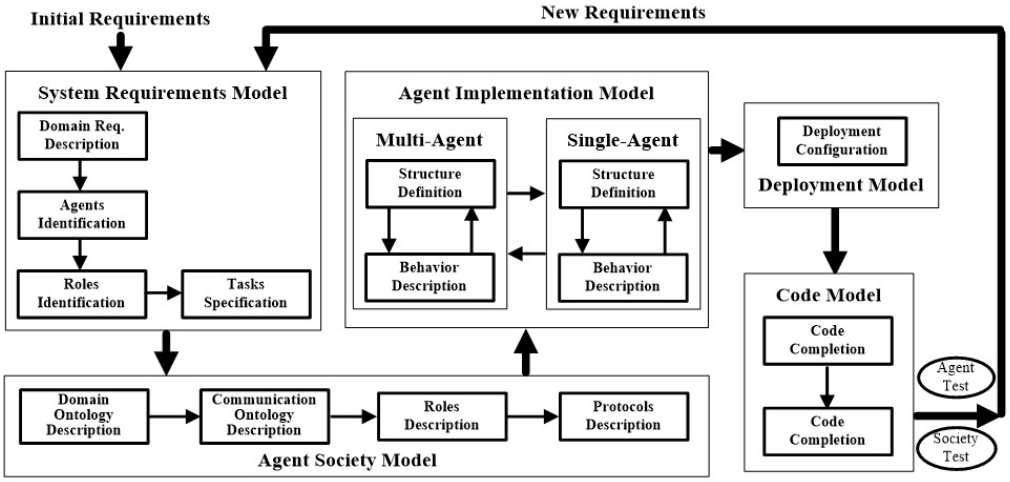

Figure 2.20: Stages of the PASSI Methodology, source: [BC02]

2.3.2 PASSI

PASSI, standing for "Process for Agent Societies Specification and Implementation". PASSI is a MAS design methodology focusing on turning system requirements into JADE code. Burrafato and Cossentino support the idea that the design of an agent society is made in a most natural manner [BC02]. According to Cossentino, PASSI is a step-by-step requirement to code method to develop multiagent systems. PASSI uses or refers to common languages or standards, such as Agent UML (AUML), FIPA, JAVA, RDF, and Rational Rose. As figure 2.20 reflects, the PASSI methodology consists out of five stages with their reflecting models:

1. **System requirements model**
   Analyzing stage, consisting of four substages: The domain description, the agent identification, the role identification and the task specification.

2. **Agent society model**
   Dependent on the role identification, this is the creative stage, consisting of the ontology description, the roles description and the protocol description.

3. **Agent implementation model**
   The iterative implementation phase, consisting of two mutual models, the agent behavior description and the agent structure definition

4. **Code model**
   It focuses on the code reusing and code completion.

5. **Deployment configuration**
   It deploys the designed MAS.

As in many design methodologies, PASSI starts with an analysis phase, the “system requirement model“, of which the the domain description phase (DD phase) is the first step. This phase is the equivalent to a classical requirement engineering approach, starting with a system analysis. The system analysis will be captured in several use case diagrams. The functionality of the system will be mapped as use case. If further detailing is required due to the complexity of the use case diagram, supporting sequence diagrams are suggested. Considering the tool cycle, a

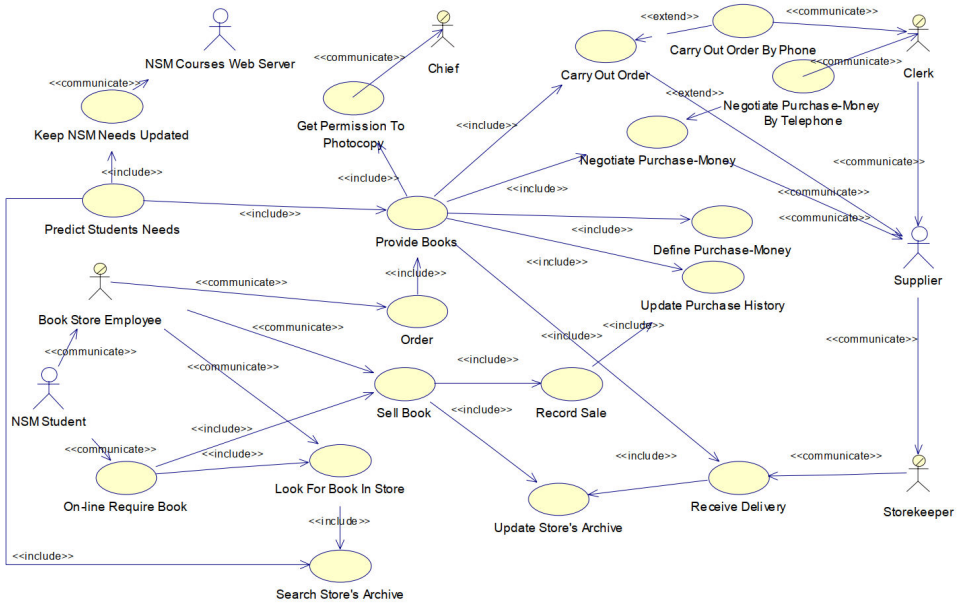

Figure 2.21: The domain description phase of the Moller Bokhandel case, source: [BC02]

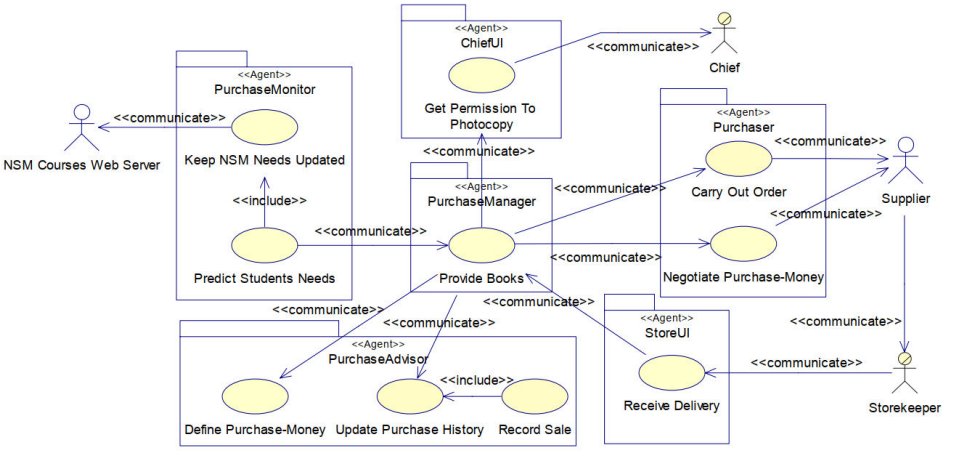

Figure 2.22: The agent identification of the Moller Bokhandel case, source: [BC02]

multitude of use case diagrams must be developed. Burrafato and Cossentino examined the PASSI methodology in their article [BC02] on the Juul Moller Bokhandel A/S, a book buying and selling bookstore. The author had chosen to examine only one possible scenario, which is not uncommon in scientific articles, but taking the extent of this single scenario into account, indicates the high effort of this methodology for more complex application processes.

In the next step, the agent identification phase starts by composing agents from the use cases. The authors grouped one or more use cases together into new diagrams, composing packages of functionalities. For the presented use case of the Bokhandel, the authors restructured the use case depicted in figure 2.21 into the role description diagram (see figure 2.22). Although one or multiple use cases are grouped into packages, which define functionalities of a specific agent, the authors have not explained any grouping mechanisms. The use cases “carry out of order“, and “negotiate purchase-money“, for example, could be grouped together with “receiving delivery“, or with other use cases.

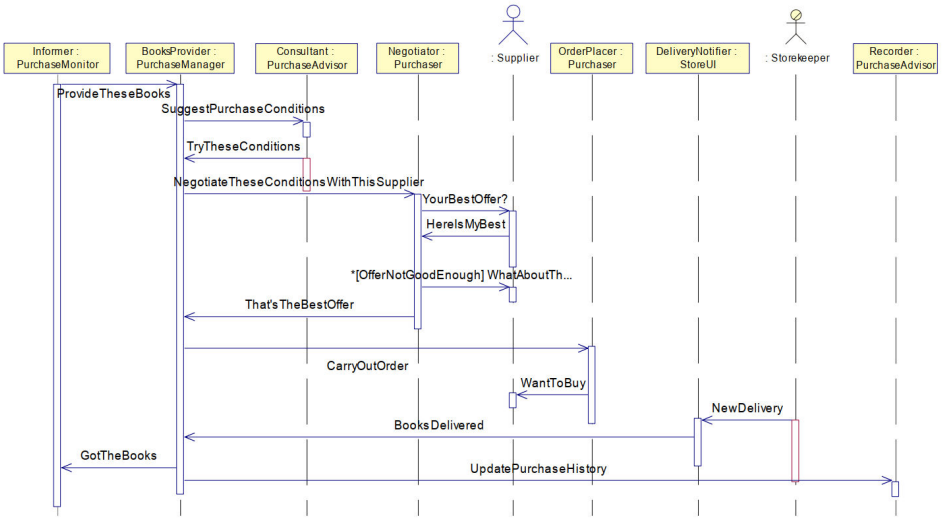

Figure 2.23: Role Identification for the scenario in which the Purchase Monitor announces the need for a books purchase, source: [BC02]

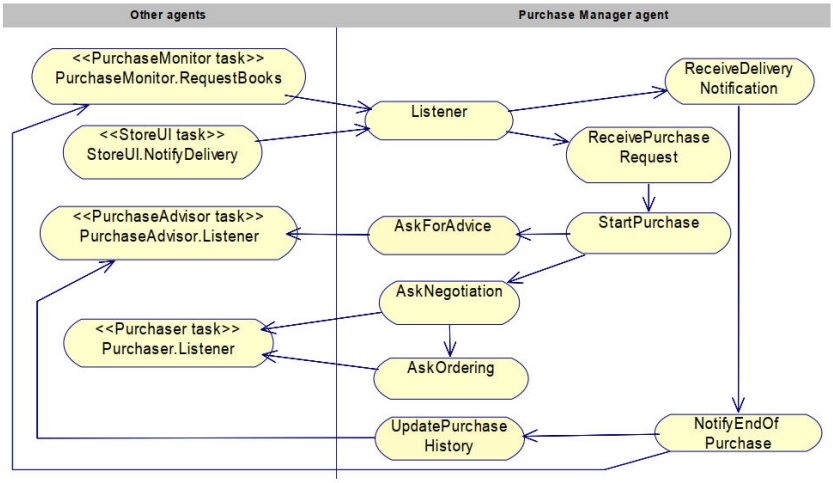

Figure 2.24: Task specification for the PurchaseManager, source: [BC02]

The next step is the role identification phase, which as a social concept is part of the agent society model, and the third part of the system requirement model. The authors defined roles as the sum of all possible "paths "of the agent identification. A path is seen as all possible scenarios of interactions between agents trying to achieve a behavior. The result of this step is a sequence diagram for each scenario (use case) enlisting all possible roles of an agent by the syntax of $< role_name >:< agent_name >$. Figure 2.23 displays such a scenario. In case of the Purchaser Agent two roles have been identified, one as Negotiator and one as OrderPlacer.

The next step is the task specification phase. The result of this phase are activity diagrams for each agent with two swim lanes, one on the right side, containing all activities brought by the agent and one on the left side with all activities required from other agents. The authors stated that the task specification diagram is a recapitulation of all tasks the agent can complete, ignoring the information about the roles of the agent. Observing the task specification diagram in figure 2.24, the reader might guess, that those described tasks could be future methods.

The next model is the agent society model containing the ontology description, the role description,

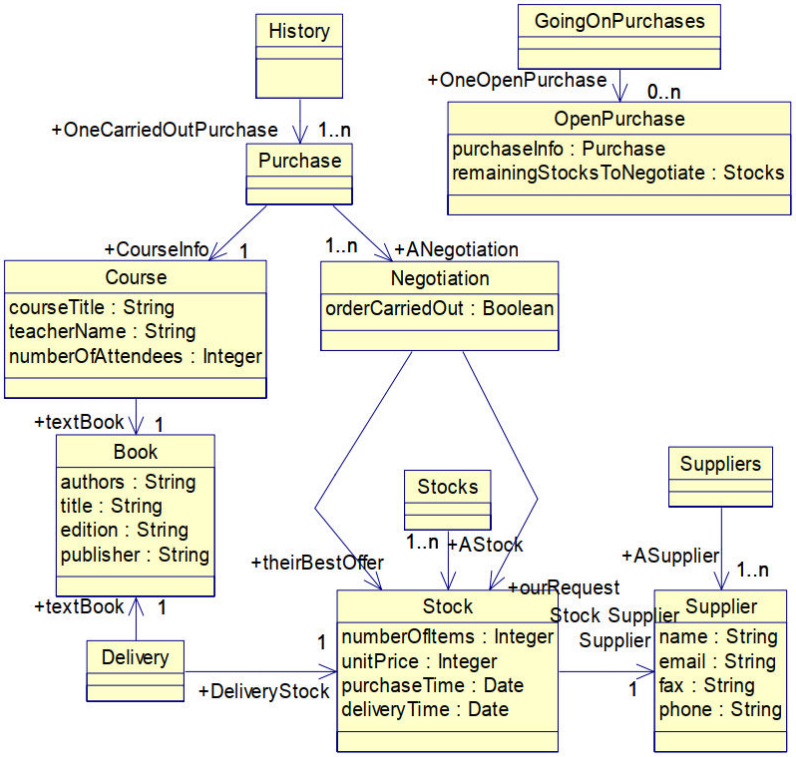

Figure 2.25: The domain ontology diagram describing system entities and their attributes in a class diagram, source: [BC02]

and the protocol description. The three next steps of the PASSI method will address the FIPA requirements for ontology and communication protocols. The next step is the ontology description phase, and it consists of two diagrams: the domain ontology description and the communication ontology description. The domain ontology description is represented in an XML scheme and in future work realized in resource description framework (RDF), as figure 2.25 shows. The second model, communication ontology description, addresses FIPA definition of communications, consisting of speech acts, that the FIPA has defined in several protocol types, categorized by their intention. The authors based their communication ontology diagram on the template structure provided by the FIPA, as they refer to the same protocol types. The communication travels from the initiator to the participant, just as the QueryForAdvise, which travels from the PurchaseManager to thePurchaseAdvisor in figure 2.26. This stage of the methodology serves to define the use case relevant knowledge as domain ontology class Diagram and to refer to each communication a FIPA protocol type and the domain ontology. Both modeling steps are helpful for developing Java code for JADE.

The next phase is the roles description phase, which models the lifecycle of agents based on their roles. As collaboration needs conversation and social rules, social rules will be expressed in OCL or other formats. An agent is symbolized by a package containing role classes. This package is a package of use cases from the agent identification phase, plus the given name of the role. A role in the role description phase summarizes several tasks in the course of a resulting behavior. The RD diagram models role changes (see figure 2.27) as connections between roles of the same agent. This model incorporates a dependence scheme to preserve an agent as an autonomous entity, when refusing crucial services. The authors have realized such a scheme, by introducing three kinds of role dependencies into their role description phase:

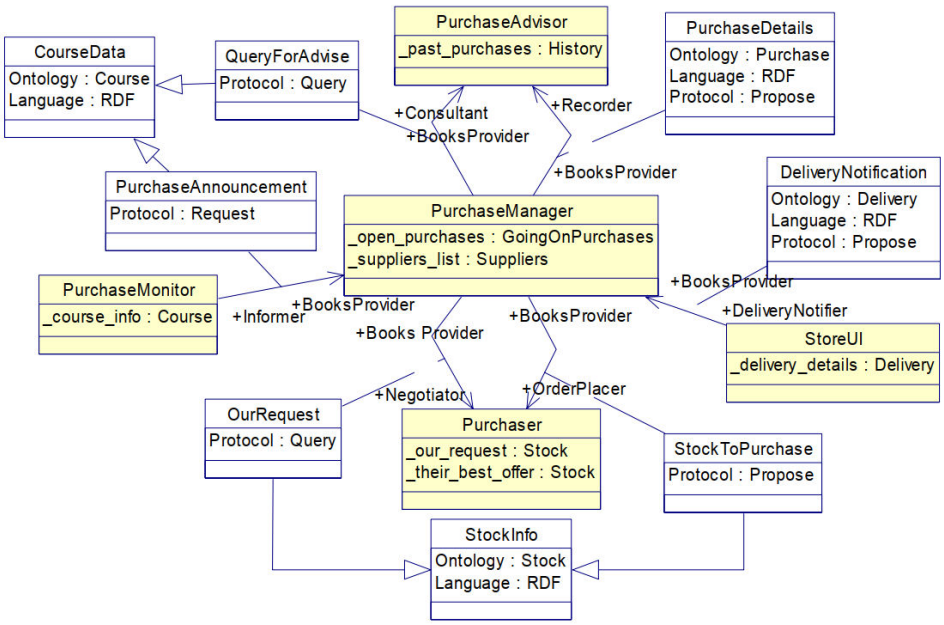

Figure 2.26: The communication ontology diagram referring to FIPA protocol types and the domain ontology, source: [BC02]

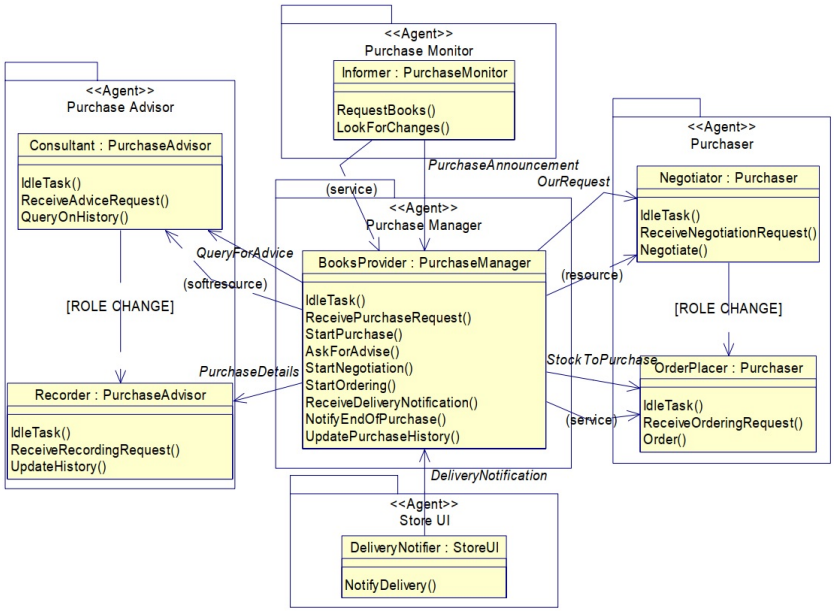

Figure 2.27: The role description diagram mapping dependencies between agents, source: [BC02]

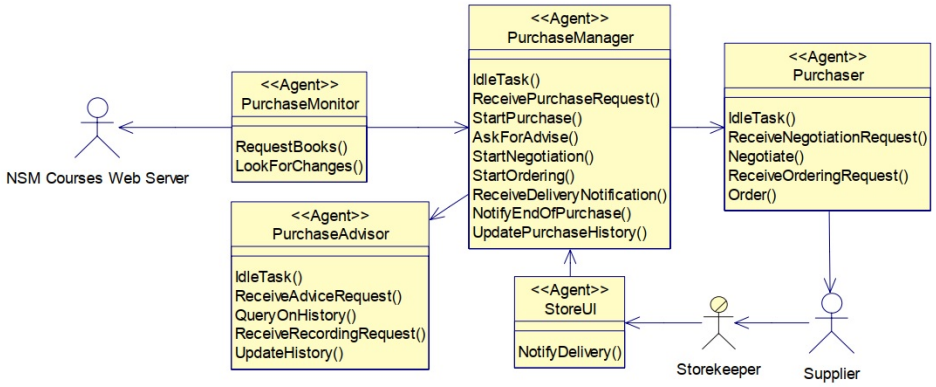

Figure 2.28: The multiagent structure definition diagram, source: [BC02]

Service dependency
which describes a role depending on another entity to deliver a service (dashed line with "service"name).

Resource dependency
which describes a role depending on the availability of an entity (dashed line with "resource"name).

Soft-Services and Soft-Resources dependency
which describes if requested services or resources are desired but not crucial (dashed line with "soft-service"or "soft-resource"name).

This dependency scheme could be practicable for considering the dependencies on the shop floor. The next phase in this design methodology is the protocol description phase. It addresses the cases, in which a FIPA standard protocol is not adequate. The authors suggested the use of the FIPA documentation approach and did not provide further details.

The next model is the agent implementing model, containing the agent structure definition and the agent behavior definition. This modeling phase iterates. Its result effects the earlier system requirement model. The next step is the agent structure definition phase, where several class diagrams with a single agent and a multiagent perspective will be produced. The multiagent structure definition (MASD) is a UML class diagram representing the complete MAS system, in which each class represents an agent identified in the Agent Identification phase (see figure 2.28). The attributes of an agent class describe its knowledge, which is discussed in the communication ontology diagram. The single agent structure definition (SASD) diagram is used to display the internal structure of an agent including its tasks (see figure 2.29). This phase combines the attributes (knowledge) and the methods with the tasks' classes, which are methods required to deal with communication events. This combination serves as a template for a software structure, which is ready to implement.

The agent behavior description phase is the second phase of the iterative agent implementing model and it produces diagrams divided into multiagent and single agent perspectives. Its result is an activity diagram displaying each communication including the exact methods triggered for each single agent and in the sum for the whole multiagent system. Different than DeLoach (MaSE), the authors stated that they do not need a specific diagram for concurrency and synchronization since the UML activity diagram supports it. Both diagrams, the single agent behavior and the multiagent behavior diagram, serve as implementation guide for the programmer. The next phase is the code reuse phase which typically takes place in CASE

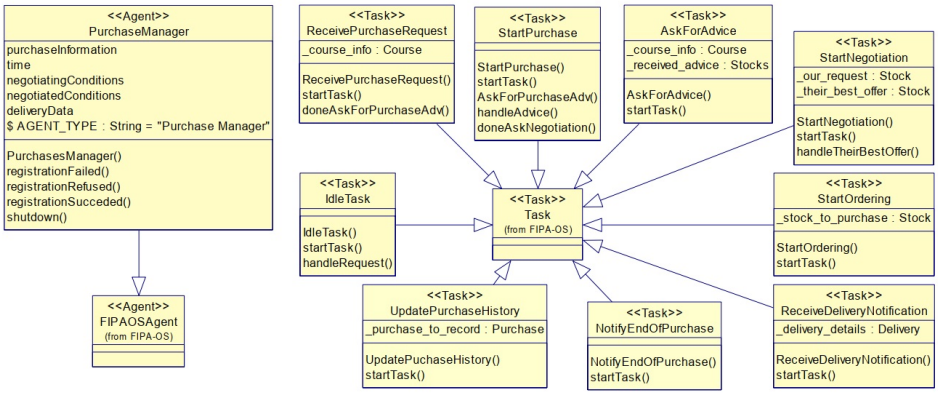

Figure 2.29: The single agent structure definition diagram, source: [BC02]

(Computer Aided Software Engineering tool environment). The PASSI method first introduced its own CASE tool, called PASSI Tool Kit (PTK) design as an add-on for Rational Rose [CSC03]. Its functionalities are total or partial automatic compilation of some diagrams, design consistency checking, generation of report documents, design documents, access to the FIPA protocol data base and generation of code. In the following years, the team around Cossentino developed AgentFactory tool, an automatic code generator for JADE and the FIPA-OS platform. Its result is a collection of reusable code pieces. Those pieces are generated on the base of the MABD, SASD, the previously described behaviors and methods and the FIPA protocol types. Its result is a code skeleton of agent main classes with their tasks. The next phase is the code completion phase, which is best accomplished by a JADE experienced programmer. The final phase is the deployment configuration phase. Its diagram shall depict the location of agents, their movements, and their internal communication support. The authors have extended UML with a syntax extension with a dashed line and a move_to stereotype to map the mobility of agents.

Advantages and disadvantages of PASSI for our investigation area

PASSI uses UML and its Agent extension AUML, as it is a widely accepted modeling language and an advantage. The Agent extension is intuitive and easy to understand since it is embedded in existing diagram types and extends them mostly by semantical names. An additional benefit of using UML is that it does not require an additional developed diagram for concurrency and synchronization. Another benefit is that PASSI ties in with existing standards and platforms such as FIPA, JADE, UML Rational Rose, RDF and XML. It also offers a CASE tool kit easing the methodology deployment and the code implementation as it generates code.

During the years, the authors have not only deployed their methodology successfully on project [CSC03], but have also extended their PASSI supporting tool kit by the “AgentFactory“, a CASE tool generating code for JADE and the FIPA-OS platform. Despite those benefits, its suitability for industrial use cases is doubted. Manufacturing use cases are very complex and mutually dependent. PASSI, would require a high number of diagrams to be modeled. The domain description phase requires displaying each possible scenario occurring in a system as a use case diagram. This is a disproportionate effort considering the simplicity of capturing those possibilities textually or functionally. Using the PASSI methodology the tool cycle, would urge to map over 18 use cases and at least as many sequence diagrams.

Although the methodology is generally extensive, it lacks comprehensibility in the agent identification phase. The authors stated that it is reasonable to combine required behavior into

one responsible unit. Since neither the criteria for a responsible unit was explained, nor for the behavior, the made choices appear arbitrary. It would be comprehensible to bind the behavior (task) to required artifacts or hardware and to highlight those necessary resources in this phase to consider them in the agent identification.

In the case of the role identification, it must be stated that the actual design phase occurred earlier, and that this phase only includes naming the structure defined before. A "package of role classes", for example shall be a package of use cases composed in the agent identification (equals agent class), addressed the name given to its role in the Role Identification phase and its tasks. It would have been more comprehensible to address the "package of role classes"as identified agent packages classified by their roles and including their tasks according to these roles.

In conclusion, it can be said, that the PASSI design methodology is a mature helpful and well-designed methodology. Its use of AUML and its embedding of domain ontology into FIPA protocol-templates is worthwhile for MAS developers, especially for those without FIPA experience. PASSI provides the designer with CASE tools such as PTK and AgentFactory, which generate code skeletons for JADE. PASSI also offers a scheme to model the mutual dependencies of agents in a system, which is very useful. Despite its numerous advantages, conducting the PASSI methodology for the integrated tool cycle and any other shop-floor processes in general, appears to be associated with a high effort. This can be reasoned as this methodology mainly uses AUML and use case diagrams with subordinated sequence diagrams if the use cases are complex and not self-explanatory. This procedure would cause, in the case of the tool cycle, numerous use cases and sequence diagrams to map the system requirements. In case of such complex systems with a high number of dependencies and restrictions, a textual description appears to be more feasible and suitable.

2.3.3 Gaia

Its name reminds of the mother earth in Greek mythology, although it was chosen after the influence hypothesis Gaia by James Lovelock. Lovelock stated that the effect of all living entities on earth can be understood as parts of an ecosystem. Wooldridge, Jennings and Kinny developed Gaia for analyzing and designing agent-based systems, where the key concepts are roles, [WJK00].

Gaia is a top-down approach, which decomposes behavior into roles. Those roles are associated with responsibilities, activities, permissions, and protocols. The interaction of roles is managed by defined protocols. The authors developed Gaia because object-oriented approaches face shortcomings during modeling individual or agent classes. Models of object classes are static, while agents and their relationships are dynamic. Beside the requirements statement, the Gaia methodology consists of two phases, the analysis phase with its roles model and its interaction model, and the design phase (see figure 2.30). The design phase consists of the agent model, the services model, and the acquaintance model. Gaia introduces two types of main concepts, the abstract ones and the concrete ones. The abstract entities appear during the analysis phase and conceptualize the observed system. Those entities do not automatically effect the resulting system directly. The concrete entities on the other hand are utilized during the design process and influence the resulting system or its parts directly. Table 2.4 lists the entities according to the two concepts. The first phase, the analysis phase builds an understanding of the system without any reference to possible implementation. In Gaia one single role does not directly define an individual as it is possible for an individual to serve in several different roles. The role model identifies all key roles of the system [WJK00].

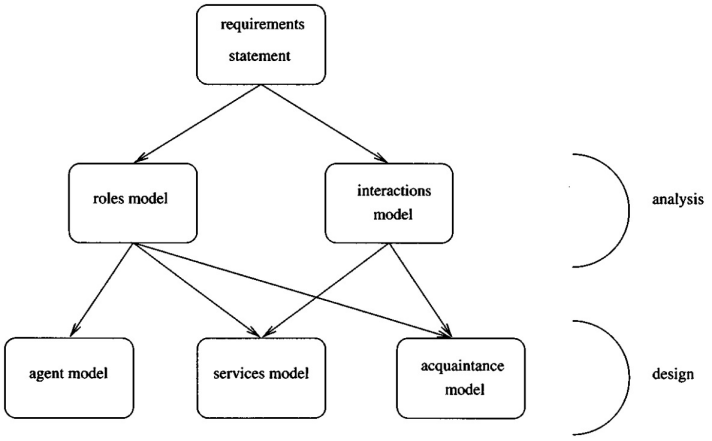

Figure 2.30: Influence of the Gaia models, source: [WJK00]

abstract entities	concrete entities
Roles	Agent Types
Permissions	Services
Responsibilities	Acquaintances
Protocols	
Activities	
Liveness Properties	
Safety Properties	

Table 2.4: List of abstract and concrete concepts in Gaia, source: [WJK00]

The authors explain their model using coffee making in an office as an example [WJK00]. The role COFFEEFILLER is written in capital letters and has a description explaining: "its purpose is to ensure that the coffee pot is kept full". A role is defined by four attributes: permission, responsibility, activities, and protocols [WJK00]. Roles can be characterized by two perspectives:

Permissions and rights associated with a role
It includes permissions for resources that are associated with a role.

Responsibilities of a role
It includes the purpose of a role, or its functionality and action which a role must fulfill.

The authors defined **permissions** as resources and tend to see them primarily as information resources. Those information resources can either be accessed (read), manipulated or generated. Permissions of a role are identified in two ways:

Identify directly deployable resources
Those are resources, which can be used intuitively by the role, in a "can be spent"manner.

Identify resource limits to be maintained
Resource limits which must be respected or maintained by the role in "cannot spent or reached"manner, are called parameterized permissions. Those parameterized permissions will be defined by the **supplied** keyword.

Operator	Interpretation
$x \cdot y$	x followed by y
$x \mid y$	x or y occurs
x^*	x occurs 0 or more times
$x+$	x occurs 1 or more times
x^ω	x occurs infinitely often
$[x]$	x is optional
$x \parallel y$	x and y interleaved

Figure 2.31: Operators of liveness properties, source: [WJK00]

In the COFFEEFILLER example, the following permissions could be modeled.

$$reads coffeeStatus \; \backslash\backslash full \text{ or } empty changes coffeeStock \; \backslash\backslash stock \; level \tag{2.1}$$

In Gaia "read "and "changes "describe the permission, while coffeeStatus and coffeStock define a resource. The parametrized permissions of the COFFEEFILLER could be:

$$reads \textbf{supplied} CoffeeMaker \; \backslash\backslash DeLonghi \; 612 \tag{2.2}$$

Beside responsibility and permission, activities and protocols do exist. Activities are computations in the sense of private actions. Protocols serve as defined interactions between roles. The authors have defined several protocols (see [ZJW03]), which are not referring to the FIPA protocols. Responsibilities reflect the functionality of a role and are the key attributes of a role. Responsibilities are divided into two types, **liveness properties** and **safety properties**. The difference between those responsibilities lies in the perspective on the incidence in which they occur. Liveness properties state a "something good happens"incidence as they define the agent's states in a certain environment conditions. Safety properties catch not desired incidence, so that "nothing bad happens". In fact, safety properties are invariants to maintain an acceptable state, as for example not exceeding a temperature maximum. In case of the COFFEEFILLER, its responsibility is to keep the coffee pot full. This responsibility can be divided into two liveness properties:

- whenever the coffee pot is empty -> fill it up
- whenever coffee pot is full -> inform worker

Liveness properties do have their own liveness expressions, which define the lifecycle of a role with the help of operators (see figure 2.31). The general form of a liveness expression is: $ROLENAME = expression$. The role name is the name of the liveness expression, while the expression refers either to activities or protocols. In case of our COFFEEFILLER role, the liveness expression for the above-mentioned responsibilities would be: $COFFEEFILLER = (Fill.InformWorkers.\underline{CheckStock.AwaitEmpty}^\omega)$. The activity of this liveness expression occurs infinitively often since the superscript ω operator was used. The behavior executes the Fill protocol "followed by"the InfromWorkers protocol, "followed by" the activities (underlined) CheckStock "followed by"the activity AwaitEmpty. For the current moment, protocols are being treated as interaction labels.

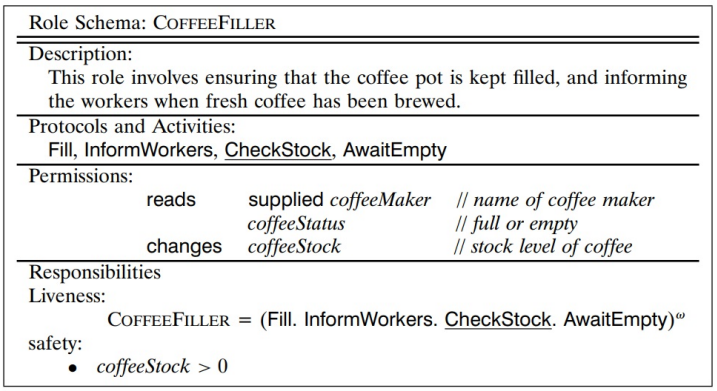

Figure 2.32: Role schema for the COFFEEFILLER role, source: [WJK00]

Beside liveness properties the role COFFEEFILLER also has safety properties, represented as invariants, which must be maintained during the roles live time. These safety properties are a list of predicates (see [WJK00]). Those predicates arise from the variables listed under the role's permissions. Pre-empting what permissions are, in the case of the COFFEEFILLER one possible safety property could be to ensure it does not run out of coffee, $coffeeStock > 0$. In summary, the above mentioned figure 2.32 records all role concepts of the COFFEEFILLER role.

Beside the roles model, the analysis phase includes the interaction model. Some dependencies or relationships between the roles are inevitable and must be mapped in the interaction model. An interaction model combines the roles with protocols, which are defined interactions between roles. Those protocols' definitions consist of:

- **Purpose:** textual description, "assign. . . ", "request . . . "
- **Initiator:** role starting the interaction
- **Responder:** the receiving role
- **Inputs:** resource/permission used by initiator
- **Outputs:** resource/permission offered by responder
- **Processing:** textual description of processing/actions the initiator performs during interaction

Figure 2.33 of the COFFEEFILLER role shows, what a protocol definition for the Fill protocol looks like. At the top is the protocol name. Under the protocol name on the left side is the initiator role name, which starts an interaction with the responder role standing on the right side. On the same level as the roles, but outside the box permissions and resources are written. This protocol definition states, that the COFFEEFILLER interacts with the COFFEMACHINE. It further states that the COFFEEFILLER fills a coffee machine (resource) named coffeeMaker with coffee. In case of the Fill protocol, "supplied"marks that using the coffeeMaker resource is bound to a parametrized permission. This protocol calls the action "Fill coffee machine". The output permission of this protocol is the generated information of the coffeeStock. An interaction model maps the possible paths of interactions by executing a chain of such protocol definitions. Thereby an arrow leads from one protocol to another. Figure 2.34 displays the scheme of a partial interaction model. One protocol definition leads to another protocol definition, and

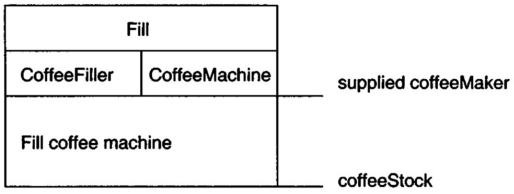

Figure 2.33: Fill protocol definition for the COFFEFILLER role, source: [WJK00]

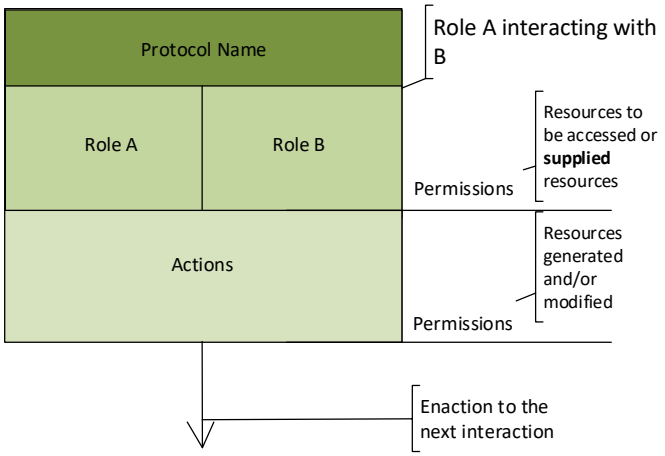

Figure 2.34: Scheme of a partial interaction model, source: Own illustration

thereby maps the interactions between the roles. The analysis phase will be iterated until all outputs, interactions and roles have been captured. The second phase of the Gaia methodology is the design phase. It transforms abstract models from the analysis phase to artefacts of a lower level of abstraction, to make them easier to implement. The authors emphasize that ways of implementing agent services are beyond the scope of Gaia and vary with the application domain. The design process consists of three models: the agent model, service model and acquaintance model. The agent model captures the various agent types and the agent instances of each type during run time. The authors defined an agent type as a set of (agent)roles. Combining the roles to a set, the designer can either choose a one-to-one correspondence between role and agent, or to cluster closely related roles together as agent type. The agent model is a simple agent type tree, in which leaf nodes are roles and other tree nodes are agent types. Figure 2.35 displays such an agent model for another Gaia use case. The roles (at the bottom in this figure) are being combined into agents (top). As the reader might have noticed, each agent type has its multiplicity of instances defined at the end of the association. The authors defined a set of instance qualifiers, as displayed in figure 2.36. Those qualifiers originate from the FUSION methodology. The authors also stated that Gaia does not consider inheritance in its agent model. This is reasoned by the fact that Gaia is applied best to systems with a small number of roles and agent types, with a one-to-one mapping. In such application areas, inheritance has no additional benefit. They also admit that inheritance is desirable when it comes to the implementation. The authors offered no particular rules on how to set the agent instance qualifier.

The second model of the design phase is the service model. This model shall aid in identifying the services, that each agent role can offer to its environment. Those services are in fact functions the agent fulfills. In object-oriented approaches, a service is in further consequence a method.

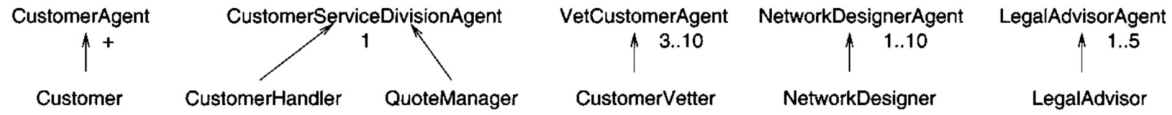

Figure 2.35: Agent model from another use case, source: [WJK00]

Qualifier	Meaning
n	there will be exactly n instances
$m..n$	there will be between m and n instances
$*$	there will be 0 or more instances
$+$	there will be 1 or more instances

Figure 2.36: Agent instance qualifier, source: [WJK00]

Service	Inputs	Outputs	Pre-condition	Post-condition
obtain customer requirements	*customerDetails*	*customerRequirements*	**true**	**true**
vet customer	*customerDetails*	*creditRating*	customer vetter available	*creditRating* $\neq$ *nil*
check customer satisfactory	*creditRating*	*continuationDecision*	*continuationDecision* = *nil*	*continuationDecision* $\neq$ *nil*
chect service type	*customerRequirements*	*serviceType*	*creditRating* $\neq$ *bad*	*serviceType* $\in$ {*standard*, *bespoke*}
produce standard service costing	*serviceType*, *customerRequirements*	*quote*	*serviceType* = *standard* $\wedge$ *quote* = *nil*	*quote* $\neq$ *nil*
produce bespoke service costing	*serviceType*, *customerRequirements*	*quote*, *serviceIsLegal*	*serviceType* = *bespoke* $\wedge$ *quote* = *nil* $\wedge$ *serviceIsLegal*	(*quote* $\neq$ *nil*) $\vee$ (*quote* = *nil* $\wedge\neg$ *serviceIsLegal*)
inform customer	*customerDetails*, *quote*		**true**	customers know quote

Figure 2.37: Example of a service model of a different use case, source: [WJK00]

Those services not available for other agents, like a method would be. The services are a "single coherent block of activity". Each activity from the analysis phase corresponds to a service. The service model also specifies the main properties of these services. Those attributes (see figure 2.37) are input, output, pre-condition and post-condition. Input and output of a service will be derived from the protocol definition. Pre-condition and post-condition capture the constraints of the service and relate to the safety properties. The last model in the design phase and in the Gaia methodology is the acquaintance model. According to the authors, it is the simplest model and defines the communication links between the agents. Figure 2.38 shows an example of an acquaintance model from another use case. The communications link only reflects the existing interactions between the agent types. They do not give qualitative information about the messages.

Advantages and disadvantages of Gaia for our investigation area

Gaia dissociates from object-oriented design methodologies, as it states that they are too detailed and lack a higher level of abstractions. Gaia includes, however, aspects from the FUSION methodology, such as the agent instance qualifier. Since Gaia uses natural concepts of roles, responsibilities, permission, and services without any additional modeling language, it would easily gain acceptance from industrial and mechanical engineers. Also, the progress from one

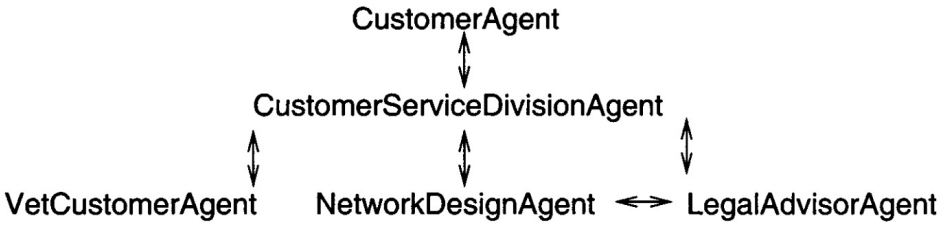

Figure 2.38: Example of an acquaintance model, source: [WJK00]

methodology step to another is intuitive and easy to accomplish.

Gaia demands no prior computer-scientific knowledge, as it does not at any stage consider implementation concerns. Another benefit of Gaia is that its textual description and natural concept captures an agent society better, than object-oriented approaches such as class diagrams. Further, such object-oriented approaches do not consider any system changes during run-time. Mapping an agent society with object-oriented approaches, such as UML, forces the designer to make design decision, which affect the outcome strongly and, way too early in the process. Gaia honors the idea of encapsulation as it offers self-containing concepts as protocols (interaction mapping), and permissions (resource facilitation), giving the possibility to link those concepts if required.

On the other hand, Gaia has been developed for the application on small systems, as each agent type is designed individually, and no inheritance exists. It also limits its focus on the design aspect of the MAS, not offering any mechanism to check for feasibility of the implementation. Gaia is, different than PASSI not developed for any specific platform. As a further consequence, the tools that Gaia offers the designer are not as extensive as for example PASSI. Gaia also does not support any implementation of the autonomy defined protocols, leaving the designers on their own, when it comes to the question of how to realize a negotiation interaction between the agents. This is reasoned by a high level of abstraction, that Gaia maintains during its design steps. On the one hand, this high abstraction gives the designers more creative freedom and requires less implementation knowledge. However, it does not offer insurance that an implementation is feasible, and requires high effort at the implementation phase.

In conclusion, it can be said, that Gaia enables every process stakeholder regardless of what her or his professional background is, to design a MAS. It does not demand any modeling language skills, as most parts are captured either in a textual way or in intuitive models. The introduced concepts of Gaia are natural concepts, such as roles, resources, permissions, and interaction models based on simple protocols. However, Gaia is limited to small use cases with a limited number of roles and restrictions. Its suitability for a process in the manufacturing domain is therefore doubted.

CHAPTER 3

MAS design methodology for the manufacturing execution system domain (DM-MESD)

This chapter presents the developed MAS design methodology for the manufacturing execution system domain (DM-MESD). It is structured in six phases, each one describing the higher purpose of its methodological phase. Those phases are the evaluation phase, the assessment phase, the analysis phase, the service identification phase, the service clustering phase, and the transformation phase. Each phase is divided into methodical steps, guiding the designer through the developing process. While presenting the design methodology, the DM-MESD is being continuously compared with the PASSI and Gaia methodology. This comparison shall ease the understanding of the newly presented methodology and show its differences.

3.1 Design methodology

This design methodology was developed for the presented investigation area. Its main objective is to enable process stakeholders from the manufacturing domain to evaluate, if an agent-based process control could be of any benefit and if so, to enable those stakeholders to draw their first design. Identifying the process stakeholders as industrial and mechanical engineers, sets high demands for this methodology. One of those demands is to integrate agent-oriented aspects as mechanisms within the methodology. Those mechanisms enable designers without MAS experience to fully exploit the MAS advantages. Another demand on this methodology is the continuous guidance through the design process. The designer should not be compelled to second guesses her or his next step. The DM-MESD is built on the process-oriented perspective and differs from common design methodologies, which are based on role concepts. The DM-MESD elaborates services from the core process by textual description and structured analysis. This methodology offers an evaluation phase at the beginning of the process to evaluate if this design methodology is suitable for the desired application domain. The DM-MESD focuses on the MAS design and does not consider any implementation subjects or limitations. It sets Key Performance Indicators (KPI) and offers thereby its user the chance to pursue optimization strategy. The design methodology consists of six phases and their methodological steps, which are displayed in figure 3.1.

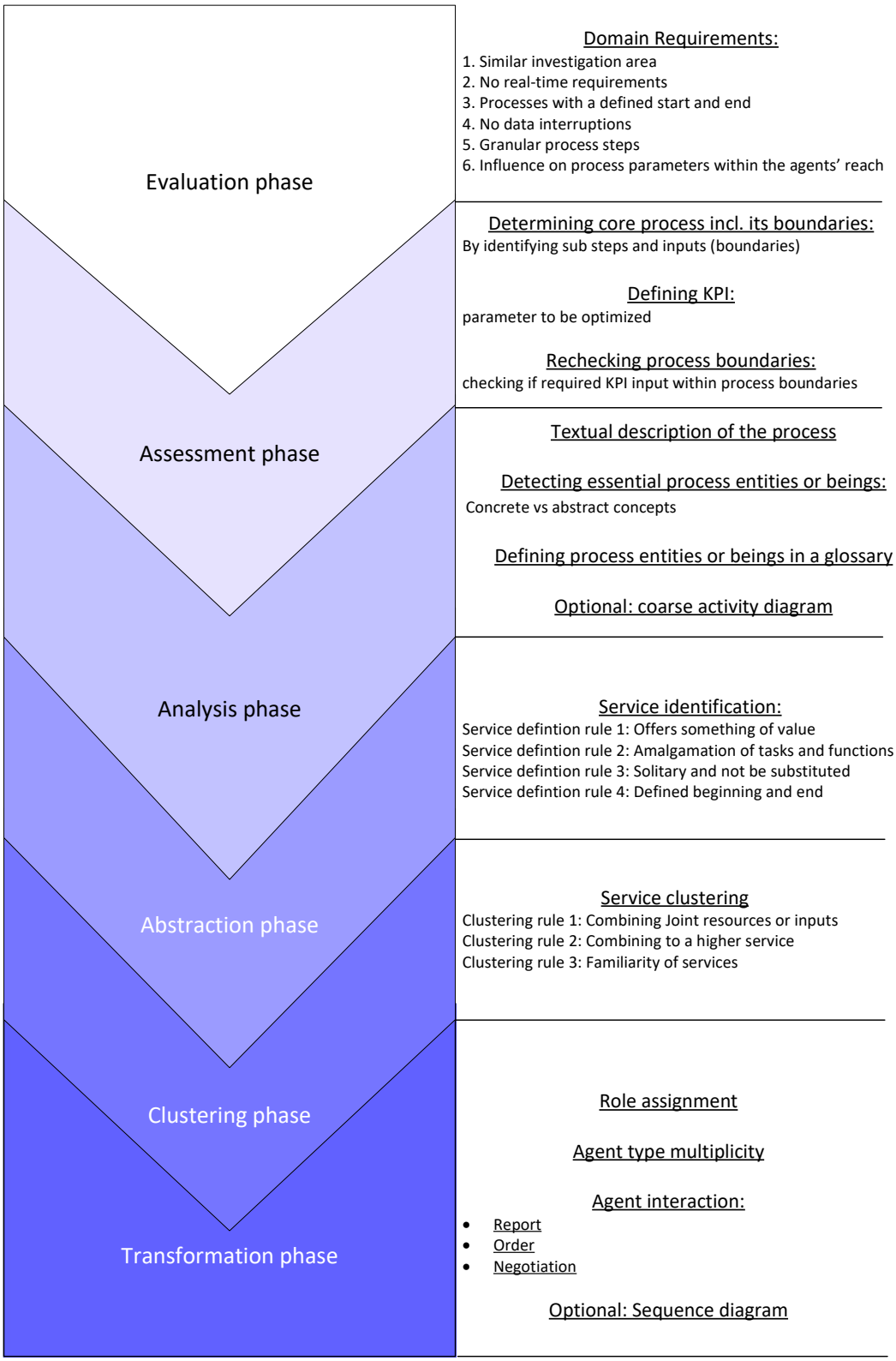

Figure 3.1: The DM-MESD and its six phases, source: own illustration

3.2 Evaluation phase

The reviewed design methodologies have not included any previous form of evaluation. This is a clear disadvantage. The first phase is the evaluation phase, which has only one step. This step is controlling if all six domain requirements, which derive from the investigation area, are fulfilled. The evaluation phase ensures that the target application meets the criteria of the investigation area, and thus the DM-MESD is suited. If the target application meets all six requirements the designer may proceed to the next phase. If the target application does not meet one or more requirements, either the issue must be resolved, or the domain perspective must be adapted. The DM-MESD is not suited for the target application if none of both options work. The six domain requirements are:

- Process within a similar investigation area
- No real-time requirements
- Processes with a defined start and end
- No data interruptions during the process
- Granular process steps with measurable input and outputs
- Influence on process parameters within the agents' reach

Requirement 1: Process within a similar investigation area
This requirement was established since the design methodology presented here is limited to the earlier presented investigation area. It can not be stated for certain, if the DM-MESD can be successfully deployed outside the investigation area. The main characteristics of the investigation area are introduced here. The investigation area is limited to **discrete manufacturing**, as the continuous manufacturing has different demands to on its process control. The continuous production process is influenced by the quality and quantity of its input material, which is not the case in discrete manufacturing.
The investigation area is also limited by the **size of the production**, as only small to mid-size serial productions are addressed. This limitation effects the order execution time. While a mass production handles orders in sizes, which occupy the production lines over several hours or shifts, a small size to mid-size serial production has shorter productions cycle more suitable for flexible orchestration by multiagent systems. Even smaller productions, such as job shop production could benefit from the flexible qualities that a MAS control introduces.
From the perspective of the application layer, the target application shall **not be located lower than the MES layer**. One reason is that processes on a higher layer offer a higher grade of abstraction. The higher the grade of abstraction, the easier is to integrate multiagent systems as process control. This design methodology might also be a suitable for processes on the ERP layer. Although it has not been designed for them in the first place, nor has it been tested on such processes.
Requirement 2: No real-time requirements
This requirement emphasizes that the methodology presented here is not suitable for real-time requirements such as those required for the PLC layer. Such requests invariably require the processing of incoming data without a buffer delay. There is a large number of scientific works dealing with the high requirements of such systems and the special adaptation of the multiagent paradigm to those. The author cannot stress enough, that neither this methodology nor the use case, presented here touches an area in which real-time requirements would be an issue. This is

important as no design mechanisms or concepts were developed, that would ensure a proper handling of this subject.

Requirement 3: Processes with a defined start and end

This restriction was set, to ensure that only processes are considered with a defined starting and end point. This restriction excludes processes that are run as a manner of a continuous controlling loop.

Requirement 4: No data interruptions during the process

For a process to be managed by a multiagent system, a complete digital representation of the process stages must be available. The MAS must be capable of monitoring the process and to set actions, that either directly or indirectly influence the process. Data interruptions are blind spots in the process and can either be overcome by technological solutions, such as interfaces, storage etc., or by process adaptation, providing or excluding this data.

Requirement 5: Granular process steps with measurable inputs and outputs

To identify if a process is going along as planned, process steps with significant results must be available and defined.

Requirement 6: Influence on process parameters within the agents' reach

To implement a MAS based process control, the agents must be capable of manipulating the actuating parameter either directly (by setting them) or indirectly (environmental changes). This might be best explained on the example of a MAS deployed to draw and sell paintings to human customers. The agents must be able to access feedback data from the customer (no data interruption). Based on that feedback, data on customer preferences can be gathered and evaluated. Now the agent must be capable to adapt the next painting (process parameter in the field of action). The future agents must have one or more process parameter in their field of action, so that they can influence the process and its outcome.

If the target application does not support any of the above-mentioned requirements in its original state, adaptations must be made. Those adaptations can either be made by resolving the issue directly or by limiting the application to an area, that meets the requirements. If neither resolving the issues nor limiting the target application work, the DM-MESD is not suited. But if all requirements are fulfilled, the next phase can be proceeded.

3.3 Assessment phase

This phase helps the designer to determine the core of her or his application process. This is accomplished by determining which part of the desired process meets the core process definition. The first step defines a core process and how it can be differentiated from other process forms by using criteria. Those specific criteria determine the core process entities. Its result is in a first table of sub steps and their required inputs, which mark the process boundaries. The second step of this phase chooses the Key Performance Indicators (KPI). The set Key Performance Indicators (KPI) give the designer the chance to pursue optimization strategy during the MAS designing. Analyzing the KPI shows what information and tasks are required to optimize them. The outcome of this step is a short textual description of the pursued strategy, as well as its KPI. The outcome of this KPI assessment is important groundwork for the next and third step of this phase, the rechecking of process boundaries. The process boundaries have been drawn during the determination of the core process and did not consider possible KPI and their required information or task. This last step of this phase rechecks the earlier set process boundaries based on whenever they include the required KPI information or tasks.

3.3.1 Determining the core process, including its boundaries

The term core process is used in many different areas. It is mostly used to describe the core business process of an enterprise. The core process is defined in this thesis as a process pursuing a main objective by fulfilling several sub steps with defined inputs and tasks. **A core process has an overall objective.** The achievement of this overall objective must be measurable and divisible in sub-achievement. **Its objective is realized by fulfilling several subordinated tasks, here called sub steps**. Complex objectives can be achieved by fulfilling several sub steps. A core process has sub steps with qualitative or quantitative parameters, which illustrate their success. A step consists of a task or a function and has an input and an output. A task is connected to the physical world either by its physical input or by its physical action, that must be performed. A function represents here the formal transformation, which takes fully place in the digital world. It might be a mathematical function or a method call returning a defined digital output for a digital input. Defining a core process fulfills the aim to reduce every aspect not necessary for the process. Doing so automatically limits the design effort as well as the complexity of the use case. To assess a core process the following entities must be determined:

Overall objective
: A core process is the remaining part of a process, which cannot be furthered narrowed without losing its functionality to achieve the overall objective. This overall objective must be measurable either qualitative or quantitative manner.

Sub steps
: The achievement of the overall objective must be divisible into a sequence of sub steps. A sub step has a defined task or function, processing a defined input into a defined output.

Input for each sub step
: Those inputs and outputs must be defined in a qualitative or a quantitative manner, to evaluate the achievement of the sub step

Task or function in each sub step
: Each sub step describes a task or function to be fulfilled on input or output. A task is an action set on a physical item. A function is a relation between one or more logical inputs and outputs.

If the target process meets the demands of a core process, its overall objective is being mapped by an informal textual description. Then it will be roughly clarified if the core process has sub steps. If so the sub steps, their tasks or functions and their inputs are listed, in an assessment table. The **core process is defined in an informal textual manner**. This textual definition can be divided into two parts. First, the overall objective is described in a short text, a few sentences are enough. Second, the sub steps including their task or functions and their inputs and outputs are listed. This structured listing of the sub steps does not have to be detailed. However, this coarse listing must mention all important tasks or functions and their inputs. This description can either be realized by a short text or a structured table as schematically depicted in table 3.1. After the sub steps are listed in the table, the next step is to evaluate, whether all inputs can be accessed within the core process. This practice is essential for determining the process boundaries. By limiting the boundaries to the inputs of the sub steps, the process will be narrowed automatically. The input of the sub steps defines the artefact or interface that marks the process boundary. The finer the grading of the sub steps and inputs, the more precise the process boundaries are. The previous sub steps listing serves as a control for completeness.

Core process			
Overall objective:			
Objective measurability:			
Sub steps	**Input**	**Task or function**	**Output**
Sub step 1	Input 1 [unit]	Task 1	Output 1 [unit]
Sub step 2	Input 2 [unit]	Task 2	Output 2 [unit]
Sub step 3	Input 2 [unit]	Task 3	Output 3 [unit]
Sub step 4	Input 4 [unit]	Task 4	Output 4 [unit]
...	...		...
...	...		...
...	...		...

Table 3.1: Core process assessment table

3.3.2 Defining the pursued strategy and its KPI to be optimized

This step sets Key Performance Indicators (KPI) and offers the designer the chance to pursue optimization strategy. If the designer wants to map the process in its origin state this step of the assessment phase can be skipped. KPI are measurable indicators matched to the pursued strategy, stating the performance of an action or process. A KPI is qualitatively or quantitatively measurable. Further, a KPI must not conflict with the overall objectives of the core process. All the actuating values and functions of the KPI must be either within the process boundaries or have a defined interface to it. The defined interface must allow the core process to influence those actuating values or functions. This does not mean that their value is only effected by the core process, but that some influence from the core process is granted. A KPI can be a value influenced and set by several sub step parameters. The number of KPIs, which can be defined within a core process depends on their mutual dependencies and the designer's choice. It is possible to choose two or more KPI, depending on each other to find an equilibrium between those variables. First, the KPI is described in a textual form, capturing its objective in words. Afterwards the KPI must be formulated as a relation between its inputs and the action. This can be realized either in a textual manner or with the aid of mathematical formalism. The main interest is to clarify, which parameter or tasks and function has an influence on the KPI. This clarification is needed for evaluating in the next step if all influencing parameters and functions of the KPI's are still within the process boundaries and therefore within the field of action.

3.3.3 Rechecking the process boundaries

While the earlier drawn process boundaries only depended on the required inputs, the revised boundaries consider the KPI. This step controls if the process boundaries exclude the required data or function for the previously defined KPI. The KPI might require input outside the process boundaries. If so, either the process boundaries must be adapted, or interfaces and artifacts must be defined. In this case the data as an artefact must be defined and provided from the outside the process boundaries. This definition includes its format (XML, SQL query, text file, OPC UA, etc.), its content and the interface it uses to be transmitted. This interface is one way to define process boundaries as they mark the link between the internal and external system. If establishing such artefacts or interfaces is not possible, the KPI must be adapted. If the designer chooses to adapt the KPI, the author suggests to evaluate if the required data or function can be derived from another source or in another manner. The process boundaries need adapting, if the

required data or functionality can not be provided through artifacts or interfaces. In that case the process boundaries must be adapted so that they include the required data or functionality. In both mentioned cases, adaption of the KPI or of the process boundaries, step two and three of this phase must be repeated. These three steps are the abstract part of the analysis phase, sharpening the designer's perspective for her or his core process.

3.4 Analysis phase

This phase produces a textual description of a core process and its entities. It consists of four methodological steps. Now that the observed process is defined and demarcated, the analysis phase begins with the textual description of the core process. During second step in this phase the core process is reduced to the smallest possible number of tasks and entities. Essential process entities are elaborates based on the textual description. If essential process entities require further description, they are being defined in an informal glossary giving the designer the chance to sharpen her or his understanding of them. Based on the textual description and the glossary a coarse activity diagram can optionally be designed.

3.4.1 Textual description of the process

The textual description serves as a mapping mechanism. This modeling method can easily be used by industrial and mechanical engineers with no modelling experience. Most industrial or mechanical engineers have only basic or no experience with modeling languages, such as UML, Petri Net, etc. A textual description is the most natural way to capture a process. The textual description is also a common method to capture a production process in manufacturing. A good example for this is a work plan, which describes in detail the sequence of tasks necessary to achieve a desired outcome. Although ERP and MES software offer process flow-based representations of work plans, it is still common on the shop floor to have a textual described work plan. The advantage of a textual description is its unambiguous grasp of the process, if worded precisely. It is possible to achieve the same quality with a UML model in a more eloquent manner, but that would require an experienced modeler on the one side and a just as experienced model user on the other side. If this is not present, inaccuracies or misinterpretations are predestined. Another reason for the textual description is the low effort required to map even complex systems. In the modeling with UML or any other modelling language, complex systems are mapped with nesting mechanisms. Modeling a system is always a tradeoff between abstraction, offering simplicity and detailedness, offering the entire information. Modeling a system reflects a purpose. This purpose guides the designer through decisions, whether to neglect a systems' aspect in the course of abstraction or rather to observe it in detail.

The textual description has no specific form. The designer should however keep in mind that a short and precise text, is easier to handle in the upcoming tasks. The textual description begins with the very start of the core process and ends with its last sub step. It covers in the course each necessary action or task in the progress, as well as each entity required to fulfill the tasks. Thereby the process is described in a narrative manner. The degree of detail or the number of detours to explain possible scenarios, is the choice of the designer and the stakeholder. Another benefit of a textual description is that the process stakeholder could deliver such a textual description, to the person that should design the MAS.

3.4.2 Detecting essential process entities or beings

Based on the textual description process entities can be elaborated. The process entities are divided into two concept groups; the concrete and the abstract concept group (see table 3.2). While the concrete concept group covers the input and outputs previously introduced during the assessment phase, the abstract concept group covers tasks and functions, which are the basis for services. In the concrete group of process entities, we differ between the following concepts:

- Resources
- Information
- Material

Process entities	
Concrete concepts	**Abstract concepts**
Resource Information Material	Task Function

Table 3.2: Process entity concept

While some methodologies such as Gaia consider only information as a resource, the DM-MESD defines **resources** in a manufacturing manner. The resources are physical items, which are required for the value adding process, but are not consumed during it. Examples for such resources are machine tools, workstations, robots, etc. Resources are occupied during fulfilling a task. While an information or a material runs through changes during the process, resources remain at its original state. A resource aids a task or function to perform a transformation on an information or material, without being merged into the outcome.

Information is a necessary process entity, required in a certain form and quality at a certain stage of the process to proceed. In this context information is defined as a entity serving a task or function as input. An information can be data, a list, a file or a state. An information can be accessed or to manipulated it when needed or included in the upcoming task or function.

Another concept of the process entity is the **material**. During the process it is consumed or transformed. If consumed, the material feeds into the output during a value-adding process. If transformed, one or multiple characteristics of the material are being changed, in the course of the value-adding process.

Tasks and functions are abstract process entities. Abstract in the sense that both concepts are not representing a physical entity or its data. Tasks and functions initiate a transformation from one state to another. The term of a **function**, used here, describes the formal transformation of a logical input, such as an information into a logical output. A function is a transformation taking place in the digital or logical world, such as calculations, generations, simulations, etc. A function is defined as a qualitative or quantitative tool set to transform a predefined input into a predefined output. A function can be a mathematical function, or a method call returning a defined output for the given input.

A **task** on the other hand is a transformation at least partially taking place in the physical world. A task is performed on a resource or a material and can also require information for its fulfillment. An example for a task could be the loading or unloading of cutting tools or their

Process Entities		
Input [Concrete Types]	**Task or Function**	**Output [Concrete Types]**
Input 1 [Resource] Input 2 [Information] Input 3 [Resource]	Task 1	Output 1 [Information]
Input 1 [Resource] Input 4 [Material] Input 3 [Resource] Input 5 [Information] Input 5 [Information]	Task 2	Output 2 [Material] Output 3 [Information] Output 1 [Material] Output 1 [Information]
... ...	...	... ...

Table 3.3: Process table

assembly. Both types of concepts can be determined by analyzing the textual description. The abstract concepts, such as textbftasks and functionscan be found in the textual description by analyzing verbs used to describe the process flow and by underlining them. These underlined words are not only limited to the verbs themselves, as pronouns are required to clarify the task or function. For example, not only the verb "update" should be underlined, but also the object that should be updated.

The concrete concepts, such as information, material or resources that are required to fulfill the underlined verbs can be emphasized by boxing them. Both process entity types should then be listed in a table such as table 3.3 shows. The reader might notice that at the core of this table are either tasks or functions. Since this methodology is built on the service concept, tasks and functions are indications of future services, and the main point of interest in this phase. During this step, all required tasks and functions are listed with their required inputs and their resulting outputs. Thereby a task or a function can utilize several inputs at once, which will be captured in the table.

3.4.3 Describing process entities in a glossary

Now that the essential process entities are detected, process entities requiring an additional explanation or definition can be described in a glossary. To avoid interrupting the narrative by detailed descriptions of process entities, an additional informal glossary is compiled. It shall offer the designer a place to describe essential process entities in detail if required. The informal glossary defines the name, by which the entity will be referred. It further describes in a textual manner its nature, purpose, characteristics, conditions and its restrictions.

3.4.4 Optional: Coarse activity diagram

Based on the textual description and the glossary a coarse activity diagram can be designed optionally. This UML diagram is not required. It serves as an overview of the process if desired. To design this optional class diagram the process table (see table 3.3) can be consulted.

3.5 Abstraction phase

The earlier mentioned design methodologies, Gaia and PASSI, utilize roles to design multiagent systems. This approach does not use the role concept, because the author thinks that identifying

and clustering of services would be a more suitable fit for manufacturing processes on the MES layer. This is reasoned, as the role perspective of a system is limited to its current state. The role concept does not consider the possible development of responsibilities by new interfaces or tasks, nor does it consider extending the roles range during that. A role is an abstract concept not directly bound to the physical nor the digital represented entity, it most commonly reflects already existing actors and their tasks. It does not combine them to higher level abstract services. The service-oriented perspective of the DM-MESD leads to the development of self-containing services out of tasks and functions of the core process. Further the DM-MESD offers tools to develop new services, which extend the functionality of the core process. The prior phases of the DM-MESD ensured that all required tasks, functions, inputs and output of the core process, are self-contained within the boundaries of the process.

Assumption 1 *„The services found within the core process boundaries are most properly also self-contained, as all they require will be offered within the process boundaries. “*

This assumption reflects the top-down perspective of the DM-MESD approach. The abstraction phase consists of only one step. In this step the previously listed tasks and functions are being combined into services. Beside those services deriving from the mapped tasks and function this phase offers the designer the chance to introduce new services into the core process. Those **new services** can be designed freely along as their required inputs and are within the core process boundaries.

Service identification

This step identifies services either from the previously mapped tasks and functions or defines so called new services. New services are only partially or not all based on existing process functions or tasks. Those services can be chosen freely or by considering the set KPI if their required inputs are available within the process boundaries. Both types of services are determined with the aid of four service definition rules. The result of this phase is a list of services summarized in a table (see table 4.3). Those four rules define the service term and its requirements.

> **Service definition rule 1: Value**
> *„A service provides or offers something of value to another party.“*

A service must offer something of relevance to its environment. The relevance is not only defined by the output (data or physical transformation), but also as something of a higher value. Offering another party data does not provide a higher value, but offering information is, as information is worked-up data for a specific a purpose.

> **Service definition rule 2: Amalgamation of tasks and functions**
> *„A service is an abstract amalgamation of tasks and functions, combined to provide or offer something of value“*

Simple tasks and functions can be merged, while achieving an output of a higher quality. Those tasks or functions must either support each other in their pursuit or cover different aspects of a higher goal. For example, the tasks existing solitary in a car shop, can be combined to a higher service. A car shop disposes individual tasks, such as, “checking the pressure of the tires”, “regulating tire pressure”, “changing oil”, "reading board computer error” and “error handling”. Those tasks can be combined and offered to another party as the service “Interval inspection”.

Service table		
Task or Function [Types]	**Service**	**offered value**
"T1"[Task] "F1" [Function] "F2" [Function]	Service "S1"	offered value 1
"T3"[Task] "T4"[Task]] "T11"[Task] "T15"[Task] "F2" [Function]	Service "S2"	offered value 2 offered value 3
... ...	...	... ...

Table 3.4: Service table

This service offers an outcome of a higher quality and is at the same time an abstraction of the amalgamation of tasks and functions. At this point it shall be noted that some combination of tasks or functions do not have to be performed in a specific sequence, and some do. For example, it is irrelevant if the oil gets changed before the pressure of the tires will be checked. But it is required that the task "regulating tire pressure" is performed after "checking the pressure". Some tasks, such as the "regulating tire pressure" task depend on the output of another task.

Service definition rule 3: Solitary and unique
„A service is solitary and cannot be substituted by any other service, for it is unique."

A criterion for being self-contained beside the required input, is that the functionality of the service is not part of any other service. If, for example, a service s_1 requires its input from service s_2 and returns its output to s_2, such as it is common in a controlling loop, then both services can be aggregated. A service also offers something of value in a unique manner. Its outcome or performance can not be substituted by another service. Further a service must be unique. Unique by the means that the value or benefit that this service offers another party can not be rendered by third party. Of course, heritance of the same service provider is excluded from this restriction. Thereby a task or a function can also be accessed by another service. The service "change tires" will beside other tasks include, the "checking the pressure of the tires" and "regulating tire pressure" tasks.

Service definition rule 4: Beginning and end
„A service has a precise beginning and end."

This criterion supports its encapsulation. It must have a precise beginning with preconditions or an input and an ending with postconditions or an output. The service table is used to summarize this service-information, as table 4.3 shows. This table reflects some of the service rules directly. It discloses if a service is unique by the sum of its task or functions. This table lists all values a service offers. No other service should have the exact same tasks or offer a substitutable value (service definition rule 1). However, the table shows that some tasks or functions, such as "F2" can be utilized by different services, without an effect on the uniqueness of the service.

3.6 Clustering phase

The naming of the phase conducts the creative process of clustering the identifying services. This phase consists of only one step, which offers the designer clustering rules. With the guidance of the clustering rules presented here, the designer shall combine and services to a cluster. Those service clusters will represent future agent types including their offered services. Most object-oriented MAS design methodologies share a composing phase. PASSI for example also has such a phase earlier in its process, where use cases are composed together under a bridging role. Gaia also offers such a composing phase, but rather at the end of the process, where roles are combined to agent types. Before Gaia combines the roles to agent types, every aspect of the role (responsibilities, protocols, permissions) has been defined. The DM-MESD enables the designer to compose the services and their aspects at an earlier stage, before defining protocols or acquaintances. The DM-MESD gives the designer clear guidance during the composing phase, by offering clustering rules. Offering guidance during the composing phase is not new, as the DACS methodology of Bussmann also includes such rules. But different than in the DACS methodology, the DM-MESD does not focus on decision making tasks, but on services, and so do the composing rules.

Service clustering

Since all previously elaborated services are self-contained and encapsulated, the creativity is encouraged. Now the designer can combine those services into clusters in search of benefits, such as synergies, resource allocations or services of a higher quality.

> **Clustering rule 1: Familiarity of services**
> *„Services should be clustered, if they have a familiar nature or structure and their combination could work out synergies, such as resource allocations “*

This rule is developed to cover services sharing the same mechanism. If for example two services handle bidding tasks on different resources or information could maybe be joint. This familiarity rule would suggest testing if composing those familiar services could be of any benefit. Such benefits are synergies that could be exploited. Synergies, such as using the same algorithms (functions), or slightly adapted interfaces or protocols (inputs). The motivation is, that such unused synergies entail optimizing potential not only for resource allocation, but also for decision making. Merging familiar services could better exploit decision-making algorithms, optimizing them by learning from the different areas.

> **Clustering rule 2: Combine to a higher service**
> *„Services should be clustered, if a service of a higher quality could be achieved by combining them. “*

This rule covers the possibility of services with different resources and input information to be joint to a higher service. This rule only focuses on the new benefit that a service clustering could bring. A higher service is a service whose potential can only be exploited if two smaller services where joint together. If such higher services could lead to a process transformation in a beneficial manner. Again, deciding if combining services to a higher service is worth considering is the choice of the designer and process stakeholder. Since the process stakeholder can evaluate if the combination of services brings any further benefits.

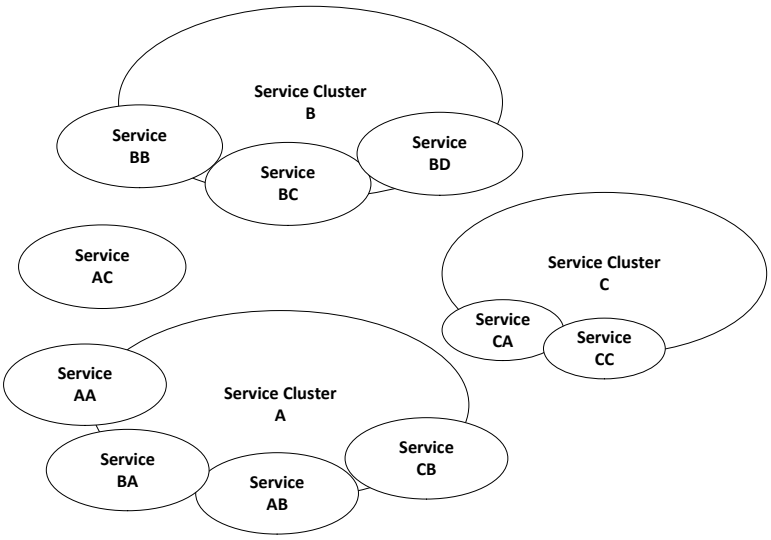

Figure 3.2: Scheme of clustered services, source: own illustration

Clustering rule 3: Joint resources or inputs
„Services should be clustered, if two services need the same input or resource.“

This rule states, that services, which access the same information, input, or the same resources are likely to be composable. It shall improve the encapsulation idea, as services utilizing the same resources or information can be jointed together. This rule shall however come into effect if the services do not access multiple other resources. It is not useful to force two services together if they share one out of five resources. For example, a service s_1, is based on the information i_1 and nothing else, and another service s_2 uses resource r_1, r_2, $r_3$and the information $i_1$and i_2. Then the designer could consider the service s_1 as a sub-service of s_2. However, the choice is most likely domain specific and the process stakeholder should have the final saying in this matter. This rule exploits service dependencies, such as shown in the example. By clustering service using the same resources controlling mechanisms of the resource can be joint. It also enforces the paradigm of encapsulation from the resource perspective and raises transparency of responsibilities and control structures. The result of this phase are service clusters, each representing a future agent type. Those agent clusters can be depicted in a similar diagram, shown in figure 3.2.

3.7 Transformation phase

Now that the services have been clustered, those clusters can be transformed into agent types. Thereby, this methodology does not focus on the internal architecture of those agent types. It rather focuses on the services the agents will provide and how they interact with each other. Both agent aspects, the functionality described by services and the interaction between those services is described in a textual manner. This methodology does not address agent implementation mechanisms, nor does it suggest any tools or platforms. The result of this phase is a multiagent diagram. Like the acquaintance diagram of Gaia, the multiagent diagram states only the role, the multiplicity and interaction between the agent types. The interaction is captured in a coarse manner only summarizing the way in which the agents interact with each other. As figure 3.3 shows the multiagent interaction diagram, in which a role will be assigned to each service cluster. Each agent type, represented by its role will receive a multiplicity, which will be determined

in the second step of this phase. Afterwards the third step of this phase will elaborate the general type of interaction between two agent types will pursue. Last step of this phase, which is optional the designer can capture the interactions between the agent's types in detail by a UML sequence diagram.

3.7.1 Role assignment

While the reviewed methodologies focus right at the being on roles and responsibilities that agent types could have, this methodology assigns roles at the very last moment. Assigning roles without an understanding of possible services, and in further consequence interactions and responsibilities is not suitable for manufacturing. The role assignment step of the transformation phase is only performed for the sake of completeness, as each cluster already has its services (methods) and interactions defined through inputs and outputs with other clusters (agents). The role assignment is done by naming the clusters. This naming is best done by observing the most relevant services or benefits a cluster has or by finding collective term for the different services. The result of this step are role-reflecting names in the interaction diagram for each agent types, placed in the middle of the service cluster (see figure 3.3).

3.7.2 Agent type multiplicity

Except for Gaia, none of the reviewed design methodologies addressed the matter of agent type multiplicity. Multiplicity is important for the manufacturing domain, as the agent types are digital representations and control entities of physical world entities. At this point the designer must question, if the multiplicity in the physical world is possible and if it would be of any benefit for the core process or the system. In doing so the designer must differ between the setting she or he has on hand and the more general possibilities the domain could offer. For example, if a production is limited by having only one industrial robot covering both the handling and the weighting service, it does not mean that the designed MAS must generally limit the "Weighting agent type" to one agent per system. This limitation could be respected for the setting in which the MAS is deployed. Overall, the designer must question if a higher multiplicity of an agent type is possible. If so, the next question must be, if a higher multiplicity of the same agent type would bring any additional benefit. This additional benefit could be reasoned in cooperating or competing agents. The result of this step are multiplicity ranges for each agent type limiting the amount of agent type instances during runtime. The multiplicity is captured as a number range between 0 and infinity (notated as "n"). This multiplicity is notated in the interaction diagram above each agent type and emphasized by an underline (see figure 3.3).

3.7.3 Redefined process entities

Since the core process was restructured and controlled by agent types (service clusters), some process entities might require adaptation. This adaption arises since process entities might need to cover either new tasks and roles or need to provide a different data set. Another reason is that new services might require new or adapted process entities. This step shall address those redefined process entities, artefacts and new entities. Those new or adapted process entities are described in an informal description.

3.7.4 Agent interaction

Inputs and outputs are the basis of directed interaction, between agent types. The dependency of inputs and outputs was determined on the function and task level in the analysis phase. The

agent interaction step focuses on the type of interactions. Since the interaction content is already defined by the inputs the type of interaction must by defined An interaction type can either be an order, or a negotiation or a report type. Different than the other methodologies, the DM-MESD does not dictate the inclusion of FIPA protocols any other protocols. It only poses the question, which type of interaction type should take place between two agent types. Thereby the choice reflects the pursuit interaction-goal of each agent type. For example, if an agent type optimizes a parameter it might need to negotiate with one agent type and to order another agent to do something. The DM-MESD differentiates between three different interaction types:

Order
: Triggering an action of another party.

Report
: A report is an active feedback mechanism informing another party of occurring events or changes. A report does not require an action or a confirmation of its receipt.

Negotiation
: This is an interaction mechanism in which both parties try to optimize the outcome to be in their best interest. The type of protocol or auction model is not important, as those choices are best made by a programmer.

The result of this step are the interactions types shown in the interaction diagram as directed arrows (see figure 3.3).

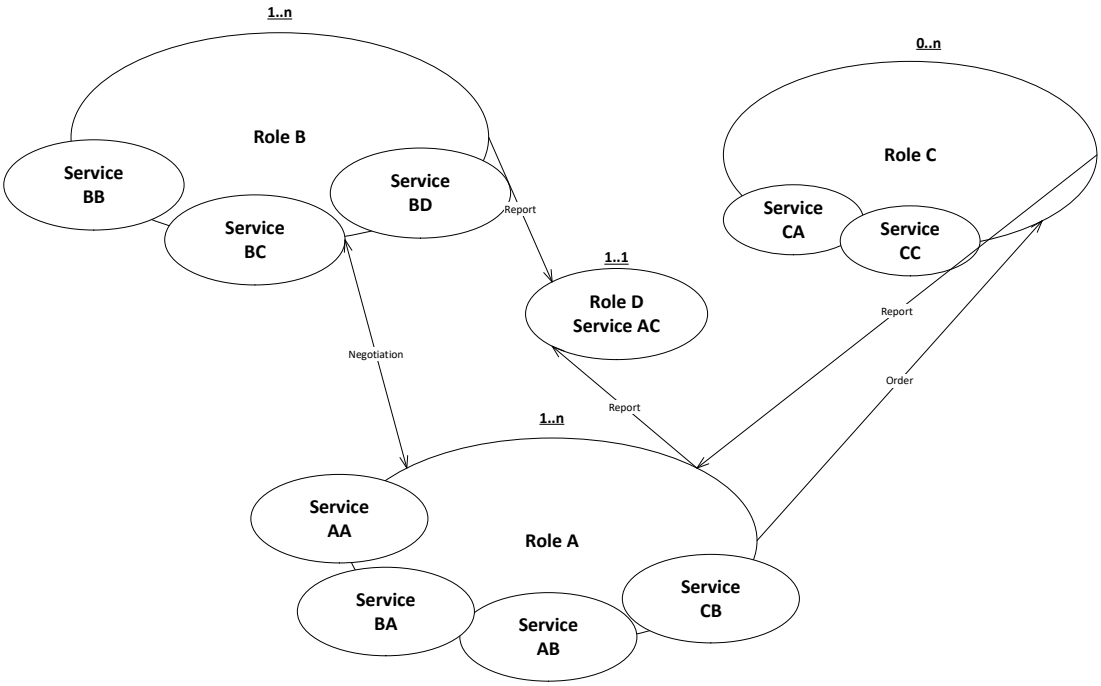

Figure 3.3: Scheme of the interaction diagram, source: own illustration

3.7.5 Optional: Sequence diagram

If desired the interaction of the agent types can be additionally captured in a detailed sequence diagram. A sequence diagram is helpful for the programmer to receive an artifact in which every parameter and value of the interaction is summarized.

CHAPTER 4

Design Process

In this chapter the DM-MESD is being used on the integrated tool cycle of the TU Wien Pilotfabrik. Thereby the application of the DM-MESD and the development of a multiagent system is presented step by step. Before the DM-MESD is applied on the integrated tool cycle the industrial setting of the TU Wien Pilotfabrik is being described. This detailed descriptions of the industrial shall grant the reader better comprehension. The rest of this chapter follows the structure of the DM-MESD. The application of the transformation phase is described in the next chapter, as it is the introduction to the final MAS design.

4.1 Presenting the industrial setting of the TU Wien Pilot Fabrik

The TU Wien Pilotfabrik [WIE] is a demonstration factory, in which industrial Industrie 4.0 approaches are being realized and tested by the TU Wien and it partners. It also serves as a learning area for students and grants them an insight into industrial solutions and processes. The aim of the TU Wien Pilotfabrik is to offer the students and the scientific staff an industrial environment, which reflects today's state-of-the-art production. The production area includes industrial size machine tools, state-of-the-art communication and solutions. Utilizing the industrial setting of the TU Wien Pilotfabrik brings the benefit of working with common industrial solutions. It serves as an example of an industrial shop floor. Of course, it lacks complexity compared to a manufacturing plant with hundreds of workers, and machines and numerous departments. But for this very reason it is a suitable and manageable system to start with. To describe the setting of the TU Wien Pilotfabrik first the process relevant hardware, and then the software settings are being presented. This knowledge is usually possessed by the process stakeholder. This proves that the mapping process for the manufacturing domain on the MES layer requires stakeholder knowledge. The target designer of the DMMD are those stakeholders, which are industrial or mechanical engineers or foremen with training. The complexity and domain specific knowledge, which are presented here briefly as industrial hardware and software settings, confirms the choice to address industrial and mechanical engineers as target designers.

4.1.1 Industrial Hardware

The core of the process relevant assets is the EMCO MAXXMill 550 machine tool. It has a Siemens control, the SINUMERIK 840DSL-711 Version 04.08 + SP 02 including the Siemens

CNC programming software SHOPMILL. This 9,2 tons heavy machine tool offers five-sided machining in a single workpiece set-up [EMC]. It has an integrated tool magazine with up to 30 pockets for cutting tools with HSK clamping adapters. Its working table is 600 millimeters long and 600 millimeters wide and is conceptualized for a maximum load of 250 kilograms. The motor spindle of this milling center covers speeds between 50-15000 rpm. The MAXXMill 550's drive power performs with 34,5 kilowatt and a maximum torque of 110 newton meters. Its rapid motion speed reaches up to 3 meters per second on each axis (X/Y/Z), offering on each axis a feed force of 5.000 newton. Those characteristics underline the before stated importance of the TU Wien Pilotfabrik as an industrial use case, reflecting the current state of the art.

To load and unload the MAXXMill an IRB robot is adjusted right before the milling machine tool. The robot is controlled by the machine tools CNC control and is part of the production cell. The workpieces are carried on the workpiece transporting palette, which is fixed on an Automatic Guide Vehicle (AVG). The AVG signals its arrival using the industrial network through the IRB robot to the MAXXMill, which then triggers the loading or unloading process. Since the AVG and the workpiece handling are just marginally relevant they are not further described.

Cutting tool components are highly standardized, it is therefore not relevant that those components in the TU Wien Pilotfabrik are not produce by the same manufacturer. In fact, the TU Wien Pilotfabrik possesses a high number of different cutting tool types and brands. The assembled cutting tools are either being used in the MAXXMill tool magazine or being in the storage. This corresponds with the actual distribution of cutting tools and components in an industry setting.

Next to the cutting tool storage is a workplace for an employee to assemble the tool components into cutting tools. In general, it can be said, that an employee will be informed about the required components and the workplan to assemble them by a terminal. The terminal consists of an RFID reader and writer unit and a tablet with a Zoller TMS Gold App connected through the Zoller Web-services interface with the TMS Gold data server ("pilot 3.0") at the presetting device. The RFID reader and writer station has been installed for the RFID-tags on the tool holder. The tags have been inlaid on the tool holder and sealed with an epoxy resin layer. The RFID tag has enough storage, but will only carry the tool inventory number, which refers an assembled cutting tool in the pilot 3.0 database. The tool inventory number is a unique cutting tool identifier, which traces a cutting tool instance. The MAXXMill also possess a RFID reader and writer terminal right next to the loading door. As described in subsection 2.1.3 machine tools can be connected via a directed network to the presetting device, in order to receive the corrective geometric data of each tool.

As a presetting device Venturion 450 from Zoller is used [Zol18]. It is equipped with the software »pilot 3.0« cutting tool database and a graphical user interface. This presetting device is equipped with a RFID reader and writer equipment. It can automatically detect cutting edge forms, zero-point monitoring and run an automatic inspection of the cutting edges. With its integrated tool management system «pilot 3.0» tools and adapters can be managed, identified and their inspection record be managed. Data from the pilot 3.0 database can be exchanged with other systems via the Zoller web-service. The presetting device offers scanning functions for cutting tools and saving the envelope contour in different drawing formats. The tool contour can be determined with the »lasso« measuring program and undergo the automatic target-actual-comparison. This data is exchanged with machine tool control to prevent any collision during the machining. This automatically approaching and focusing measurement program guarantees user-independent precise measurement. The optical measuring head is moving automatically, using CNC axes to measure the assembled tool clamped on to the identified tool holder, so

the overall measurements of the cutting tool are written automatically into the tool inventory number database.

4.1.2 Industrial Software

The exchange of cutting tool information between the presetting device and the machine tool at the TU Wien Pilotfabrik is solved querying the setting sheet data through the Zoller web-service directly from the database into the machine tool. This is accomplished by connecting the machine tool through an OPC-UA client on top of the Sinumerik control. For further information about the technical realization see [FTP19]. The exchange of the current magazine load data between the machine tool and the tool database has no standardized interface. In fact, there is no current solution that would allow to query the data from the machine tool control to the presetting device or the tool management database. In the TU Wien Pilotfabrik, a by-passing solution is deployed, by utilizing a combination of OPC-UA and Siemense CreateMyInterface. The latter technology is needed to retrieve internal machine tool control data. This solution is a temporally work around, until Siemens realizes its OPC-UA compatible machine tool control.

4.2 Evaluation Phase

Requirement 1: Process within a similar investigation area
This requirement is clearly fulfilled, as the discrete manufacturing in a small size production is being addressed. The application domain is limited to the MES layer or higher.

Requirement 2: No real-time requirements
The limited perspective on the integrated tool cycle does not require real-time responses. The machining process itself intersects with the real-time requirement, but since the machine tool is considered as a closed system and its internal control functions are not considered, this requirement is fulfilled.

Requirement 3: Processes with a defined start and end
The integrated tool cycle starts with the release of a shop order and ends with the completion of its machining. This requirement is therefore fulfilled.

Requirement 4: No data interruptions during the process
During the groundwork for this thesis, the data interruptions between the machine tool, the tool management system and the production control were resolved. As in subsection 2.1.3 described, the current magazine load can be accessed at any given time. Further is the linkage issue between the presetting device and the machine tool resolved by implementing an interface with the Zoller webservice. With this interface setting sheet data can be exchanged with the machine tool. The cutting tools in the machine tool are being identified as tool instances by using RFID tags on the tool holder of each tool, storing its tool inventory number. The machine tool and the presetting device both have RFID-terminals, to access the inventory number on the tag. In this manner the tool instance data, such as the remaining lifetime, can be accessed and manipulated via the Zoller webservice. This requirement is therefore seen conserved as fulfilled.

Requirement 5: Granular process steps with measurable input and outputs
The activity flow of the integrated tool cycle process (see figure 2.9) and its first introduction in section 2.1.3 show that this process can be divided into process steps. The second part of this requirement is the measurability of each input and output of those process steps. This requirement demands a clear definition of each input and output during the process. During the examination of the integrated tool cycle to solve all data interruptions the inputs and outputs

were defined and are therefore measurable. The measurability is granted by availability of a defined form and quality or by quantity. This requirement is therefore considered as fulfilled.

Requirement 6: Influence on process parameters within the agents' reach
The process parameters such as resources, material and information must be available to ensure the process. The integrated tool cycle has all required entities at its service. Since it accesses to all required inputs it can be assumed that the process influencing parameters must be within the field of action of the process. The future agents cover at least the origin field of action of the integrated tool cycle. This requirement is therefore assumed to be fulfilled.

4.3 Assessment phase

This phase determines if the integrated tool cycle fulfills the requirements of a core process.

4.3.1 Determining the core process, including its boundaries

The overall objective of the integrated tool cycle is to manage the cutting tools (physically and virtually) required for a shop order and to ensure that the right tools will be at the machine tool on time for the machining. This overall objective is measurable, as the required tool lifetime for each tool type must be available at the machine tool. Either this task is being successfully performed, which leads to the starting of the machining, or not. For the tool cycle to be a core process, this overall objective must be achievable through subordinated goals or sub steps. A core process objective must be achievable by dividing it into sequences of sub steps. According to the description of the integrated tool cycle (see 2.1.3) it can be divided into sub steps and their tasks. Figure 2.9 summarizes those sub steps as process steps. A sub step has a defined task or function processing a defined input into a defined output. The sub steps required to achieve the overall objectives are summarized in a coarse manner in table 4.1.

As table 4.1 shows, the determined core process has an overall objective, which is measurable and can be achieved by sub steps. The process boundaries depend on the input required for each sub step. The core process starts with the release of a shop order, and so does sub step 1, including the task "Shop order release". The input of this sub step is the received shop order data. Its outcome starts of the process. So, the shop order data is the first boundary of the core process. A shop order is a partial order of the bigger customer order. It is automatically generated, will not be automatically released. While the ERP schedules the customer orders, the release of shop orders is a task within the MES. The shop order data such as the work plan and BOM, etc. are automatically generated based on the master data located in ERP. However, this data can be manipulated on the MES layer, if required. This practice is used, if the shop order requires a modified BOM or a different work plan. So, the shop order data received here is the highest entity within the core process and marks one process boundary. The shop order data is an artefact that serves as the interface to the ERP or MES layer. As such an artefact it needs to be defined. The shop order data should include the product reference, a production quantity, and a date to be finished.

Sub step 2, which is a calculation and therefore a function determines the gross tool requirement and returns the required tool use time for each tool type to machine the order. Beside the shop order data, this function requires a tool list of the NC program listing all tools required and the exact tool use times that the NC-program plans for the use each tool type. The shop order references the NC-program. The NC-program data can either be another defined process boundary, if stored as part of the product master data in the ERP. Or it can be stored, as it is

Core Process			
Overall objective: required cutting tools (physically and virtually) on time at the machine tool for the machining of a shop order			
Objective measurability: sum of the required tool lifetime [minutes] for each tool type has to be available at the machine tool for the shop order			
Sub steps	**Input**	**Task or function**	**Output**
Sub step 1	Shop order data [Data]	Shop order release [Report]	Event [Event]
Sub step 2	Shop order data [Data] Tool list according NC-program [Data] NC-program tool use time [Data]	Determine gross tool requirement [Function]	Gross tool requirement [Data]
Sub step 3	Machine tool control [Interface]	Request magazine load [Function]	Magazine load [Data]
Sub step 4	Magazine load [Data] Current storage stock [Data] Gross tool requirement [Data]	Determine tool net requirement [Function]	Tool net requirement [Data]
Sub step 5	Tool net requirement [Data] TMS [interface]	Generate picking order [Function]	Picking list [Data]
Sub step 6	Picking list [Data] Tool Components [Physical items] Worker handling [Human labor]	Assemble tool components [Task]	Assembled tool [Physical items]
Sub step 7	Assembled tool [Physical items] Presetting device [Physical item]	Measure and store measurement [Task]	Tool measurement [Data]
Sub step 8	Assembled tool [Physical items] Worker handling [Human labor]	Commission to machine tool [Task]	
Sub step 9	Assembled tool [Physical items] Tool measurement [Data] Worker handling [Human labor] Workpieces [Physical item] Clamping system [Physical item] Machine tool [Physical item]	Tool setup [Task]	Setup accomplished [Event]
Sub step 10	Machine tool Control [data] NC-program [Data] Assembled cutting tools [Physical item] Workpieces [Physical item] Clamping system [Physical item] Machine tool [Physical item]	Machining [Function]	State [Event]
Sub step 11	Machine tool control [data]	Report machining end [Function]	End process [Event] Remaining tool lifetime [data] Processed workpiece [Physical item]

Table 4.1: Core process assessment of the integrated tool cycle

in our case, in a setting sheet data available at the TMS. The NC-program data is seen as a defined artefact marking the interface to the TMS.

Sub step 3 includes the function of requesting the current magazine load. This sub step sets the interface to the machine tool control as a process boundary.

Sub step 4 determines the tool net requirement for each shop order. The tool net requirement is the smallest amount of cutting tools required to machine a shop order, considering the cutting tools currently at the machine and at the storage. Its function requires the magazine load [data], the cutting tool stock at the storage [data] and the previously determined gross tool requirement as input. While the magazine load and the gross tool requirement are outputs of previous sub steps, the current storage stock data marks a new process boundary. The current cutting tool stock must include the inventory number and the remaining lifetime of those tool instances. This artefact is updated directly in the TMS database.

Sub step 5 requires beside the tool net requirement an interface to TMS to generate a picking order using the TMS database.

Sub step 6 is the assembly of tool components, which represents a physical task performed by a human worker. The human worker is in addition to the tool components another required input and marks another boundary.

Sub step 7 requires the previously assembled cutting tools and a presetting device to measure them. Sub step 8 also requires the assembled tools and the worker handling the task. Both inputs have been already listed.

Sub step 9, the "tool setup" requires, five inputs, among those are the physical items, the machine tool, the clamping system and the work pieces. The machine tool is a resource, which can be scheduled and occupied. The clamping systems are, similarly to the cutting tools auxiliary production resources. This input marks another boundary, as the clamping systems is required, but not managed in the core process. Their availability at the machine tool is assumed. Another input in this sub step is the workpiece, which is here also clarified as a physical item. The workpiece is of the type material, as it will be consumed or transformed in course of the value adding process. Like the clamping systems, workpieces are assumed to be available at the machine tool on time. Both decisions are reasoned in the attempt to narrow the core process. The tool cycle is defined as the core process with the objective of providing the "required cutting tools (physically and virtually) on time at the machine tool for the machining of a shop order". Managing and providing clamping systems and workpieces at the machine tool is not part of the core process.

Sub step 10 considers the machining, which is function fulfilled within the machine tool. It requires all inputs from the previous step and in addition an NC-program. The right NC-program is assumed to be available at the machine tool or to be accessed through an internal network.

Sub step 11 includes the function "Report machining end" and marks the end of the core process. Its input requires no additional boundary, but its output which is of an event type marks the end of the core process.

The **process boundaries** are set based on the sub step inputs. The highest boundary is shop order data as an artefact defined in the ERP system. Another artefact, which has its source in the ERP master data is the NC-program tooling data. It is available in the TMS. In the lower MES layer, the process boundaries are set by including or excluding processes or resources. The current tool stock data is provided from outside the core process system and available for the core process in the TMS database. Two resources within the process boundaries, beside the cutting tools and the TMS, are the machine tool and the presetting device.

The machine tool is distinguished resource as it is also utilized by other processes, such as maintenance, material supply, etc. The limited perspective of the machine tool must therefore be emphasized. The physical appears of the machine tool as a production resource is either available or not. The machine tool is seen as closed unit, which interacts with the core process through the machine tool control. The machine tool control offers the core process artefacts, such as the current magazine load state and the reports of the beginning and ending of the machining (sub steps 10 and 11). Both artefacts of the machine tool demarcate the process boundaries as no other task or function within the machine tool are considered. For example, the internal tool changing strategy or any other detail of the machining itself is not relevant for the core process. The perspective of the presetting device is limited to measuring the assembled tools and its connectivity to the TMS to store the measurements with reference to the tool inventory number.

4.3.2 Defining the pursued strategy and its KPI to be optimized

Through resolving the data interruption, the tool net requirement of each shop order can be determined. The tool net requirement is the minimal number of tools required at a given time to machine a specific shop order. The smaller the number of cutting tools, that must be setup for a shop order, the smaller the tooling costs will be. A setup of three cutting tools is less expensive, than the setup up of ten cutting tools. The tooling cost include the tool cost and the tool supply, including assembly, measurement and commission. Reducing the tooling cost, reduces the production cost in two ways. One way is by indirectly reducing the idle time of

the machine tool, as less setups would be required. The other way is by exploiting cutting tools to their optimum and reducing the tooling cost. Since the magazine load of the machine tool can be requested at any time, it might be possible to sort the shop order by their tool net requirement. Finding a schedule with the smallest tooling costs means that less cutting tools must be assembled, and less capital is being tied up in assembled tools.

The **pursued strategy** is to flexible schedule the orders so that the tooling cost is minimized. In this context, the KPI set for the pursued strategy is to sort the orders to achieve the minimal the tooling cost. This KPI shall be achieved by finding the optimal production schedule for each shop order based on its tool net requirements, which depends on the previous order and the current magazine load. The shop orders shall be scheduled depending on the previous shop orders and their required tools, so that the number of new assembled tools will always be the smallest possible. The integrated tool cycle covers all required inputs, so that the minimal tool net requirement for each shop order can be realized.

4.3.3 Rechecking the process boundaries

Since the required inputs for the defined KPI are all covered by inputs required to determine the tool net requirement, they are already within the core process boundaries. However, it must be considered, that a released shop order might include more than one workpiece type to be machined. A shop order could include different workpieces requiring different NC-programs to be executed on different machine tools or working stations. This would compromise our KPI aim to schedule the shop order by the tool net requirement, since one shop order could include several components with their own workpieces and tool net requirements. The possibility in this case is either to adapt the KPI or to adapt the system boundary, which is a designer stakeholder choice. Here the process boundaries are being adapted.

A shop order is not granular enough, as it does not directly refer to the order to be machined. A shop order can include to its work plan, several machining or assembling orders, which require a certain sequence. Since the set KPI is to find a schedule for a machine tool, which offers the minimal tooling cost, the orders can be reduced to machining orders. Before the shop order was defined as the highest artefact and as system boundary to the ERP. Since scheduling shop orders is not sufficient, the new boundary is set on **machining orders**. A machining order would refer to one NC-program and a tooling list. It does not require the completion of any previous order, nor does it include more than one workpiece type, NC-program or machine tool, to be machined. Based on those characteristics, a machining order can be freely scheduled. Its machining takes place at only one machine tool.

To schedule the machining orders in a manner, that would minimize the tooling cost, cutting tools must be used in different NC-programs to exploit their potential. This poses a problem. The remaining tool lifetime can only be estimated, based on the tool load, the feed and the cutting depth. The tool manufacturer offers a so-called catalog-tool lifetime, depending on a pre-defined feed, tool depth, and tool load. The actual tool lifetime and its consumption depend on several factors, besides the ones mentioned above, such as lubrication and span transportation. Both criteria depend on the actual machining circumstances. In practice, the tool lifetime is determined by machining under real circumstances and monitoring the tool wear. A high number of scientific efforts focus on investigating how to estimate the tool lifetime precisely. Besides complex mathematical and computer-aided models, there is a vibrant discussion which machining factors affect the tool lifetime more than others. Since finding an answer to those questions would provide scientific sustenance for several doctoral thesis, this work will not further explore the subject of tool lifetime estimation. However, the pursued strategy is to achieve minimal

tooling costs by scheduling machining orders based on the current magazine load. To do so, the use of cutting tools in different NC-programs is essential. It will be therefore assumed that tool lifetime consumption can be set into relation with the NC-program related tool load. It will be therefore assumed that the NC-program related lifetime is expected correctly, but also that it stands in a linear relation to the catalog lifetime. This assumption enables the use a cutting tool in different NC-programs by transforming its remaining lifetime.

4.4 Analysis Phase

The analysis phase deepens the perspective on each sub step, and breaks them down into other sub steps, if required. The main part of this analysis phase is the textual description.

4.4.1 Textual description of the process

A machining order, instead of a shop order was chosen in order to realize our KPI of scheduling the orders according to their tool net requirement. The process begins therefore with the arrival of a released machining order into the system. With this event the system receives the machining order data. Such machining order data must at least include the production quantity and a reference to the NC-program to refer to a tooling list. An additional machining order identifier and production deadline would also be suitable. With the reference to the NC-program the related setting sheet can be found in the TMS. A setting sheet, also called setup sheet, is a tooling list of cutting tools required for an NC-program on a machine tool. This setting sheet lists all required tool types, which are identified either by a unique number or a unique name. However, a setting sheet does not reflect the actual number of tools required for the future machining order. In fact, in practice a setting sheet just describes which tool types and components are required for the specific NC-program. It does not specify the number of the replacement tools (sister tools) required to process a machining order. In this setting the tool types are identifies by a tool number, called ToolID. Only three kinds of information are commonly used in setting sheets: the NC-programID, the machine tool and the tooling list. While transforming the tool cycle, the specific tool use time of each tool type to machine one workpiece was added into tooling list of the setting sheet. With this extended tooling list of the setting sheet and the workpiece quantity, the gross tool requirement can be calculated for each cutting tool type. The result of this calculation is the gross tool requirement, a minute value for each cutting tool type reflecting the required tool use time to machine the complete machining order. The gross tool requirement, which displays required minute value for each tool type, is at the same time the biggest tool requirement. Its order specific value will be reduced by the remaining lifetime of the cutting tools available on the shop floor. In the meantime the magazine load can be requested from the machine tool to determine if fitting tool type instances are currently available. The result of this request is a tooling list, summarizing the tool types and the tool instances by their unique inventory number and their remaining tool lifetime. Since required tool instances could also be available in the storage, a request is sent to the TMS, which always has the newest stock data. This request will be limited to the tool types from the tooling list of the setting sheet. Its result is a tooling list from the storage listing tool types, tool inventory numbers and the remaining tool lifetime. Those two tooling lists from the machine tool and the storage, together with the gross tool requirement are the basis to calculate the net tool requirement. The net tool requirement is the smallest required amount of cutting tools required to machine the order, reflecting the current state of the shop floor. It reflects the current state of the shop floor, as it considers the tool currently at the machine and at the storage. Like the gross tool requirement, the net tool requirement is a minute value

for each tool type required to machine the order. On this basis a dynamic setting sheet for the specific production quantity and the current state of the shop floor can be generated. This new setting sheet includes the minimal amount of tool instance required to machine the order and will be further called dynamic setting sheet as it reflects the minimal tool requirement, which shall be covered by newly assembled tools. This dynamic setting sheet is, other than the common setting sheet, order specific, as it considers a specific machining order and the specific tooling situation on the shop floor. This dynamic setting sheet would be generated by the TMS or through the Zoller web service by generating a new setting sheet ID with the amount of tool instances required from each tool type. To determine from the net tool requirement (a minute value) the tool instances quantity, the following information would be required: the tool lifetime of a new cutting tool, the critical tool life time and the tool lifetime according to the NC-program. The latter value describes how long (in minutes) a new cutting tool would last, when machining with a specific NC-program. The tool lifetime of new cutting tool is either a catalog value, set by the manufacturer or an NC-program specific value. It describes the tool lifetime under predefined cutting circumstances, such as tool load and cutting feed, which vary in each NC-program. Now a picking order is created on the basis of the dynamic setting sheet. A picking order is a bill of materials. It lists the components required for assembly for each tool instance. This picking order is handed to the worker, who collects all components and assembles them to tools. Those cutting tools are mounted to a tool holder of the type HSK, which fits the machine tool coupling. Each tool holder has a RFID tag inlaid. Now that the cutting tool is assembled and mounted onto a tool holder, it becomes a tool instance. As such it now possesses an inventory number, which will be assigned, either by the worker or by an number generated by the TMS. This inventory number is unique and refers to dynamic tool data. Dynamic tool data is data that belongs to tool instances and is being changed during the tool cycle. Such data is, for example the remaining tool lifetime, geometrical measurement values, last use time, etc. When all required tools have been assembled and their tool inventory number has been stored into the TMS, the tools will be delivered to the presetting device with the worker. Now each cutting tool instance is identified by the presetting device with its inventory number on the RFI tag and measured. The measurements are stored directly by the presetting device to the TMS and are accessible for the machine tool on a request basis. To enable a remote call for the tool data from the machine tool, the Siemens Create MyInterface and OPC UA is used. To exchange data from the TMS to the machine tool and back, either HTTP or XML based solutions (Zoller Web Service) are used. This interface is further addressed as the TMS interface. Right after this gross tool requirement is determined, the current magazine load is checked through Create MyInterface and the net tool requirement is determined. The measured data is stored in the central TMS databased and linked to the tool ID. When all tools are measured, the worker delivers them to the machine tool. The machine tool identifies the tool instance by the inventory number written on the RFID tag and requests the tool instance data. The machine tool can call up the required data through the TMS interface. Now the cutting tools are setup by inserting the tools into the machine tool spindle, which assigns them a magazine pocket. The machine control then assigns the magazine position freely to the tool instance and codes the magazine position to the tool type. Now the machining and therefore tool use can start. After the machining ends, the lifetime value of each inventory number is updated remotely in the TMS, through the web service interface. The machining ends when all workpieces are machined.

4.4.2 Detecting essential process entities or beings

The aim of this methodical step is to determine essential process entities from the above stated textual description. According to table 3.2 the entities are divided into two groups, a concrete and an abstract concept group. To determine the function and tasks of the textual description, the focus shall be on verbs used to describe the process flow. Those verbs are marked by underlining them. These underlined words are not only limited to the verbs themselves, as pronouns are required to clarify the task or function. For example, not only "update" is underlined, but also the object that should be updated. Those underlined verbs and their pronouns were then listed in table 4.2. Their abstract concept type has not been defined yet.
In the next step of the process concrete objects, that are required to fulfill the tasks or functions, are listed in the table. It is in most cases evident, which concrete concept types influence the task or function. Concrete concepts are either resources, material or information required to fulfill a task or a function.

With the tasks and functions filled in the center of the table 4.2 the required inputs can be listed and categorized by assigning concrete concept types. The process starts with "Beginning process", which is triggered by the arriving information "Released machining order". An information is a concrete concept type merging data, state and events. A released machining order is an event and therefore of the type information. Since "Beginning process" requires only the event, it is a function.
While the functions and task have been underlined in the textual description the object required have been emphasized by boxing them to aid transparency. The first such object is the machining order. The machining order is an information and includes the following parts: machining order name, NC-program No. and a production quantity. While the machining order is described as a complete entity in the textual description, the table only considers its required aspects individually. In this manner the table is completely mapping the entire process in detail. At this point it is noted that "Assign magazine pocket" or "Machining" are both functions, although they utilize and occupy a resource. This is reasoned in the previously defined process boundaries. It was stated that the internal processes of the machine tool are not concerning the process. This perspective corresponds to the assumption that the NC-program is always available at the machine tool (see process boundaries). The machine tool is therefore seen as a closed system, fulfilling a function and only requiring itself, the machine tool, to do so. The same goes for the workpieces and the clamping systems. All three concrete concepts have therefore not been listed in the table.

4.4.3 Describing process entities a glossary

This methodical step describes the process entities, that need further explanation. Thereby the stakeholder or designer decides for which process objects a further description would be beneficial. Here the machining order, the setting sheet, the tool lifetime and the dynamic setting sheet, require further explanation. However, at this point it must be stated, that no task or function should require a further description in the glossary, since they have been broken into their smallest possible pieces. During the KPI definition the machining order was chosen to be used instead of shop orders, to ensure a free scheduling of those orders without mutual interference. The understanding of such a machining order is defined in the glossary as follows:

A Machining order is limited to a workpiece type and a processing NC-program. A released shop order, on the other hand can trigger a chain of activities required to fulfill the shop order. So does a released shop order for one engine release other subordinated shop orders, such as one each for the pistons, the piston rods, the engine block etc. The machining order can be easily

Process entities		
Input [Concrete types]	**Task or function**	**Output [Concrete types]**
Released machining order[Information] Released machining order[Information] Machining order name [Information] Production quantity [Information] NC-program no. [Information]	Beginning process [Function]	State [Information] Machining order data [Information]
Machining order data [Information]	Get NC-program no. [Function]	NC-program no. [Information]
Machining order data [Information]	Get production quantity [Function]	Production quantity [Information]
NC-program no. [Information]	Get setting sheet[Function]	Setting sheet ID [Information] Setting sheet tooling list [Information]
Production quantity [Information] Tool list [Information]	Calculate gross tool requirement[Function]	Gross tool requirement [Information]
Machine tool control [Resource] TMS interface [Resource]	Request magazine load [Function]	Magazine load tool list[Information] Inventory no.[Information]. Rem. tool lifetime [Information]
TMS interface [Resource]	Request warehouse tool stock [Function]	Stock tool list[Information] Inventory no.[Information]. Rem. tool lifetime [Information]
Gross tool requirement [Information] Magazine load tool list[Information] Stock tool list[Information]	Calculate net tool requirement [Function]	Net tool requirement [Information]
Net tool requirement [Information] Catalog tool lifetime[Information]	Calculate tool quantity [Function]	Tooling list net requirement [Information]
Tooling list net requirement [Information] TMS interface [Resource]	Generate dynamic setting sheet [Function]	Dynamic setting sheet [Information]
Dynamic setting sheet [Information] TMS interface [Resource]	Generate picking order [Function]	Picking order [Information]
Picking order [Information] Worker [Resource]	Collect components [Task]	Components [Resource]
Picking order [Information] Worker [Resource] Components [Resource] Workstation [Resource]	Assemble components [Task]	Assembled tools [Resource]
Assembled tools [Resource] Worker [Resource] RFID-tag [Resource] RFID-terminal [Resource] TMS interface [Resource]	Assign inventory no. [Function]	Assembled tools [Resource] Inventory no. [Information]
Inventory no. [Information] TMS interface [Resource]	Store inventory no. [Function]	State [Information]
Assembled tools [Resource] Worker [Resource]	Deliver tools to presetting device [Task]	State [Information]
Assembled tools [Resource] Inventory no. [Information] Presetting device [Resource] Worker [Resource]	Measure tools [Task]	Measurement [Information]
Inventory no. [Information] Measurement. [Information] TMS interface [Resource]	Store measurements [Function]	State [Information]
Assembled tool [Resource] Worker [Resource]	Deliver tools to machine tool [Task]	State [Information]
RFID tag [Resource] RFID terminal [Resource] Machine tool control [Resource]	Read RFID tag [Function]	Inventory no. [Information]
Inventory no. [Information] Machine tool control [Resource] TMS interface [Resource]	Request data [Function]	ToolID [Information] Tool lifetime [Information] Measurement [Information]
Assembled tool [Resource] Worker [Resource] Machine tool [Resource]	Insert tools into machine [Task]	State [Information]
Machine tool [Resource] Machine tool control [Resource]	Assign magazine pocket [Function]	State [Information]
Machine tool [Resource]	Start machining [Function]	State [Information]
Machine tool control [Resource]	End machining [Function]	State [Information]
Machine tool control [Resource] Inventory no. [Information] Remaining lifetime. [Information] TMS interface [Resource]	Update tool lifetime [Function]	State [Information] Inventory no. [Information] Remaining lifetime. [Information]
Machine tool control [Resource]	Process end [Function]	State [Information]

Table 4.2: Process entities of the integrated tool cycle

managed and recalculated since it is limited to one machine tool as a production resource. A machining order is a realized routing step consisting of one NC-program, which is fulfilled by the machine tool as the production resource. The cutting tools and the clamping system are in this context, auxiliary production resources, assigned to the routing step (NC-program). The maximum lot size of a machining order is defined in the product or component master data and sized to fit the machine tool capacity. This capacity is not only limited by the machine tool working area, but also by the tool magazine capacity, since machining without an additional setup must be guaranteed. The NC-program, as a specified work plan for a machine tool type, must be executed on a machine tool matching its type. The machining order presented here already contains the NC-programID. The machining order contains a unique machining order ID, the quantity of the workpieces to be processed.

A Setting sheet, which is also called setup sheet, is a tooling list of cutting tools, required for an NC-program on a machine tool. In practice, a setting sheet just describes which tool types and components are required for the specific NC-program. It does not specify the number of the sister tools, that are required to process a machining order. However, to calculate the gross tool requirement, the individual tool use time for each tool type is required. To scale this tool use time for the quantity of the order, this tool use time must capture the per workpiece duration. For each cutting tool in the tooling list of the setting sheet a tool use time per workpiece is set. This per workpiece tool use time is saved as a long comment in the tool specific data set of the setting sheet. The thereby stored values are calculated due to the desired feed and tool load. Those tool use times per workpiece are stored in a comment data field, since no explicit data field is covering this issue yet. With this data the gross tool requirement of each machining order can be determined. Since the literature and the industrial practice have no standardized setting sheet structure, its structure depends on customized choices. Only three kinds of information are commonly used in setting sheets: the NC-programID, the machine tool and the tooling list. Since the net tool requirement shall be determined with the setting sheet more data is required. This data is divided into three groups, one is the machining order related data, the second is the NC-program related machining data and the third is the extended tooling list required to fulfill the NC-program. The setting sheet data provided by Zoller TMS Gold is used and for each cutting tool the tool use time per workpiece is added to it. This minor adaption enables calculating the order related required tool use time for each tool, and as a further consequence the gross and net tool requirement.

The dynamic setting sheet enables the usage of the tool data stored in the TMS. With a dynamic setting sheet, easily a picking order can be generated, which includes the bill of material for each tool type listed. The dynamic setting sheet is used to list only the required tools, after considering the available tools at the machine tool and in storage. While the traditional setting sheet, is used as a template to calculate the gross tool requirement, the dynamic setting sheet serves as a template to generate a picking order.

Tool lifetime: During rechecking the boundary, it was assumed, that the NC-program related lifetime is correct. It was further assumed, that the NC-program related tool lifetime must stand in linear relation to the catalog lifetime. Both assumptions enable transforming the remaining lifetime from one NC-program to another. This assumption enables using the same tool instance in different NC-programs. It is not yet elaborated, how to calculate this transformation. This aspect must be considered in the upcoming progress.

4.4.4 Optional: coarse activity diagram

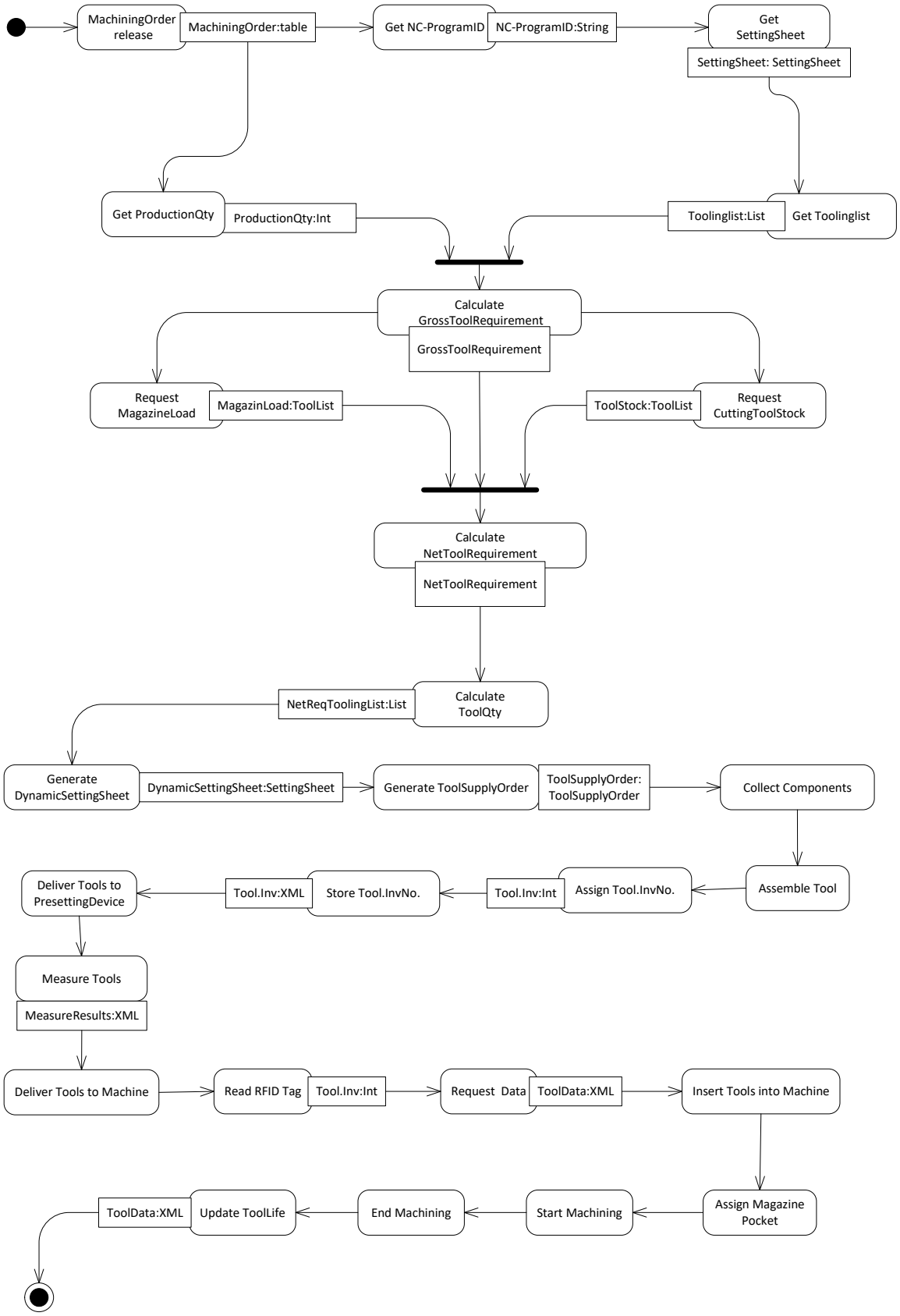

Figure 4.1: Tool Cycle of the TU Wien Pilotfabrik, described in an UML activity diagram

4.5 Abstraction phase

Now that all required tasks and functions are captured, services deriving from those tasks and functions and new services are being identified. First, the services deriving from the process tasks and functions are described. Afterwards, new services supporting the KPI are designed.

Service Identification

Machining Service

The machining service is unique and brings in a value. It starts with the core functions "Start machining" and "End machining". The function "Assigning a magazine pocket" does not count as a service by service rule 1, as it does not add value into another party. In fact, assigning the magazine pocket is a function within the machine tool, which adds no value outside the machine tool. Therefore, this function is added to the machining service, as it supports it. "Read RFID tag" and „Request data" are both functions supporting the machining service. Both have no additional value outside the machine tool and will be added to the machining service. Between those functions and the "Assign magazine pocket" function is the "Insert tools into machine" task. This task is clearly required for the machining to begin. It does however not belong to the machining service. This task will stand alone as a service.

Setup service

The worker inserting the assembled and required cutting tools provides something of value to another party, the machine tool. Its service is solitary and cannot be substituted by any other party. Inserting the tools into the machine also satisfies the service rule 4 as it has a beginning and an end. This service only consists of one task and is named the setup service.

Commissioning service

Another service in this core process, is the commissioning service provided by the worker and consisting of two tasks, "Deliver tools to presetting device" and "Deliver tools to machine tool". Both tasks are not replaceable and provide a value to the other parties. Both tasks have been summarized as in the next phase, they would have been, due to their joint resources and familiarity. Since both tasks cover the same functional range and are both executed by the worker, they have been directly summarized.

Assemble tool service

Another service is the assemble tool service, which combines the continuous tasks and functions from the collection of components to the storing of the inventory number.

Measuring service

The next service is the measuring service. It combines the task "Measure tools" and the function "Store measurements". The combination of both offers the outside a complete service of measurement.

Determine net tool requirement service

The last service deriving from table 4.3 is the determine net tool requirement service. This service combines only functions and processes therefore only information. Provides the calculation of the net tool requirement for each available position in the machining queue.

Process entities		
Task or function [Types]	**Service**	**Offered value**
"Read RFID"[Function] "Request data" [Function] "Assign magazine pocket" [Function] "Start machining" [Function] "End machining" [Function]	Machining service	Machining the order
"Insert tools into machine"[Task]	Setup service	Setup of required tools
"Deliver tools to presetting device" [Task] "Deliver tools to machine tool" [Task]	Commission service	Transporting physical items
"Measure tools"[Task] "Store measurement"[Function]	Measuring service	Measuring physical tool Generating corrective geometrical data
"Collect components"[Task] "Assemble components"[Task] "Assign inventory number"[Function] "Store inventory number"[Function]	Assembling tools service	Assembling cutting tools Generating tool instance
"Update tool lifetime"[Function]	Update tool lifetime service	Transmitting the current remaining lifetime
"Get NC-program no."[Function] "Get production quantity" [Function] "Get setting sheet" [Function] "Calculate gross tool requirement" [Function] "Request magazine load" [Function] "Request warehouse tool stock" [Function] "Calculate net tool requirement" [Function] "Calculate tool quantity" [Function] "Generate dynamic setting sheet" [Function] "Generate picking order" [Function]	Determine net tool requirement service	Minimal tool requirement [Information]
"Beginning process"[Function]	Inquiring Offer service	Beginns interaction Transmits values Requires offer
"Calculate production costs"[Function] "Calculate order execution time"[Function] "Post offers"[Function]	Proposing offer service	Turns position into production cost Turns position into order execution times Offers possible positions machining schedule
"Evaluate best offer"[Function] "Report Deny"[Function] "Report Acceptance"[Function] "Termination"[Function]	Accepting offer service	Pursuing KPI strategy Denying offer Accepting offer Report to a higher system and terminate
"Update Machining Queue"[Function] "Evaluate time preceding order"[function] "Evaluate time until setup"[function] "Managing the tools in the magazine" [function]	Managing machining service	Knowledge of the current schedule Ensuring machining position is available Defining tools for dismounting
"Calculate time tool supply" [Function] "Calculate until setup" [Function]	Managing setup service	Ensures on time tool supply Prevents idle time due to missing tools

Table 4.3: Summary of services

Update tool lifetime service

During machining, the remaining tool lifetime is reduced. The updated tool lifetime is signaled after finishing the machining. Although the function "Update tool lifetime" takes place within the timeline of the machining service, it is chosen to be a separate service. This is reasoned as the updated tool life within the machining service would not have value for another party. It is a solidary and unique service, although its source might be substitutable. The tool life could also be estimated instead of being updated directly by the machine control.

New Services based on the KPI

The above described services cover the complete functionality of the core process. However, they do not cover the KPI, which is to minimize the tooling costs by the optimal scheduling of the orders. Since there is no task or function available to build the service on, new services must be developed. In a hierarchical control structure, the scheduling mechanism would be a formal function and dictated the sequence. But since the minimal tooling cost sequence shall be achieved by a multiagent system, negotiation pattern must be installed. An example of such a

negotiation pattern could be an auction mechanism.

In such an auction, the party that is willing to pay the most would win the bid. But this negotiation pattern is not suitable for realizing a machine tool schedule to realize the minimal tooling cost. Another negotiation pattern would be a buyer-seller-system, in which the scheduling is achieved by one party inquiring and another party offering. The party who inquires can choose whether it is in its interest to accept or to deny the offer. This pattern is suitable for this case and can be designed by establishing new services. To pursue this pattern, a "proposing", "accepting" and a "scheduling" service are required. The new scheduling service should include a mechanism to determine the most suitable sequence of machining orders. But since the aim is to design a flexible process control by a multiagent system the scheduling mechanism must be separate from the execution mechanism.

The scheduling services only oversees if at any moment the machine utilization still is the best possible solution and suggest a rescheduling, if this is not the case. But whether this rescheduling is being executed or not depends on the consent of two parties, the bidder making an offer and the buyer, accepting, or denying the offer. The new services of the type "Proposing" and "Accepting" shall lay down the execution mechanism. Since the aim is an optimized schedule with a minimal tooling cost, which is reflected in minimal production cost, it is comprehensible that the buyer party will be represented by machining orders.

To start the process an arriving machining order is required. Its analogue for a buyer-seller-system would be an arriving inquiry. This arriving inquiry states some individual goals, such as the order execution time or the maximum (production) cost, etc. This inquiry is answered by proposing one or more offers for different positions in the machining schedule. At the beginning new services have no tasks or functions and consist only of a name and a value they must optimize. Their further tasks and functions will be defined in the next steps. This is possible, since their functions and tasks are not yet bound neither to any concrete nor to an abstract process entity and can be defined freely, to fit into the further multiagent system. Now the scheduling, bidding and buying services are being discussed in detail focusing on their content and which functionalities they must have including their inputs.

Inquiring offer service

The inquiring service is the only new service, which includes an existing function; the "Beginning process" function. The inquiring service is triggered by the machining order entering the system. The machining order shall actively seek for the best spot in the machining schedule. It could be debated, that the inquiring offer service is rather a function than a service, but it brings a value to another party. This other party is whoever oversees the proposition of an offer or the scheduling service. According to the service rule 1, it adds something of value to the system. Additionally, service rule 3 is fulfilled, as this service cannot be substituted. Service definition rule 2 and 4 are bound to the tasks and functions that this service will include. So, what will be the task or function of this service? The inquiring an offer service informs the system of the arrival about a new machining order. It transmits information of the machining order. Both aspects are covered by the "Beginning process" tasks, which result in the machining order data. But beside this the already defined data, a new data set is required to engage in a buyer–seller scenario. There are two possible ways to trigger this scenario, either the buyer presents a target value, that shall be meet or it provides parameters and waits for several offers. Since the setting has only one machine tool, presenting a target value does not fit the use case. The benefit of this scenario can be best exploited, if the buyer can connect several sellers with a target price, to see who is capable of meeting it. Since the setting has one machine tool and therefore one machining schedule, different positions in the machining queue (schedule) can be offered. So, the

possible scenario is to request several positions in the machining schedule, and to then choose the one with the lowest price. The advantage of this scenario is also that the no further data for the inquiry service needs to be exchanged. Considering service rule 4, this service has a beginning and end, in fact, it will only be enrolled at the arrival of the machining order. This is a design choice since machining order shall not request proactively a better machining queue position, after already accepting an offer.

Proposing offer service

Now the proposing offer service is the second required interaction, between two parties trying to achieve consent. It can be said without any doubt, that this is a service of value to another party. The design choice is that the offering party makes an offer for each possible position in the machining schedule to the new arriving machining order. The proposing offer service should end at some point. This decision is made since machining orders shall not wait until their desired value is met, but rather choose to accept or deny the offer. In fact, the offering party should limit its offers to a specific time, so that the in inquiring party receives all possible offers available at the moment of request. The resource capable of fulfilling what the proposal promises, a position in the machining queue, is the machine tool. To offer a machining queue position only the current schedule information is required, not the execution of the proposed machining. Further, the offering party must be capable of turning the variable machining schedule spots into parameters comparable with each other. Such parameters could be production dates, order execution times, process times, or production costs, etc. Those value shall serve the buyer side as decision-making parameters, reflecting, if the offer is desirable or not. So, the offering service must have a set of information, enabling it to turn a spot in the machining schedule into selling parameters. To keep it simple only two parameters are chosen to be offered by the seller, the production cost, and the order execution time. So, each possible machining queue position has at the time the request is made a known production cost and order execution time. Those values are influenced by the preceding orders as well as by the requesting order. Therefore, both values directly reflect if a machining order sequence is desirable of not. During the inquiring service it was stated that a machining order can only once request offers. This has the benefit that the next order will be offered positions in an already existing queue, instead of completely rescheduling the machining queue. Completely rescheduling the machining queue is complex, and would not meet the demands of the shop floor, as machine idle times occur due to the rescheduling. The order execution time, as its name indicates, is the time a machining order must wait until it has been completely machined and finished. It describes the time it takes from the arrival of the machining order to completing its machining. The offered production cost should somehow reflect the smaller tooling costs. In other words, it should put more weight on a smaller tool setup than on other aspects. This decision is caused by the set KPI strategy. Therefore, the main aspect of the production cost should be the cost caused by changing cutting tools. Since machining orders can vary in size, and a bigger production quantity would also cause higher handling cost and higher machining cost in the industrial practice, both aspects are considered. The handling cost derives from the quantity of workpiece per machining order. While the machining cost results from a multiplication of the workpiece quantity and the machining hour rate. It derives from the time that the machining order occupies the machine tool. The production cost includes the tooling cost, the handling cost and the machining cost. The tooling cost has a higher impact on the production cost than the other two cost aspects. So, this service does not only make an offer for each possible machining queue position at the moment of, but also determines the order execution time and production cost for these positions.

Accepting offer service

The decision making of the buyer is based on two parameters, the order execution time, and the production costs. The buyer must have a mechanism to evaluate the most suitable offer by this set. If two offers exist, for example offer A and the other Offer B, and it is assumed that offer A has a lower production cost and a shorter order execution time. Then the offer A is clearly the more desirable offer. But this comparison lacks validity if offer A has the lower production cost but the longer order execution time. The accepting offer service requires therefore a function to choose a suitable offer. The proposing offer service will derive the smallest tool setup for a smaller production cost. This leads us to the perspective that the accepting offer service should focus its interest on gaining the lowest production cost possible. However, defining only this focus might lead into accepting a theoretical infinite order execution time for the lowest possible production cost. This aspect might not be obvious if only four machining orders would be in the system simultaneously, but it is for 20 machining orders. If the buyer accepts any given position offered only due to the price, there would be no mechanism to prioritize machining orders, which is a common practice in the industry. The case might occur, that a machining order with a high priority arrives, for which the production is willing to pay whatever cost only to machine it as fast as possible. The accepting offer service must also cover this mechanism, which leads to another function and its required information. A parameter is required to put production cost and the order execution time into relation. To keep it simple, a time leeway factor is chosen. It is a factor defining how big a time leeway can be to achieve a lower production cost. If an accepting offer service deals with a leeway of 1.65 it could extend the order execution time up to 65% of the original value, only to realize the best production price. The target parameters are the maximal production cost, the maximal order execution time, and the time leeway. Based on those parameters, the accept offer service must chose if one offer can be accepted or all offers must be denied. If no offer can be accepted, this service must have a function to handle this case. Such a function could be an error handling or a termination with a report file.

Managing machining service

While the proposing offer and the accepting offer services ensure that the optimal machine utilization with the lowest tooling cost is found, a service is required to ensure the smooth operation of the machining. The proposing offer service calculates the realizable production cost and order execution time of each position, while the managing machining service decides which position is available for scheduling by considering machining restrictions, which must be respected. One constraint is that the next order for machining cannot be changed anymore. This reflects the situation on the shop floor as the tool supply takes time. Another task of the managing machining service is managing the tools in the magazine. It schedules which tool is up for replacement, and monitors that no other cutting tool is dismounted.

Managing setup service

The managing setup service provides the managing machining service with the information about when a setup can take place and how long the tool supply might take. To flexible schedule machining orders requires to postpone or prepone tool setup and supplies. While doing so, it must be ensured that an offered machining position includes only positions, which offer enough time until the setup starts, so that the tools can be prepared, and the tool setup can take place on time. In figure 4.2 all services have been captured. Here the new services are highlighted by a green background. The Inquiring Offer service, is the only new offer including an existing function.

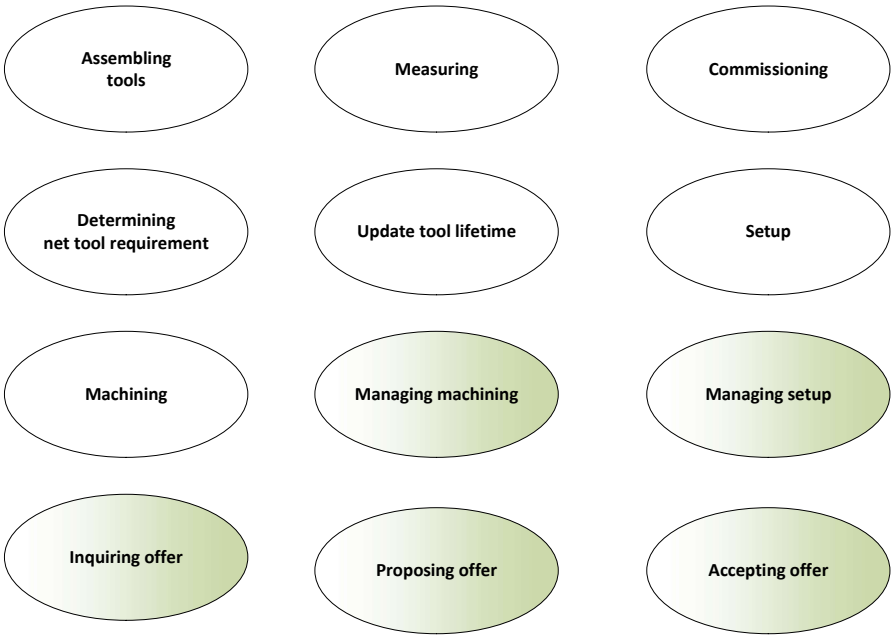

Figure 4.2: Services

4.6 Clustering Phase

Services clustering

Now that all services are defined, the old ones ensuring the core process and the new services addressing the KPI, the clustering phase can start. From this phase on the focus is mainly on the services and no longer on the functions and tasks required to provide them. During the clustering process the clustering rules provide guidance.

Cluster 1: First, the inquiring offer service and the accepting offer service are clustered together. This is an intuitive decision as it is comprehensible to assign the accepting offer service to the entity requesting an offer in the first place. This is also reasoned, as the entity requiring offers from several entities, holds the knowledge of all made offers.

Cluster 2: This cluster is based on the first clustering rule, suggesting the joining of resources. The assembling tools service, the commissioning service and the setup service have all joint resources, the human worker. While all three require also other resources for their fulfillment, the human work is their joint resource and suggests a fusion to one cluster.

Cluster 3: This cluster arises from clustering rule 3, combining familiar services to exploit possible synergies. This cluster includes the managing machining service and the managing setup service. Both services address the scheduling and the organization mechanism. Those mechanisms could be similar or even inherit from one another. Both services have the responsibility to ensure that specific restrictions are obeyed, to prevent machine idle times and to ensure a sufficient tool supply at the machine tool.

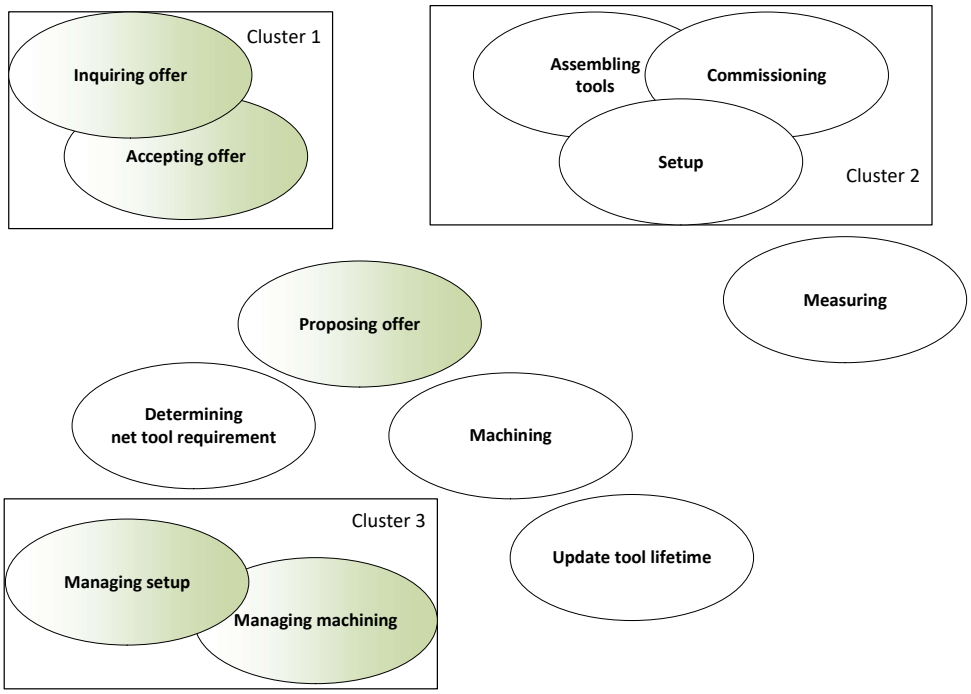

Figure 4.3: Service clusters at the state 1

Figure 4.3 shows the current clustering state. As the reader might have noticed, the already defined clusters are in the corner of the graphic, while the other services are positioned at distances according to their interactions or relations. The proposing offer service is between Cluster 1 and Cluster 3. This service interacts mainly with the services of Cluster 1, but it also shares joint resources with Cluster 3. It shares with cluster 3 not only the information it gets from the managing machining service, but it sells a position in the machining schedule organized by cluster 3. The machining service is located between the proposing offer service and Cluster 3. Its service is used by both parties, while facilitates the service of the update tool lifetime service right next to it. Although those services provide each other with something of value, their clustering depends not on this criterion, as it would result in one big cluster of all services.

Cluster 4: If the proposing offer service is combined with the machining service, the calculation of an offer and its execution would be in one hand. This has the benefit of controlling and ensuring execution of the offer. This might not seem to be required in the setting of the Pilotfabrik, but if several machine tools are involved on one shop floor this combination could benefit the validating the offer. Higher services arising from this clustering could be the validating the offer service and the monitoring the execution service. The monitoring the execution service could be a service not only monitoring the progress of the machining but also posting life feedback to the buyer. the validating the offer service could revise the made offer if something unforeseen occurs, like machine downtime. It can also adapt its offer if production order time is shorter than expected. Both higher services would speak for clustering those services together according to the cluster rule 2.
When defining the update tool lifetime service, it was argued that this service could provide more value to another party, than the machine tool, which is why the fusion with another service was postponed. Now that all services in this system are observed, it is noted that the combination of this service with any other service will neither support the familiarity rule nor the higher

service clustering rule. Therefore, this service is combined with the machining service in cluster 4. The update tool lifetime service could remain solitary, but it does not have enough substance to find its own cluster, nor has it any other cluster services as main consumer of its service. Figure 4.4 illustrates the current clustering state. The determining net tool requirement service

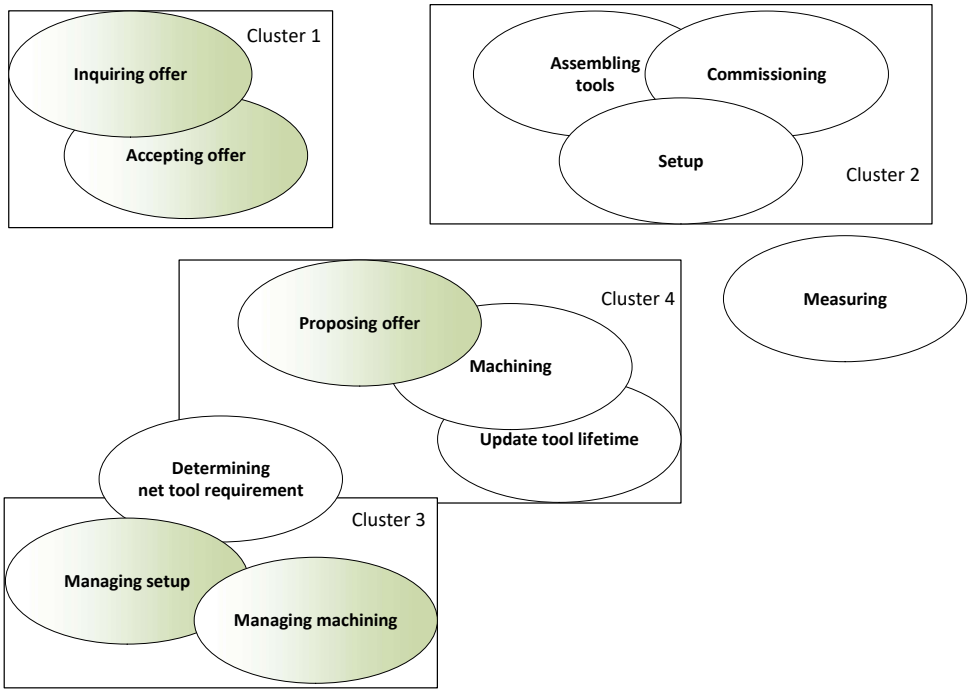

Figure 4.4: Service clusters at the state 2

is now located between two clusters. The determining the net tool requirement service is of value for the proposing offer service. This is reasoned, as the proposing offer service requires this valuable information to calculate the production cost for the specific machining schedule spot. On the other hand, the determining the net tool requirement service is also of high value to the managing setup service, as it offers the valuable information of tools are required. This value is turned by the managing setup service into managing the tool setup individualized to each machining schedule spot. Now the question is, which cluster the determining net tool requirement service belongs to. It does not have any familiarity with an existing service on this map. It shares resources with the machining service, as it acquires the machine tool control for the magazine load. However, it does also share resources with the managing setup service, as it requires a TMS interface to request the current tool stock. The second clustering rule, addressing the combination of services for a higher service, will resolve this. Currently cluster 4 is the machining cluster and cluster 3 the managing machining cluster (see figure 4.4). The determining net tool requirement service would fit both clusters but does not add up to a higher service if jointed with only one cluster. However, if both clusters are combined with the determining net tool requirement service, a higher service, such as the scheduling machining service, could be gained. This cluster combines all required services to propose, execute and manage the flexible machining schedule.

So cluster 3 and 4 are combined to one cluster (see 4.5). Since the measuring service depends on the human as resources, it shall be added to cluster 2. Since the required functions were defined previously the creative clustering will not jeopardize the functionality of the core process.

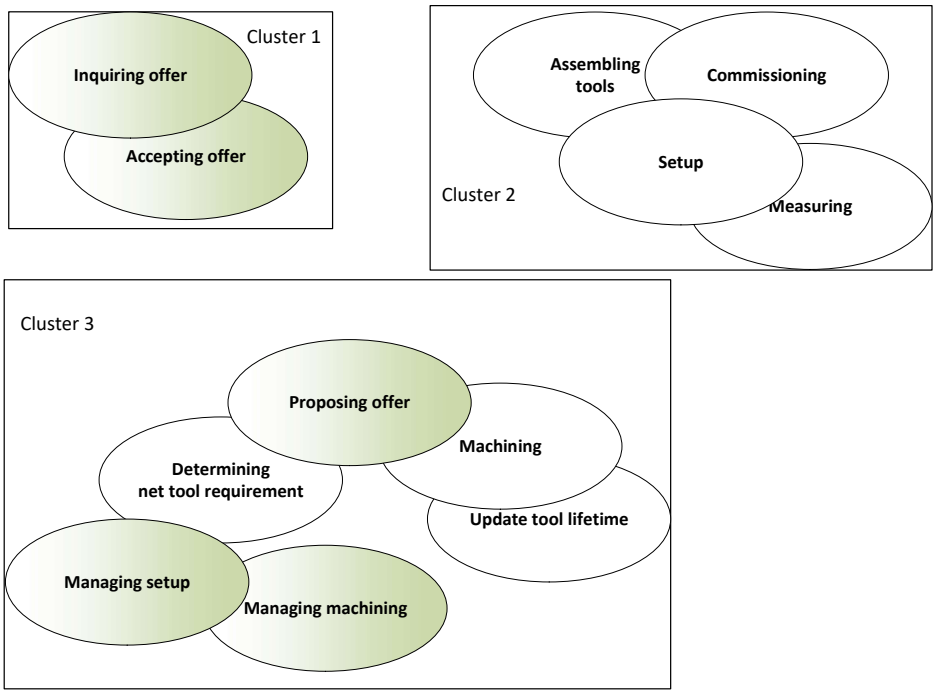

Figure 4.5: Service clusters at their final state

4.7 Transformation phase

Now that the services are clustered, the transformation those three clusters into agent types can begin. First, their role is defined and then their responsibilities. Then their multiplicity is being discussed and defined. Afterwards, their interactions are being resolved. Then the redefined process entities required to support those interactions are being presented. At the end of this section, each agent type and its structure is being described. And finally, their interaction is being described.

4.7.1 Role assigning

This step of the phase will assign roles and responsibilities to the previously presented clusters. Since the MAS resulting from this thesis of this work will only be realized in JADE and not yet directly in the Pilotfabrik setting, the human work and the database must be mapped into the final MAS design. Since the integrated tool cycle shall be managed by the MAS design both process entities must be included as additional agent types. Cluster 2 would for example, if the human worker executes those tasks in the factory manage all handling and assembling tasks and trigger the required steps. During this thesis the work of the worker is simulated by splitting the cluster into two agent types. One covering the core service, which is the tool supply and one simulating the humans' tasks and the measurements of the presetting device.

No Cluster: The VisualAgent

Similar to the DataAgent, the visual Agent is a compromise between process-necessary entities and the wish to map the tool cycle completely in a MAS simulation. It covers those tasks in the simulation, that a real human being would fulfill in a real-world implementation. If the MAS would be implemented into the Pilotfabrik, this agent would not be necessary.

No Cluster: DataAgent

Similar to the VisualAgent, the DataAgent is implemented into this MAS not due to its services or its role. It has been implemented to enable simulating the tool cycle process controlled by the MAS in JADE. Due to JADEs' XML interface it would be possible to establish a connection to the TMS database of the Pilotfabrik. But since the generated data would not be based on real-world data (measurements, inventory number, magazine load), the additional effort is not reasonable. Therefore, the DataAgent is established as the simplest representative of the TMS connection in JADE.

Cluster 1: The OrderAgent

An OrderAgent represents a released machining order in the MAS. It offers the system the following services or actions:

- Triggering the process start
- ERP-related parameters (maximum order execution time, maximum production cost, time factor)
- NC-ProgrammID, required for determining the gross tool demand
- Negotiating
- Acting in its best interest
- Accepting or denying an offer
- Reporting to the higher MES or ERP layer
- Indirect competition between OrderAgents by their multiplicity

Cluster 2: The SupplyAgent

The SupplyAgent is designed to fulfill all tasks required in the tool supply. It represents the state of the storage, informs about the location of the components, and manages the presetting and measuring. Although it supports the human worker by triggering the VisualAgent, it does not assemble the tools, nor does it deliver them to the machine tool. Those tasks are covered by the VisualAgent and the human worker. The SupplyAgent monitors all cutting tools, which are currently not in the machine tool magazine. It also covers the following services:

- Tool management in storage
- Creation and management of Setup orders
- Measuring
- Calculation of the $Time_{UntilSetup}$

The SupplyAgent is designed together with the VisualAgent, to ensure that the whole tool supply is covered. The SupplyAgent maps the process steps required for the simulation, such as the tool assembly or, the initiation of a tool instance, the tool measurement at the presetting device, and the tool data management. It is also a valuable partner for the MachineAgent since it provides the required tools on time at the machine tool by processing the ToolSupplyOrder.

Cluster 3: The MachineAgent

The MachineAgent represents the machine tool and all its states including its classical tasks, such as machining, and updating the tool lifetime. Reasons for choosing a MachineAgent are:

- Multiplicity: one or more could be generated
- Competition based on pricing, MachineAgents could compete directly with each other by adapting the price
- Competition based on current magazine load
- Knowledge of magazine load at any possible time
- Authorized to negotiate, since it provides the service
- Machining order sequencing, with the aim of minimal production cost achieved by minimal setup rates

The MachineAgent also covers following services:

- Offering the OrderAgents coverage-based production prices
- Negotiating with one or more OrderAgents
- Planning the future magazine load
- Managing its own tool disposition
- Determining the net tool requirement
- Orchestrating the order sequence (machine utilization) towards a minimum setup time

4.7.2 Agent type multiplicity

VisualAgent multiplicity

The multiplicity of the VisualAgent depends on how many human workers and working stations are available on the shop floor. Since a shop floor could have more than one worker and working station, there could be more than one VisualAgent representing them. According to the functionality, the system would benefit, if more than one VisualAgent would exist. In fact, the VisualAgents and their resources could compete against each other for the bid. Generally, it can be said, that the multiplicity of the VisualAgent could be between 1 and n, but since the Pilotfabrik has only one worker and one working station its multiplicity is set between zero and one for the simulation.

DataAgent multiplicity

The DataAgent is limited by its multiplicity to one. One DataAgent is enough to represent all relevant services in this MAS. The other reason is, that JADE offers the existence of a Container Agent (see 2) in its structure. A Container Agent is required in JADE to initiate a container (MAS environment) in which the agent society runs. While each MAS could have a nearly infinite multiplicity of different agent types, it can only have one Container Agent. Another reason for the multiplicity of one is that a second DataAgent has no additional benefit. Unless

the databases are not separated, either by location or organizational reasons, it has no additional benefit (except for back-up) to have two databases involved. If the databases were separated either by location or by organizational reasons, a higher multiplicity and therefore competition would be beneficial. In fact, JADE offers, similar to this idea, the functionality of Container Agents being connected and communicating, which could be elaborated as two shop floors, each managed by its own DataAgent competing with each other. Competing DataAgents would only be competing based on the characteristics of their shop floors, which they can not influence actively. This is an additional reason why the competition of DataAgents does not contribute to the system. Since the DataAgent is only established for the simulation and it serves as Container Agent its multiplicity is set to 1 (see figure 4.6).

OrderAgent multiplicity

Representing the machining orders as OrderAgents allows to raise their multiplicity and therefore their competition. Since all OrderAgents are trying to optimize towards their best possible outcome, they are (currently) indirectly competing against each other. An OrderAgent can either accept or deny an offer and competes indirectly by its net tool requirement and its existence. The mechanisms of indirectly competing are the missing knowledge of the offers made to the competitor and the inability to make a counteroffer. Its multiplicity is set between zero and infinity.

MachineAgent multiplicity

In the Pilotfarbik more than one machine tool could theoretically be deployed, which backs the extensibility of the system. However, given the fact that a CNC-program is a numerically coded routing step fitting a specific machine tool type, only machine tools of the same type could compete. Although the competition is limited to one machine tool type, it still could benefit the overall productivity. The MachineAgents could adapt their prices to get the bid, as a form of direct competition. The MachineAgents could also be in indirect competition, due to their availability or current Magazine load. The multiplicity of the MachineAgent could be between 1 and n, but since the Pilotfabrik has only one machine tool its multiplicity is set between zero and one for the simulation (see figure 4.6).

SupplyAgent multiplicity

The multiplicity of the SupplyAgent depends on the number of presetting devices and storages available. If more than of those resources are available at the shop floor their competition will bring a benefit to the system. Generally, the multiplicity of the SupplyAgent could be between 1 and n, but since the Pilotfabrik has only storage and one TMS its multiplicity is set between zero and one for the simulation.

4.7.3 Redefined Process Entities

Abstracting services from the initial tool cycle process requires the adaption of process entities, inputs and outputs. Those adapted process entities evolved to cover either new tasks and roles or to provide a different data set. Some services, which did not derive directly from process activities, require the introduction of completely new process entities, such as the dynamic setting sheet and the machining queue. This section presents the redefined process entities, artefacts and those which had to be developed.

Machining Orders

The machining order presented here contains the NC-programID. Beside the NC-programID it contains a unique machining order ID, the workpieces quantity, the maximum production cost, the maximum order execution time and the time factor f. The last three parameters are developed so that the OrderAgent can decide on their basis to accept or deny an offer. The maximum production cost and maximum order execution time are inspired by the two major KPI. The OrderAgent representing a machining order shall try to find the position in the machining schedule with the lowest production cost (tooling cost) without exceeding the order execution time. All three parameters outline an enterprise strategy, even if just vaguely. The strategy is reflected by the time factor value, which describes the possible leeway to exceed the order execution time. If the time factor is one, the maximum order execution time shall not be exceeded. If it is higher than one, a longer order execution time is traded for a smaller production price.

NC-program

An NC-program is a work-plan for a specific machine tool type programmed in G-code. It is an executable numerical control program designed for a specific workpiece on a specific machine tool type. The here defined NC-program built the required connection between a machining order and the required setting sheet. The data of the machining order includes the NC-programID, which refers to a specific setting sheet. For the MAS the tool use time required to process one workpiece is stored for each tool, so that the overall machining time of a machining order can be calculated, as well as its required tool use time. The tool use time for one workpiece and per tool are set to ranges taken from real-world practice. The NC-programs is stored locally on the machine tool and are compiled on a CAM program with a suitable post-processor for the machine tool control format. The NC-program is designed and converted in NX CAM.

Setting Sheet

A setting sheet has a data structure, such as displayed before in table 4.4. The literature and the industrial practice have no standardized setting sheet structure, its structure depends on customized choices. Only three kinds of information are commonly used in setting sheets: the NC-programID, the machine tool and the tooling list. Since the net tool requirement shall be determined, the setting sheet requires additional data. This data is divided into three groups, one is the machining order related data, the second is the machining related data and the third is the extended tooling list required to fulfill the NC-program. The setting sheet data provided by Zoller TMS Gold is used and extended by the tool use time per workpiece for each cutting tool type. This minor adaptation makes it possible to calculate the order related required tool use time for each tool, and as a further consequence the gross and the net tool requirement. To calculate the unit processing time T_{UP} of an order the production time per workpiece T_{Pwp}, the nonproductive time per workpiece T_{NPwp} must be multiplied with the order specific workpiece quantity and be added up with the setup time $T_{SetupTime}$. The unit processing time T_{UP} of an order is required to determine the total order execution time. The production time is the sum of all tool use times in the extended tooling list. Like the nonproductive time it is a per workpiece time value. The nonproductive time describes the sum of all nonproductive tasks, that the machine tool fulfills during processing an NC-Program, such as tool changes and positioning. In practice it is usually measured or estimated. The values presented here are estimated as a per workpiece values. In the industrial practice, those values can be simulated in a CAM software, but they still lack accuracy, with a deviation of at least 15%. This deviation is caused

Setting sheet data						
Machining order related data		Machining related data		Extended tooling list		
Adapter	HSK 63	**Production time per workpiece[min]** T_{Pwp}	3,95	Tool type	Tool descr.	Tool use time [min]
Setting sheet ID	SettingSheet12072018	**None prod. time per workpiece [min]** T_{NPwp}	0,79	**6585290**	DrillD13	0,40
Product	Prod.A	**Setup time [min]** $T_{SetupTime}$	3,00	**6585298**	ShellMillD66	1,96
Machine tool type	MaxxMill750	**Unit processing time** T_{UP}	10,81	**6585302**	MillingCutterD9,25	0,85
Machine ToolID	M007		-	**6585295**	CyclindricalShaft	0,74

Table 4.4: Example of a setting sheet

by missing data about the real magazine load, the precise tool position in the magazine, and the tool changing strategy the machine tool utilizes for the magazine. Since this work is build on a process simulation, those set values are assumed to be sufficient. The setup time is also an estimated value, which describes the estimated time for a worker to set up the machine tool. It represents a onetime value and does therefore not scale with the number of workpieces. This can be reasoned with the small variance of the setup duration wherever 10 workpieces or 20 workpieces are handled. The unit processing time T_{UP} sums up the production time and the required setup time. It is required to calculate the order related machining costs since it sums up the time a machining order is occupying the machine tool. The extended tooling list offers the tool type specific tool use time. It is required to calculate the gross tool requirement per tool type for each machining order, which is here a minute value. In the industrial practice, the gross tool requirement is a generously estimated number of cutting tools, if determined at all (see tooling strategies in section 2.1.2). The setting sheet (as displayed in table 4.4) can be seen as a per workpiece template for calculating the order specific machining values.

Dynamic Setting Sheet

The dynamic setting sheet is developed to display the net tool requirement for each machining queue position. The dynamic setting sheet serves as an artefact that lists the minimal quantity of tools to be assembled for a position in the machining schedule. This minimal quantity of tool instances to be assembled is called the total net tool requirement $TReq_{NetTotal}$ [piece]. The template to determine the tool supply requirement for each position in the machining waiting line. The dynamic setting sheet is used to automatically generate a tool supply order of the total net tool requirement if the machining position is granted.

Machining Queue

A machining queue represents the machining order sequence at a given moment in time. It reflects in which order the MAS processes the machining orders. The machining queue is a flexible machine utilization, rescheduling the machining order if thereby smaller tooling costs can be realized. The machining queue has a frozen zone, which ensures that the first and second position can not be changed. All other positions outside the frozen zone can be modified until they reach the frozen zone. In fact, infinite machining orders could be sorted in a sequence. Some production parameters vary with machining queue position, such as the order execution time, the net tool requirement and the production cost. Each machining queue position has its own magazine load, which depends on the remaining tool lifetime of the currently mounted tools and the preceding machining orders. Therefore, the production cost and the net tool requirement vary for each machining queue position. Determining the net tool requirement for each machining queue position and its production cost is done by the MachineAgent. It manages the machining queue.

Frozen zone

This new process entity will force the system to set a sequence. Since the tool cycle starts by an incoming OrderAgent representing a machining order, the MAS will start to sequence the machining orders according to a local optimum. The MAS might find it comprehensible, that waiting for more machining orders to arrive would prevent a disadvantageous sequence. If the multiagent system has the freedom to decide how long it could wait to collect machining orders, it might never start. The frozen zone prevents a long order execution time, since the first and the second machining order will always be machined next. The first and second position in the machining queue are called "frozen zone". This mechanism also reflects the reality of the industry, where machine idle times are expensive and must therefore be prevented. Assembling cutting tools, measuring, and delivering them to the machine tool takes time. Without a frozen zone the sequence could be changed again and again, without any machining order being processed. In such a case, various redundant tools would be assembled and commissioned to the machine tool without being used. Such a development would further lead to a high demand for the tool components, which once assembled, would not be under the oversight of the tool management system.

Predicting the remaining tool lifetime

Several possibilities to track and manage the remaining tool lifetime exist, either by storing it on an RFID tag or by updating the value in the database. During the simulation of the tool cycle in JADE, the remaining lifetime before and after the machining orders is determined by prediction. Even in a real-world implementation, a previous prediction of the tool lifetime is a practical way to schedule cutting tools, even though they were not subject to oversight at all times. However, the industrial practice abstains from this solution since it demands a high organizational effort. The remaining tool lifetime is a product of the NC-program related tool lifetime minus the tool use time per workpiece. The tool use time is information stored in the setting sheet and describes the time in minutes, which the cutting tool is cutting into the workpiece according to the NC-program.

Estimating the NC-program related tool lifetime

The remaining tool lifetime can only be estimated based on the tool load, the feed and the cutting depth. The actual tool lifetime and its consumption depends on several factors besides the ones mentioned above, such as lubrication and span transportation. Both criteria depend on the machining circumstances. In practice, the tool lifetime is determined by machining under real circumstances and monitoring the tool wear. A large amount of scientific effort focuses on investigating how to estimate the tool lifetime precisely. Besides complex mathematical and computer-aided models, there is a vibrant discussion about which machining factors affect the tool lifetime more than others. Since finding an answer to those questions would provide scientific sustenance for several doctoral thesis, this thesis abstains from further addressing this matter. This thesis rather assumes that the NC-program related tool lifetime can be correctly estimated. In fact, the NC-program related tool lifetime in this approach is estimated and has not been verified by a practical test. While setting the values of the NC-program related lifetime, the focus was placed on a fair presentation of realistic values. Not only does the author assume that the NC-program related lifetime is estimated correctly, but also that it stands in linear relation to the catalog lifetime. The catalog tool lifetime depends on a pre-defined feed, tool depth, and tool load and is provided by the tool manufacturer. If the NC-program related lifetime stands in linear relation the catalog lifetime, a translation from one specific tool load to

the other must be possible. Since the aim of this work is to minimize the tooling costs, using the same cutting tools with different NC-programs is essential. In the industrial reality a significant amount of the tool lifetime is left unused to prevent tool damage or breakage, nor is it common to switch cutting tools between different NC-programs. The NC-program related tool lifetime is an estimated tool lifetime based on the tool load, feed and the cutting depth. Therefore, it most commonly constitutes a smaller part of the catalog tool lifetime, if the tool load is higher than assumed in the catalog lifetime. But there are exceptions, for example if the NC-program related cutting tool depth, feed and load falls below the values referenced in the catalog lifetime. In such cases the estimated NC-program related tool lifetime would be higher than the catalog lifetime, since the wear and tear would be less than the ones assumed by the manufacturer.

Implementing a Lifetimefactor L_f for converting the NC-program specific tool use time to allow the use of tools in different NC-programs

To minimize the tooling cost, it is important to use the same cutting tool instance with different NC-programs. This practice is not common, since the estimated or tested tool lifetime relates to a specific NC-program. However, the feasibility of this idea is exactly the subject of this developed process entity. Based on the assumption, that the NC-program specific tool lifetime, and the catalog tool lifetime do relate in a linear manner, a translation from one NC-program specific tool lifetime to another must be possible. Therefore, the tool wear intensity can be scaled, by dividing the estimated tool lifetime, which is NC-program specific, by the catalog tool lifetime provided by the manufacturer. The result of this division is the lifetime factor, which is the subject of this developed process entity. The higher the tool wear, the smaller the tool lifetime factor. Creating this factor allows the use of a cutting tool instance in different NC-programs, without completely losing track of wherever the tool lifetime must be reduced or increased. Each tool has its NC-program specific lifetime stored in its setting sheet, as each NC-program displays the lifetime consumption based on the lifetime a new tool would last in this specific NC-program. Knowing this consumption makes the switch from one NC-Program to another possible. This can be accomplished by normalizing the NC-program related value according to the catalogue tool lifetime, using the Lifetimefactor L_f.

This Lifetimefactor sets the catalog lifetime in relation with the NC-program based lifetime.

$$L_f = \frac{CatalogLifetime}{NCLifetime} \tag{4.1}$$

The Lifetimefactor is determined and multiplied with the tool use time of the NC-program to normalize its value based on the catalog tool lifetime. By doing so the remaining lifetime is managed on the base of the catalog tool lifetime and eliminates the necessity of tracking NC-program changes. A face mill for example with catalog lifetime of 200 minutes shall be used in NC-program A for 20 minutes, in which it would last 150 minutes with a newly assembled. The newly assembled face mill shall afterwards be used in a different NC-program B for 30 minutes, in which a new face mill would last 170 minutes. The Lifetimefactor L_f for translating the tool use time from NC-program A into a catalog lifetime normalized value would be:

$$L_f = \frac{200minutes}{150minutes} = 1,33 \tag{4.2}$$

The normalized tool use time would therefore be 26,67 minutes, instead of the original 20 minutes. The face mill has therefore 173,33 minutes left of its normalized remaining lifetime. Now the face mill shall be used in NC-program B for 30 minutes.

$$L_f = \frac{200minutes}{170minutes} = 1,18 \tag{4.3}$$

The normalized tool use time of the NC-program B would be 35,4 minutes instead of 30 minutes. After the machining with NC-program B, the face mill still has 137,93 minutes lifetime remaining. Using the catalog lifetime and normalizing the NC-specific tool use time is a suitable method to track and convert the remaining lifetime, while using the tools in different NC-programs. This approach does not require storing the usage history for each tool inventory number with a timestamp, as it were necessary if the remaining lifetime would be converted from each NC-program to the next one.

Tooling Cost

Costs that are caused either directly or indirectly by cutting tools are nothing new. As described in section 2.1, classical tooling costs are purchase, storage, handling and managing cost. Most producing companies do not measure or calculate the cost. This has several reasons. One is that each tool must be recorded, which consumes time and effort. In addition, most standard cutting tools do not cost enough for the company to trace their consumption. In this work, the tooling cost is emphasized. It is therefore set as key performance indicators. The strategy to minimize the tooling cost is to schedule the machining order in a manner that requires the smallest net tool requirement, and therefore the smallest amount of newly assembled tools to be setup. An already assembled tool is as cheaper than a newly assembled cutting tool since the tool supply causes hidden costs. In this work a cutting tool already at the machine tool is priced cheaper than an already assembled tool from the storage or a newly assembled tool.

Tool Supply Order

A Tool supply order is nothing new. As the state-of-the-art chapter explains, there are different ways to address the tool supply, as for example with a Kan Ban system or by storing a large number of tools directly at the machine tool. In this work on the other hand, a tool supply order is used to set up the minimal number of required tools and to commission them to the machine tool just in time. This minimal number of required tools is the predetermined total net tool requirement displayed in the dynamic setting sheet. To provide the tool supply process with enough time, a frozen zone is used. The second machining order (frozen zone) can not be changed since the tool supply order for this machining order has already been released and is being processed. To further prohibit nonproductive time, due to a delayed tool supply, each tool supply order has a predicted time frame from its ordering to the finished setup. The MAS should therefore be capable to orchestrate the machining order, so that the nonproductive time can be held at a minimum.

Time until setup $Time_{UntilSetup}$

Creating this new value is necessary to prevent a machine idle time due to missing tools. Before offering a possible position in the machining queue, the $Time_{UntilSetup}$ value must be compared with the time remaining until the time slot scheduled for the order. Thereby the $Time_{UntilSetup}$ must be shorter than the order execution time of the preceding orders. This restriction ensures, that the required cutting tools will be at the machine tool before the scheduled setup starts. It is a simple assumption but demands high logistical and managing efforts across different shop floor areas. Making this constraint a priority, effects the process, as only machining orders with an on-time tool supply can be considered for a machining queue position. It requires a higher consultation between the agents to meet the delivery deadline, compared to designs presupposing a very restrictive buffer time.

Human Labor

Although a lot of tasks in manufacturing have been automated, there are still many tasks that depend on human dexterity, creativity and skill. Some tasks are also automated, simply because they are not profitable and easy to outsource to robots. Assembling cutting tools is a complex task to be automated, but it is easy to handle for a human with experience. Since the system is running only on JADE and has no linkage to the real-world process yet, the human labor must be mapped in the process. The human labor, including the assembly, the measurement and the commissioning of the required cutting tools, is a crucial part of the tool cycle. Therefore, an object must be designed, which maps the services depending on the human labor. This subject is the VisualAgent which simulates the tasks of the worker and their fulfillment. Those tasks are picking tool components, assembling, measuring, commissioning tools to the machine tool and doing the setup.

4.7.4 Agent interaction

The MAS is triggered at the moment a machining order enters the system, represented by an OrderAgent. Its advent can be triggered by an arriving XML file or a boot-command from the outside of JADE directing to an XML file with the machining order data. The instantiated OrderAgent requests several offers from the MachineAgent. Each offer reflects a different machining queue position. The decision of the OrderAgent depends on the ERP-given parameters, such as the maximum (production) cost [Euro], maximum order execution time [minute] and the time factor f [1]. Based on those given parameters, the OrderAgent decides if it accepts or denies an offer. An OrderAgent contacts directly the MachineAgent and requests an offer.

The cooperation between the OrderAgent and the MachineAgent is conceptualized in a manner, so that each one is focused on its win. The interaction type between the OrderAgent and the MachineAgent is a negotiation (see figure 4.7). The MachineAgent seeks minimal tooling costs and smaller tool setup, while the OrderAgent seeks to optimize either its order execution time or its production cost. A lower production cost could for example be traded for a longer order execution time and vice versa. The time factor f gives the OrderAgent its required decision frame. A higher time factor means more leeway to optimize its production cost at the expense of the total order execution time. A lower time factor indicates to the OrderAgent that the order execution time is set as the most valuable parameter and makes it try to keep its order execution time target at the expense of the production cost.

Requesting an offer requires transmitting two types of data, the NC-programID and the workpiece quantity. The NC-programID leads the MachineAgent with the help of the DataAgent to the setting sheet. The setting sheet summarizes all machining relevant information, based on the NC-program. The setting sheet contains machining related data and extended tooling list. With this data the MachineAgent can calculate the $TReq_NetMachine$ for each available machining queue position.

Before the MachineAgent offers the OrderAgent the resulting production cost and the order execution time, it checks that time until setup $Time_{UntilSetup}$ is smaller than the total order execution time T_{TOE} of the preceding orders. Since the MachineAgent predicts the net tool requirement and the total order execution time, it requires at this stage the $Time_{UntilSetup}$ value from the SupplyAgent. Before checking that the time until setup value is smaller than the total order execution time T_{TOE} of the preceding orders, the MachineAgent calculates the net tool requirement for each possible machining queue position as well as, the unit processing time. With that information it can derive the production costs and the total order execution time to

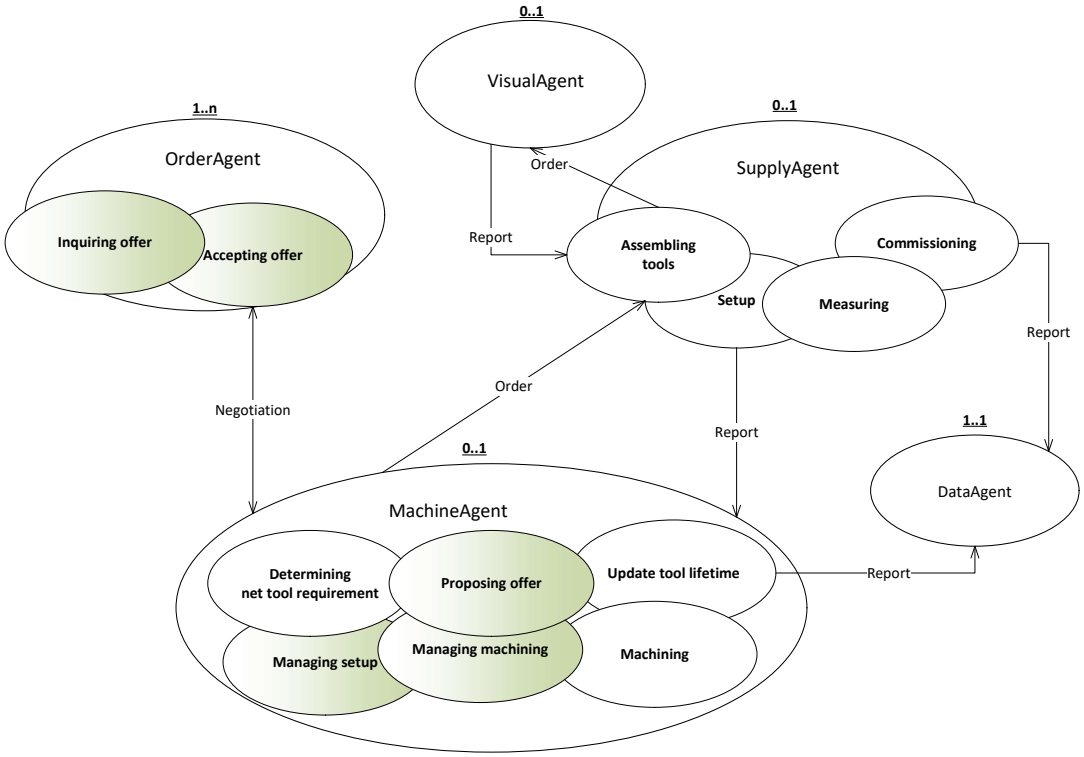

Figure 4.6: Multiagent Interaction Diagram

offer the OrderAgent both values. Now the OrderAgent needs to decide whether the offers made are suitable or not.

At this stage of work, no learning algorithm or competitive mechanism are implemented, so that the OrderAgent does not consider the possible decisions of other OrderAgents. The OrderAgent bases its decision on the received values for the production cost and the order execution time. It either accepts or denies an offer. It can currently not start bargaining, for example offering to wait a bit longer for a better price. If it accepts or denies an offer, it is currently based on the ERP-given maximum production cost and order execution time. If the offer does not exceed the cost or the time, the OrderAgent will certainly accept it. In case the order execution time will be exceeded, the OrderAgent has an ERP-given leeway, the time factor f. With its help it can decide if the excess is still in an acceptable range. If the maximum production cost is exceeded, the OrderAgent cannot accept the offer. It denies and returns an error message to the ERP layer, explaining that the ERP-given production cost is currently not realizable and then terminates.

The new offer is made by the MachineAgent if a new sequence would save more tooling cost. If for example an OrderAgent is scheduled on the machining queue position three, and another OrderAgent arrives into the system, which would require less tools on that position to be setup, then the MachineAgent makes a new offer to the OrderAgent currently in third position. This offer must succeed the previous made offer in one way or the other. Since the MachineAgent cannot offer the OrderAgent a shorter order execution time, as it wants to delay its machining by squeezing another order in, it has to offer a better price. The MachineAgent calculates how much

money it would save by the changed sequence and offers the previous OrderAgent a production price reduction, by passing half of the cost reduction on. So, the OrderAgent receives a new offer for the fourth position, which includes a production price reduced by 50% of the realized saving. Now the OrderAgent can decide if it accepts or denies the second offer. It evaluates if the order execution time is still in the range with the respect of the time factor f leeway or not. If the order execution time is acceptable, it certainly will accept the offer. If not, it denies it. In case the OrderAgent accepts it, the MachineAgent will restructure the sequence and put the new OrderAgent in third position. This simple negotiation between the MachineAgent and the OrderAgents realizes a local optimum and reduces tooling costs. In case the OrderAgent denies, for whatever reason, the MachineAgent must accept its decision and the MAS has failed in realizing a possible local optimum. If an OrderAgent accepts a machining queue position, it could change it again by accepting a newly made offer from the MachineAgent. Until the OrderAgent rises to the second position in the machining queue might be rescheduled.

If an OrderAgent reaches the second position in the machining queue, the MachineAgent orders the SupplyAgent to schedule the supply order (see figure 4.7). The interaction type is set as an order due to the importance of its fulfillment. The supply order send to the SupplyAgent includes the already calculated the net tool requirement after considering the tools available at the stock $TReq_{NetSt}$ and then the total net tool requirement, which reflects the list of tools to be newly assembled. To calculate the net tool requirement in stock, the MachineAgent orders the SupplyAgent to give a report of all fitting tool instances and their remaining lifetime available in the storage. The SupplyAgent is, from now on, in charge of providing the MachineAgent with the right tools and the estimated $Time_{UntilSetup(k)}$ at the machine tool. It generates a dynamic setting sheet, which includes all newly assembled cutting tools and their bill of material (picking list). The SupplyAgent requests of the DataAgent to store the dynamic setting sheet and to generate a picking list. Then the SupplyAgent hands the picking list over to the VisualAgent, which is the interface to a human worker, and orders it to collect and assemble the tools. The VisualAgent aids the MAS, as it maps the process steps fulfilled by a human worker, and it aids the worker by being a GUI guiding her or him through the work steps. Based on the picking list, it visualizes the components for the worker to be assembled. It also generates an inventory number with a TMS related number generator for each newly assembled tool. Now that the components are assembled to cutting tools and an inventory number is assigned to them, all tools (newly assembled and pre-existing ones) are handed over to the measuring unit. The measuring unit is an automatic presetting device, which has an interface to the SupplyAgent. The tool corrective geometries are measured and stored in the database. Then the SupplyAgent orders the VisualAgent to transport the tools to the machine tool. Arriving at the machine tool the VisualAgent requests a take-out list from the MachineAgent. This list displays all tools to be dismounted from the magazine, with their inventory numbers and their remaining lifetime. Since the MachineAgent has now no magazine load nor unload strategy, it removes the tools beginning at the magazine position one, if they are not required in the upcoming machining order. It only considers the next two machining orders. Receiving the take-out list, the VisualAgent displays it on its GUI and the worker can work through it and then sign it as completed, which allows the MachineAgent to start its machining process. Before the MachineAgent starts machining, it requests the corrective geometrical data of the newly assembled tools from the DataAgent. While the MachineAgent is machining, the VisualAgent and the human worker deliver the dismounted tools back to the storage and inform the SupplyAgent and the DataAgent of their remaining tool lifetime. After the MachineAgent finishes machining it informs the OrderAgent, which creates a logfile, including all received offers and the one it chosen. It then terminates right after transmitting the log file to the ERP.

4.7.5 Optional: Sequence Diagram

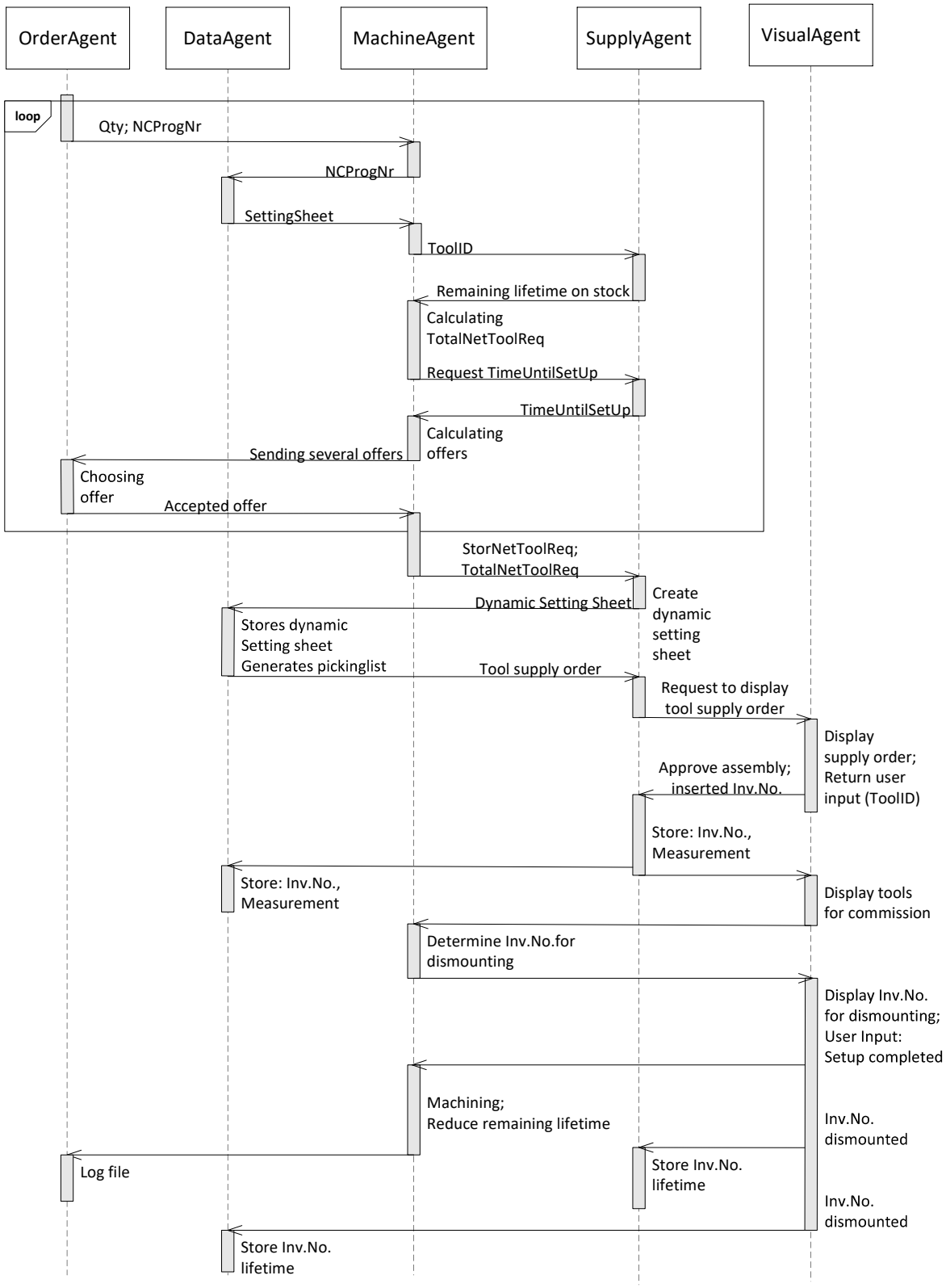

Figure 4.7: Sequence diagram of the agent cooperation

4.8 The Final MAS Design

The following section describes the final agent types of this MAS with focus on their internal functions.

4.8.1 The DataAgent and its functionality

This agent type owes its existence to the fact that the tool cycle is simulated in JADE. The DataAgent can access systems outside JADE by accessing the TMS database. When the DataAgent terminates, the entire platform in JADE is also terminated. DataAgent reads writes and accesses the TMS database. The DataAgent controls the following services:

- Managing and storing the tool data
- Non-redundant and latest tool data
- Representation of the TMS database
- Representing the JADE Container Agent
- Displaying the database in the multiagent system.

4.8.2 The VisualAgent and its functionality

The VisualAgent mainly offers the service provided by human workers, such as initializing a new tool with a tool inventory number, assembling and commissioning tools, as well as performing the machine tool setup. It represents the interface of the MAS tool cycle process to the real-world as it utilizes the interface to the worker. The above-mentioned steps require the performance of the worker in the process. Depending on, which agent triggers the VisualAgent, a different GUI will be generated, and displayed. These GUIs are displayed on the console and filled out or acknowledged by the worker. The VisualAgent also displays its information for the worker, which are:

- Displaying the supply order
- Displaying the tools for commission
- Displaying the tools to dismount from the machine tool
- Returning the new inventory number
- Confirming the commission of tools
- Confirming the setup completion
- Reporting the tool inventory number of dismounted tools

Those actions are described in detail in section 4.7.4.

OrderAgentID	Quantity	NC-programmID	Maximum production cost	Maximum order execution time	f
MachiningOrderID [String]	n [pieces]	NC-progID [String]	costs [Euro]	time [minutes]	$1 \leqslant f \leqslant 2$

Table 4.5: Data of an OrderAgent

4.8.3 The OrderAgent and its functionality

The OrderAgent represents the highest possible instance in the MAS, since it contains ERP-given parameters. Depending on the given parameters, the OrderAgent evaluates the ERP-given strategy. As table 4.5 exemplifies, an OrderAgent always has a unique identifier, a quantity to be produced, a referring NC-programID and its ERP-given parameters. The OrderAgent has three ERP-given parameters, the maximum production cost in the following referred to as maximum costs, the maximum order execution time and the time factor. Depending on the set parameter, the ERP can transform its pursuit strategy. If the maximum cost is set to a low value, and the order execution time to a high, the OrderAgent has more freedom to seek the best possible price, at the cost of a longer order execution time. If the cost is set to a high value and the order execution time is set to a low value, the OrderAgent needs to accept an expensive bid in order to meet the imposed order execution time. The time factor f ranges from one to two and reflects the deviation the OrderAgent allows from the maximum order execution time. If this value did not exist, the OrderAgent would have to reject offers, even if the target value would be missed by only a fraction.

$$1 \leqslant f \leqslant 2 \tag{4.4}$$

Depending on the given parameter the OrderAgent is capable of negotiating, accepting and denying offered positions in the machining queue.

4.8.4 The MachineAgent and its functionality

Although, the number of machine tools is theoretically unlimited only one MachineAgent is deployed in the Pilotfabrik. The MachineAgent has a machining queue, which displays the current machine tool utilization (see figure 4.8). Every position of the machining queue beyond the frozen zone (position 1 and 2), is changeable. The MachineAgent sorts the OrderAgents in the machining queue with the intention of lowering the tooling costs. OrderAgents utilizing the same tools or requiring a smaller tool setup will be machined back to back.

Figure 4.8 displays a schematical machining queue of the MachineAgent. The blue area depicts the magazine with its current tools, displayed as their ToolID number. The right orange bar shows only a part of the machining queue since the number of positions could theoretically be infinite. The first two positions are marked with a grey background, illustrating the frozen zone. Since the machining sequence is addressed as an online problem it can change continuously as new OrderAgents arrive at the systems. Figure 4.8 illustrates this circumstance by multiple machining queues at different moments (t). Each machining queue at a different moment in time describes either a change in the machining queue caused either by a machining order or a rescheduling due to a new arriving order. The magazine load in the figure shows the tool types available at the moment t1 in which the order OA is being machined. The magazine load changes depending on the machined order on position one. At the moment t1 every position in the machining queue is changeable, except for the OA and OB.

For adapting the sequence and changing an already assigned and accepted machining queue position, the MachineAgent must offer the OrderAgent, who is currently holding the desired position, a revised offer and wait for its acceptance or denial. A revised offer includes either a shorter order execution time or a lower production price. The machining queue can only

be adapted, when both OrderAgents, the one holding the desired position and the other one intended for this position, accept.

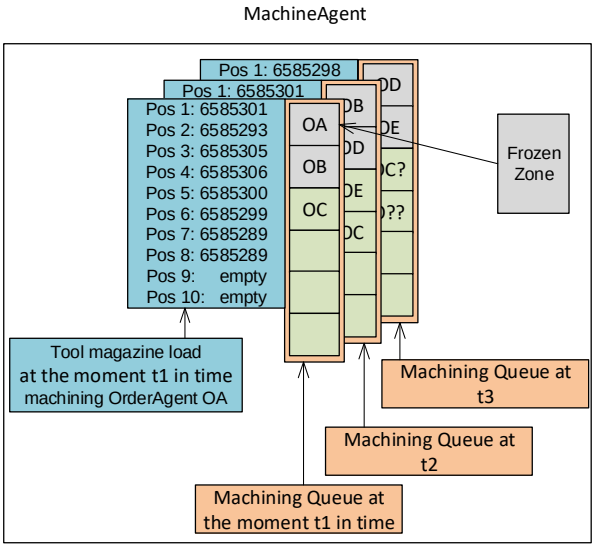

Figure 4.8: Scheme of the machining queue

The MachineAgent does not differentiate when pricing, whether a cutting tool is changed because it is worn out, or because it is not required by the next OrderAgent. The MachineAgent does not differentiate between the reasons why a cutting tool is changed and prices each change equally. To implement the sequencing aim in a more abstract and machine understandable way, the MachineAgent makes each newly arriving OrderAgent X an individual offer for each possible machining queue position. The thereby offered production price $C_{P(X)}$ [Euro], reflects the net tool requirement for the proposed machining queue position. The pricing is designed so that only the tooling costs $C_{T(X)}$ [Euro] are variable and depend on the machining queue position, while the machining cost $C_{M(X)}$ [Euro] and quantity dependent handling costs $C_{H(X)}$ [Euro] do not vary. The production cost $C_{P(X)}$ that the MachineAgent offers to the OrderAgent, identified as object X, adds up the three cost types for the offered machining queue position.

$$C_{P(X)} = C_{M(X)} + C_{H(X)} + C_{T(X)} \tag{4.5}$$

The $C_{M(X)}$ is the machining cost resulting from multiplying the unit processing time $T_{UP(X)}$ [minute] for object X with the machining cost rate C_{MR} [Euro/hour].

$$C_{M(X)} = \frac{(T_{UP(X)} * C_{MR})}{60} \tag{4.6}$$

The unit processing time $T_{UP(X)}$ [minute] reflects the time it takes to set up the machine and the production quantity $Qty_{(X)}$ [piece] of object X with its specific NC-program. The NC-program specific machining time is stored for each object as the unit processing time per workpiece T_{TWp} [minute] in its setting sheet. T_{Pwp} is the production time per workpiece [min]. T_{NPwp} is the nonproductive time per workpiece [min]. The $T_{SetupTime(X)}$ [minute] is the onetime duration, that the operator takes to set up the machine with the correct clamping system. The unit processing time for each order is calculated as follows:

$$T_{UP(X)} = (T_{Pwp} + T_{NPwp}) * Qty_{(X)} + T_{SetupTime(X)} \tag{4.7}$$

The $C_{H(X)}$ and $C_{M(X)}$ values are independent from the position of object X in the machining queue. Their value is therefore easy to calculate and does not change once the quantity of the order has been set. $C_{H(X)}$ is the handling cost caused by the handling of the production quantity at a handling cost rate C_{WpR} [Euro/piece].

$$C_{H(X)} = C_{WpR} * Qty_{(X)} \tag{4.8}$$

The variable tool cost $C_{T(X)}$, on the other hand varies with the position in the machining queue and depends on the tools available after machining the preceding order. $C_{T(X)}$ multiplies the amount of required tools to be set up $TReq_{NetTotal(X)}$ [piece] which varies with the machining queue position, with the cost rate per tool C_{TR} [Euro/piece].

$$C_{T(X)} = TReq_{NetTotal(X)} * C_{TR} \tag{4.9}$$

Beside the costs, the MachineAgent offers the OrderAgent the total order execution time $T_{TOE(X)}$ [minute]. $T_{TOE(X)}$ is the total order execution time of object X. $T_{UP(Y)}$ is the unit processing time of all preceding orders, symbolized as objects Y.

$$T_{TOE(X)} = T_{UP(X)} + \sum_{Y \in \mathbb{J}} (T_{UP(Y)}) \tag{4.10}$$

where:

$$\mathbb{J} = \{\text{Objects in the machining queue before Object X}\}$$

The MachineAgent requests the $Time_{UntilSetup(X)}$ [minute] value from the SupplyAgent to ensure that the total order execution time of the preceding objects $T_{TOE(Y)}$ is longer than the time it takes to assemble, measure and commission the tools for object X. $Time_{UntilSetup}$ describes the time required for the cutting tool assembly, measurement and commission to the machine tool. This restriction shall prevent idle times, caused by missing or delayed cutting tools.

$$Time_{UntilSetup(X)} \leq \sum_{Y \in \mathbb{J}} T_{TOE(Y)} \tag{4.11}$$

The MachineAgent is also in charge of calculating the lifetime factor L_f [1] and to determine the translated tool use time based on the catalog lifetime. The lifetime factor L_f is calculated for each ToolID, which represents each tool type with a seven-digit number. While the ToolID identifies the tool type (for example a mill cutter with a diameter of 8 mm), the inventory number identifies a tool instances with a three-digit number.

The lifetime factor sets the NC-program related lifetime consumption $NCpLT$ [minute] into relation with the predicted catalog lifetime $CatLT$ [minute]:

$$L_{f(ToolID)} = \frac{CatLT_{(ToolID)}}{NCpLT_{(ToolID)}} \tag{4.12}$$

The NC-program specific tool use time $Tuse_{NCp}$ is then multiplied with the lifetime factor L_f [1] to transform the tool use time into a catalog lifetime basis $Tuse_{trans}$ [minute].

$$Tuse_{trans(ToolID)} = Tuse_{NCp(ToolID)} * L_{f(ToolID)} \tag{4.13}$$

The translated $Tuse_{trans(ToolID)}$ serves as the basis to calculate the gross tool requirement $TReq_{Gros}$ [minute] for each tool type, which is a minute value displaying the use time required to process the machining order. Starting from the gross tool requirement several calculations are made with the aim to determine the total net tool requirement $TReq_{NetTotal}$ [piece]. The total net tool requirement displays the number of newly assembled tools required to process a machining order, considered the magazine load and the stock at the warehouse at a given moment. It is needed to calculate the tooling cost C_T for each offered machining queue position. First, the available lifetime minus the critical lifetime of all cutting tool instances with the same ToolID at the machine tool magazine is reduced from $TReq_{Gros}$, resulting in the net tool requirement at machine $TReq_{NetMach}$ [minute]. Then the available lifetime of all cutting tool instances with the same ToolID in storage is reduced from $TReq_{NetMach}$, resulting in the net tool requirement in storage $TReq_{NetStor}$ [minute]. The $TReq_{Gros}$, $TReq_{NetMach}$ and $TReq_{NetStor}$ are minute values, describing the tool requirement based on the tool use time. $T_{NewT(ToolID)}$ is the lifetime value a newly assembled cutting tool of a specific tool type could offer. To turn the $TReq_{NetStor(ToolID)}$ minute value into a cutting tool quantity it is divided by $T_{NewT(ToolID)}$ [minute].

The gross tool requirement $TReq_{Gros(ToolID)}$ is determined by multiplying the translated tool use time $Tuse_{trans(ToolID)}$ with the workpiece quantity of machining order X.

$$TReq_{Gros(ToolID)} = Tuse_{trans(ToolID)} * Qty \tag{4.14}$$

The net tool requirement at the machine tool $TReq_{NetMach}$ reflects the required lifetime after considering the tools currently or in future use at the machine tool magazine. For each machining queue position the MachineAgent predicts, which previous order needs to be machined, which tools are at the machine tool and what their pre and post machining lifetime will be. $TReq_{NetMach}$ results from subtracting the currently available tool lifetime at the machine tool magazine from the $TReq_{Gros}$. The currently available tool lifetime of a cutting tool instance is its remaining lifetime $RemLT$ [minute] minus the critical lifetime, that shall remain unused $CritLT$ [minute]. $TReq_{NetMach}$ reflects which tool lifetime will still be needed to process the machining order after considering the magazine load for the specific machining queue position. The magazine load will be predicted based on the preceding machining orders.

$$TReq_{NetMach(ToolID)} = TReq_{Gros(ToolID)} - \sum_{q \in \mathbb{Q}} (RemLT_{(q)} - CritLT_{(ToolID)}) \tag{4.15}$$

where:

$$\mathbb{Q} = \{\text{tool instances with the required ToolID available at the magazine}\}$$

In the second step, the net tool requirement at the storage $TReq_{Storage}$ [minute] will be determined. $TReq_{Storage}$ results from subtracting the currently available tool lifetime in storage from $TReq_{NetMach}$. $TReq_{Storage}$ reflects which tool lifetime will still be needed to process the machining order, after considering the current cutting tool stock in storage.

$$TReq_{NetStor(ToolID)} = TReq_{NetMach(ToolID)} - \sum_{s \in \mathbb{S}} (RemLT_{(s)} - CritLT_{(ToolID)}) \tag{4.16}$$

where:

$$\mathbb{S} = \{\text{tool instances with the required ToolID available in storage}\}$$

If the net requirement after the storage $TReq_{NetStor(ToolID)}$ equals zero, no further tool use time and therefore no further tools are required to process the order. If the net requirement after the storage $TReq_{NetStor(ToolID)}$ is larger than zero, then $TReq_{NetTotal(ToolID)}$ is also larger than zero and has to be determined. While $TReq_{NetStor(ToolID)}$ is a minute value of the required tool use time for each ToolID, $TReq_{NetTotal(ToolID)}$ is the tool quantity, which covers the required tool use time. To derive the needed quantity of sister tools from $TReq_{NetStor}$, its value is divided by the lifetime of a newly assembled tool, called $T_{NewT(ToolID)}$ [minute]. $T_{NewT(ToolID)}$ reflects the lifetime that a newly assembled tool of the specific ToolID can offer. Therefore the catalog lifetime $CaTLT$ [minute] is used to determine $T_{NewT(ToolID)}$:

$$T_{NewT(ToolID)} = (CaTLT_{(ToolID)} - CritLT_{(ToolID)}) \tag{4.17}$$

With T_{NewT} [1] and a $TReq_{NetStor}$ [minute] larger than zero, the total net tool requirement $TReq_{NetTotal}$ can be determined as follows:

$$TReq_{NetTotal(ToolID)} = \frac{TReq_{NetStor(ToolID)}}{T_{NewT(ToolID)}}, \text{ for } TReq_{NetStor(ToolID)} \geq 0 \tag{4.18}$$

$TReq_{NetTotal(ToolID)}$ is the minimal number of tools required to machine a specific machining order for a specific machining queue position. It considers the current state in storage and the machine tool magazine, depending on the machining queue position. This tooling list is stored in a dynamic setting sheet, which enlists all tool instances needed from the storage and the tool components which must be assembled to a new tool instance. With $TReq_{NetTotal(ToolID)}$ determined by the MachineAgent, the MAS realizes the until now only theoretical net tool requirement. The sum of $TReq_{NetTotal(ToolID)}$ for all ToolIDs is communicated by the MachineAgent to the SupplyAgent as a supply order.

4.8.5 The SupplyAgent and its functionality

The SupplyAgent calculates the time until setup $Time_{UntilSetup}$, which is requested by the MachineAgent to ensure that the scheduled time until the machining of object X begins is longer than the time required to supply the tools.

$$Time_{UntilSetup(X)} = \sum_{Y \in \mathbb{J}} (Time_{UntilSetup(Y)} + Qty_e(Y) * f_{com} + Qty_n(Y) * (f_{assem} + f_{com})) \tag{4.19}$$

$\mathbb{J}$ = {Objects in the machining queue before Object X}

$Qty_e(Y)$ [piece] is the quantity of **e**xisting tools in storage available for Object Y

f_{com} is a minute value estimating the average commissioning time

$Qty_n(Y)$ [piece] is the quantity of **n**ewly assembled tools for Object Y [piece]

f_{assem} is a minute value estimating the average assembling time

CHAPTER 5

Scenarios tested on the final MAS design

The following chapter presents the designed multiagent system in practice. The first section presents the simplifying assumptions made to implement the MAS during this thesis. The second section presents the cooperation between those agents by four potential test scenarios. The MAS reactions during the four test scenarios is being described and cross-checked, to ensure a proper process flow.

5.1 Preface: Simplifying assumptions

For practical reasons further simplifying assumptions are necessary for the project's feasibility and to cut the time expense of checking calculations.

5.1.1 1. Simplification: Process simulation

As previously described the tool cycle process is simulated in JADE. This simplification is reasoned by the project's feasibility and three additional advantages. The first advantage is the intactness of the TU Wien Pilotfabrik hardware and software settings. The second advantage is the time and effort saved by reducing the implemented interfaces between JADE and the process assets to only one required interface between JADE and the TMS. The last advantage is that no recover strategy is necessary for a JADE simulation. A recovery strategy includes mechanisms to return to a safe and correct state in case of an exception, which is mandatory for a real-world implementation. A sufficient recovery strategy presupposes that negotiations between agents are collected and recovered in the event of a termination, for whatever reason it may occur. A simulation in JADE reduces the pressure to comply with such a mechanism, since evolving such strategies poses its own scientific questions. The MAS developed during this thesis has no recovery strategy. A termination with a preceding error message is accepted as a sufficient endpoint. Nevertheless, despite this simplification, all design choice are based on the real-world factory setting.

5.1.2 2. Simplification: Utilizing a single machine tool

In the present application case, more than one machine tool could be deployed, which promotes the extensibility of the system. However, at this stage of work it is decided to test the MAS with

only one deployed machine tool. This decision is reasoned by the challenging complexity that deploying more than one machine tool would bring for cross-examining the made MAS decisions. Considering a second (competing) machine tool would blur the comprehensibility of the system's response and outcome. One machine tool and therefore one MachineAgent is enough to evaluate if the MAS can reduce the tooling costs by sequencing the machining orders through negotiating agents.

5.1.3 3. Simplification: Utilizing only one database

It is common to consider redundant storage as a backup solution. Having a second database with a different set of data does not contribute to the system in any way, unless the shop floor is divided for either a geographical or an organizational reason. The implemented MAS design utilizes only one database.

5.1.4 4. Simplification: Reducing the magazine pockets from 30 to 10

The EMCO MAXXMill 500 has a drum tool magazine with 30 magazine pockets. In the first stage all calculations must be cross-examined, so the number of magazine pockets is limited to an amount of 10. This simplification lowers the number of machining orders necessary to cause the replacement of all tools at the magazine. Forcing a faster replacement of all tools increases the tool assembly rate and the system's dynamic significantly. A lower number of magazine pockets forces the system to sort the machining orders after a shorter time, to minimize the setups and the tooling costs.

5.1.5 5. Simplification: Limiting the workpiece quantity per order to less than 20 pieces

This simplification has three reasons; one is the higher number of machining orders, that can be processed without exhausting the order execution time. The second reason is the reduction of the magazine pockets, which makes it necessary to adapt the workpiece quantity. With the workpiece limit of 20 pieces at least three or four machining orders can be processed before a setup is required. Finally, the focus of this thesis lies on productions covering job shop and large-sized-series with high product variation, which is granted by limiting the workpiece quantity.

5.2 Agent cooperation tested by examined calculations

The cooperation of the designed agents is examined in four test scenarios. Each scenario aims for a defined output. If the MAS returns the desired output, it reacts in the desired manner. It is assumed, that the MAS will return the expected results even on a bigger scale, if all four test scenarios return the desired output. All four scenarios require an initial setting for the machine tool magazine and the storage, to predict possible outcomes. To limit the complexity to a processable amount, individual initial settings are for each scenario. This review focuses on the interaction between the MachineAgent and the OrderAgents.

5.2.1 Test Scenarios

To ease the comprehensibility of future calculations and to enable the reader to reproduce the calculations, table 5.1 summarizes all cutting tools in the mentioned NC-programs. The table lists the NC-program related tool use time $Tuse_{NCp(ToolID)}$. The NC-program specific lifetime

NC-program ID	Tool ID	Tool Description	NC-p.Lifetime	Cat Lifetime	L_f	$Tuse_{NCp(ToolID)}$	$Tuse_{trans(ToolID)}$
NC12072018	6585290	DrillD13	74,00	100,00	1,35	0,40	0,54
NC12072018	6585298	ShellMillD66	81,00	93,00	1,15	1,96	2,25
NC12072018	6585302	MillingCutterD9,25	68,00	70,00	1,03	0,85	0,88
NC12072018	6585295	CylindricalShaft	79,50	79,50	1,00	0,74	0,74
NC12072019	6585290	DrillD13	92,00	100,00	1,09	0,85	0,92
NC12072019	6585298	ShellMillD66	89,00	93,00	1,04	1,25	1,31
NC12072019	6585304	MillingCutterD31,71	59,00	59,00	1,00	1,45	1,45
NC12072019	6585292	FaceMillD60	80,00	85,00	1,06	1,64	1,74
NC12072019	6585289	ThreadMillM5X8	63,00	70,00	1,11	0,86	0,96
NC12072019	6585300	MillingCutterD16	89,00	96,00	1,08	1,34	1,45
NC12072019	6585299	MillingCutterD10	91,00	98,50	1,08	1,62	1,75
NC12072020	6585301	CountersinkD9	98,00	98,00	1,00	0,16	0,16
NC12072020	6585293	FaceMillD160	73,00	83,00	1,14	0,78	0,89
NC12072020	6585305	RoundShankDrillD8,5	58,50	58,50	1,00	0,64	0,64
NC12072020	6585306	DrillD12,7	57,50	57,50	1,00	0,48	0,48
NC12072020	6585300	MillingCutterD16	90,00	96,00	1,07	0,89	0,95
NC12072020	6585299	MillingCutterD10	91,00	98,50	1,08	0,58	0,62
NC12072020	6585289	ThreadMillM5X8	61,00	70,00	1,15	0,46	0,53
NC12072020	6585264	FaceMillD60	74,00	80,00	1,08	1,24	1,34

Table 5.1: Table with the NC-program based tool use times $Tuse_{NCp(ToolID)}$ in relation to the transformed use times $Tuse_{trans(ToolID)}$

and the catalog lifetime from the template setting sheet. The lifetime factor L_f, as well as the translated tool use time $Tuse_{trans(ToolID)}$ are calculated previously. The values from this table are used in all four scenarios.

5.2.2 Test Scenario I

Objective:
This scenario shall test if exceeding the maximum order execution time returns an error message from the OrderAgent. The exceeded time could have several reasons, such as a high number of preceding orders or an unrealistic forecast of the required order execution time. If no offer meets the set maximum order execution time, the OrderAgent terminates after reporting an error message. At this stage of work the MAS does not differentiate between causes for the exceeded time. The message outside the MAS could help production professionals to optimize their set goals.

Initial Setting:
For this test, two OrderAgents (OA and OB) are used. Both are in the frozen zone. Machining those two orders should exceed the maximum order execution time of the OrderAgent OC (see table 5.2). The machine tool magazine is filled with the cutting tools that table 5.3 displays. the current state at the stock is displayed by table 5.4. According to the tables 5.5, 5.6 and 5.7 the OrderAgent OA has a unit processing time of 50,40 minutes minus the 3 minutes of the already done setup ahead. The next order OB has a unit processing time of 47,24 minutes ahead of it. OrderAgent OC must wait 94,64 minutes until its setup could start. OC's maximum order execution time is 100 minutes. Since its total order execution time sums the 94,64 minutes of the preceding orders and its own unit processing time of 76,76 minutes. It exceeds its limit. OC has a time factor of 1,20, allowing it to exceed the order execution time by 20 percent of the original value. The total order execution time of 171,40 minutes still exceeds the given leeway, which leads us to the next test scenario.

OrderAgentID	Qty	NCProgrammID	Maximum production cost	Maximum order execution time	f	Status
OA	**10**	NC12072018	2000	200	1.45	machining
OB	4	NC12072019	3000	180	1.30	preparing
OC	12	NC12072020	2000	**100**	1.20	open

Table 5.2: OrderAgents for test scenario I.

Magazine load for scenario II						
Magazine Pos.	T ID	Tool Descript.	T Inv.No	Crit.TLifetime [min]	Cat.TLifetime [min]	Rem.Lifetime [min]
1	6585288	ShankMillD20	349	3,00	90,00	87,00
2	6585291	ThreadMillD17	350	5,00	85,00	65,00
3	6585297	SlottingCutterD124	351	2,00	100,00	89,00
4	6585303	MillingCutterD15,5	352	1,50	60,00	57,00

Table 5.3: Magazine load for scenario I

Assembled tools in stock for scenario II						
Magazine Pos.	T ID	Tool Descript.	T Inv.No	Crit.TLifetime [min]	Cat.TLifetime [min]	Rem.Lifetime [min]
1	6585300	MillingCutterD16	353	2,50	96,00	90,00
2	6585291	ThreadMillD17	354	5,00	85,00	73,00

Table 5.4: Stock of assembled tools for scenario I

5.2.3 Test Scenario II

Objective:
This test scenario shall test if the time gained with the time factor f is utilized properly. The time factor shall grant the OrderAgents more leeway. The time factor ranges from 0 to 2. An OrderAgent with a maximum time of 200 minutes and a time factor of 2 could, if required, extend its order execution time to up to 400 minutes.

Initial Setting:
The initial setting of test scenario I offers the perfect circumstances for this test. The initial OrderAgent parameters are adapted, so that the time factor can be tested (see table 5.8). As stated in the earlier test scenario, OrderAgent OC requires a total order execution time of 171,40 minutes (see 5.7). In this test scenario the time factor f is adapted from 1,20 to 1,80. Now the OrderAgent has an order execution time leeway of 80 percent, which means OC can and will accept the MachineAgent's offer for the third position in the machining queue.

5.2.4 Test Scenario III

Objective:
This scenario shall test if exceeding the production cost returns the desired exception. The production cost C_P, that the MachineAgent offers the OrderAgent, adds up to the order bound machining cost C_M, the handling costs C_H depending on the workpiece quantity and the tooling costs C_T reflecting the amount of tool setups. For this scenario, no fitting tool instances are available in storage. This means only the $TReq_{NetMach}$ and the $T_{NewT(ToolID)}$ must be considered to calculate the total net tool requirement $TReq_{NetTotal}$. This test scenario is built around the OrderAgents OD, OE and OE1, which are displayed in table 5.9. The orders OE and OE1 utilize the same NC-program, workpiece quantity, maximum production cost, time factor and maximum order execution time. Their machining cost is identical. OE1 follows OE, so the

OrderAgent OA related calculation for NC-program NC12072018						
Machining order related data		Machining related data		Extended tooling list		
Quantity	10	**Production time per workpiece [min]** T_{Pwp}	3,95	ToolID	Tool descr.	$Tuse_{NCp(ToolID)}$
Adapter	HSK 63	**None prod. time per workpiece [min]** T_{NPwp}	0,79	**6585290**	DrillD13	0,40
Setting Sheet ID	SettingSheet12072018	**Setup Time [min]**	3,00	**6585298**	ShellMillD66	1,96
Product	Prod.A	**Unit processing time [min]** T_{UP}	50,40	**6585302**	MillingCutterD9,25	0,85
Machine Tool	MaxxMill750			**6585295**	CyclindricalShaft	0,74
Machine ToolID	M007					

Table 5.5: OrderAgent OA related calculation for NC-program NC12072018

OrderAgent OB related calculation for NC-Program NC12072019						
Machining order related data		Machining related data		Extended tooling list		
Quantity	4	**Production time per workpiece [min]** T_{Pwp}	9,01	ToolID	Tool descr.	$Tuse_{NCp(ToolID)}$
Adapter	HSK 63	**None prod. time per workpiece [min]** T_{NPwp}	1,80	**6585290**	DrillD13	0,85
Setting Sheet ID	SettingSheet12072019	**Setup time [min]**	4,00	**6585298**	ShellMillD66	1,25
Product	Prod.B	**Unit processing time [min]** T_{UP}	47,24	**6585304**	MillingCutterD31,71	1,45
Machine Tool	MaxxMill750			**6585292**	FaceMillD60	1,64
Machine ToolID	M007			**6585289**	ThreadMillM5X8	0,86
				6585300	MillingCutterD16	1,34
				6585299	MillingCutterD10	1,62

Table 5.6: OrderAgent OB related calculation for NC-program NC12072019

OrderAgent OC related calculation for NC-Program NC12072020						
Machining order related data		Machining related data		Extended tooling list		
Quantity	12	**Production time per workpiece [min]** T_{Pwp}	5,23	ToolID	Tool descr.	$Tuse_{NCp(ToolID)}$
Adapter	HSK 63	**None prod. time per workpiece [min]** T_{NPwp}	1,00	**6585301**	CountersinkD9	0,16
Setting Sheet ID	SettingSheet12072020	**Setup time [min]**	2,00	**6585293**	FaceMillD160	0,78
Product	Prod.C	**Unit processing time [min]** T_{UP}	76,76	**6585305**	RoundShankDrillD8,5	0,64
Machine Tool	MaxxMill750			**6585306**	DrillD12,7	0,48
Machine ToolID	M007			**6585300**	MillingCutterD16	0,89
				6585299	MillingCutterD10	0,58
				6585289	ThreadMillM5X8	0,46
				6404264	StirnfäserD60	1,24

Table 5.7: OrderAgent OC related calculation for NC-program NC12072020

OrderAgentID	Qty	NCProgrammID	Maximum production cost	Maximum order execution time	f	Status
OA	10	NC12072018	2000	200	1.45	machining
OB	4	NC12072019	3000	180	1.30	preparing
OC	12	NC12072020	2000	**100**	**1.80**	open

Table 5.8: OrderAgents for test scenario II.

OrderAgentID	Qty	NCProgrammID	Maximum production cost	Maximum order execution time	f	Status
OD	10	NC12072018	59	200	1.45	machining
OE	4	NC12072019	28	180	1.30	preparing
OE1	4	NC12072019	28	180	1.30	open

Table 5.9: OrderAgents for test scenario III.

C_{WpR}[€]	C_{MR}[€]	C_{TR}[€]
5	10	10

Table 5.10: Cost rates for test scenario III and IV.

tooling cost C_T should be zero, unless the lifetime of a cutting tool expires. This scenario is designed so that two cutting tools in the magazine do not last in the machining of OD until the machining of OE. This causes a higher production cost for OE than for the identical upcoming OrderAgent OE1. This additional cost, caused by two tools to be set up results in exceeding the maximum production cost of the OrderAgent OE.

Initial Setting:
Table 5.9 lists the initiated OrderAgents. All three agents refer to NC-programIDs with already illustrated setting sheets (see scenario I). Table 5.11 displays the current magazine load of the machine tool for this scenario. Table 5.10 summarizes the set cost rates for scenario III and IV. The machine tool magazine is filled as listed in table 5.11. The two tool instances in positions one and two (Inv.No 355 and 356), are required in all three orders. As the catalog tool life of both suggests (100 minutes and 93 minutes) the actual set lifetime for this scenario is comparatively very low with 10 and 30 minutes. Those values are set low to ensure that both tools are worn out until order OE1 can begin processing, thus raising its tooling cost.

The magazine pocket no.10 is empty and should have no effect on the test scenario since the MachineAgent checks the magazine load from OrderAgent to order and not over the range of more than three orders. Table 5.12, 5.13, 5.14 and 5.15 display how the MachineAgent predicts the tool requirement. The predicted production cost of each OrderAgent is displayed in table 5.16. Table 5.15 shows that the tool type 6585290 in magazine position one needs a sister tool, since $TReq_{NetMach}$ is larger than zero. The storage holds no fitting tool instance, $TReq_{NetTotal}$ equals $TReq_{NetMach}$. Since magazine slot no. 10 is empty, the MachineAgent fills it with a sister tool (drill diameter 13 mm). The MachineAgent has no tool changing strategy yet, it therefore picks the next free magazine slot. After that the MachineAgent picks the next magazine slot with a tool that is not required in the upcoming machining order. The MachineAgent schedules magazine position four for the setup. If a different OrderAgent would be assigned to the third position in the machining queue, which needs the tool type 6585295, the MachineAgent would change the tool in the fifth magazine slot.

This test scenario is designed to use only three OrderAgents to disable the MachineAgents capability to sort the OrderAgents. This decision is reasoned in providing the reader with a simple example to understand the calculation of $TReq_{NetTotal}$ and of the production cost

Magazine load for scenario III						
Magazine Pos.	T ID	Tool Descript.	T Inv.No	Crit.TLifetime [min]	Cat.TLifetime [min]	Rem.Lifetime [min]
1	**6585290**	**DrillD13**	**355**	2,00	100,00	**10,00**
2	**6585298**	**ShellMillD66**	**356**	2,95	93,00	**30,00**
3	**6585302**	MillingCutterD9,25	306	2,50	70	68,00
4	**6585295**	CyclindricalShaft	307	2,95	79,50	79,50
5	**6585304**	MillingCutterD31,71	299	2,90	59,00	59,00
6	**6585292**	FaceMillD60	300	3,50	85	80,00
7	**6585289**	ThreadMillM5X8	301	4,00	70	63,00
8	**6585300**	MillingCutterD16	302	2,50	96,00	89,00
9	**6585299**	MillingCutterD10	303	3,00	98,50	91,00
10	empty	empty	empty	empty	empty	empty

Table 5.11: Magazine load for scenario III

C_P. The MachineAgent predicted the tool wear for each machining queue position and each order after OD and before OE a sister tool for 6585290 and 6585298 and initiates two tool setups. If the $TReq_{NetMach}$ exceeds the minute value of the lifetime a newly assembled tool could offer, then the $TReq_{NetTotal}$ rises to two or more pieces. As the tables 5.13 and 5.14 show the $TReq_{NetMach}$ of the drill and the mill is higher than zero, with 1,08 and 0,44 minutes. Examining table 5.12 shows, that the mill is near its critical lifetime limit after OD. Independ from the remaining lifetime, the MachineAgent only pursues a tool setup, if it is necessary for an upcoming machining order.

Due to the high production cost of OE, this test scenario returns an exception. This OrderAgent has an ERP-given maximum production cost of 28 Euros and faces due to the dual tool setup an additional cost of 20 Euros (see table 5.16). OrderAgent OE1 has a production cost of 27,88 Euros and stays under the requested 28 Euros. OrderAgent OE can not and will not accept the offered production price of the MachineAgent. As a further consequence, it terminates right after submitting its error message, displaying its reason.

Magazine state									
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**355**	2,00	**10,00**	0,54	5,40	8,00	0	4,60
2	**6585298**	**356**	2,95	**30,00**	2,25	22,25	27,05	0	7,75
3	**6585302**	306	2,50	68,00	0,88	8,80	65,5	0	59,20
4	**6585295**	307	2,95	79,50	0,74	7,40	76,55	0	72,10
5	**6585304**	299	2,90	59,00	x	x	56,10	x	x
6	**6585292**	300	3,50	80,00	x	x	76,50	x	x
7	**6585289**	301	4,00	63,00	x	x	59,00	x	x
8	**6585300**	302	2,50	89,00	x	x	86,50	x	x
9	**6585299**	303	3,00	91,00	x	x	88,00	x	x
10	empty	empty	empty	empty	empty	empty	empty	empty	empty

Table 5.12: Predicted magazine tool wear before processing OD

Magazine state									
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**355**	2,00	**4,60**	0,92	3,68	2,60	1,08	≡ CritLT
2	**6585298**	**356**	2,95	**7,75**	1,31	5,24	4,80	0,44	≡ CritLT
3	**6585302**	306	2,50	59,20	1,45	5,80	56,70	0	53,40
4	**6585295**	307	2,95	72,10	Not Required	x	x	x	x
5	**6585304**	299	2,90	59,00	Not Required	x	x	x	x
6	**6585292**	300	3,50	80,00	1,74	6,96	76,50	0	73,04
7	**6585289**	301	4,00	63,00	0,96	3,84	59,00	0	59,16
8	**6585300**	302	2,50	89,00	1,45	5,80	86,50	0	83,20
9	**6585299**	303	3,00	91,00	1,74	6,96	88,00	0	84,04
10	empty	empty	empty	empty	empty	empty	empty	empty	empty

Table 5.13: Predicted magazine tool wear after processing OD and before OE

Magazine state									
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**355**	2,00	**4,60**	0,92	3,68	2,60	1,08	≡ CritLT
2	**6585298**	**356**	2,95	**7,75**	1,31	5,24	4,80	0,44	≡ CritLT
3	**6585302**	306	2,50	59,20	1,45	5,80	56,70	0	53,40
4	**6585290**	350	2,00	**100**	0,92	3,68	98,00	0	96,32
5	**6585304**	299	2,90	59,00	Not Required	x	x	x	x
6	**6585292**	300	3,50	80,00	1,74	6,96	76,50	0	73,04
7	**6585289**	301	4,00	63,00	0,96	3,84	59,00	0	59,16
8	**6585300**	302	2,50	89,00	1,45	5,80	86,50	0	83,20
9	**6585299**	303	3,00	91,00	1,74	6,96	88,00	0	84,04
10	6585290	351	2,95	93,00	1,31	5,24	90,05	0	87,76

Table 5.14: Predicted magazine tool wear after processing OD and before OE, revised

Magazine state									
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	empty	empty	empty	empty	empty	empty	empty	empty	empty
2	empty	empty	empty	empty	empty	empty	empty	empty	empty
3	**6585302**	306	2,50	53,40	1,45	5,80	50,90	0	47,60
4	**6585290**	350	2,00	**96,32**	0,92	3,68	94,32	0	96,32
5	**6585304**	299	2,90	59,00	Not Required	x	x	x	x
6	**6585292**	300	3,50	73,04	1,74	6,96	69,54	0	66,08
7	**6585289**	301	4,00	59,16	0,96	3,84	55,16	0	55,32
8	**6585300**	302	2,50	83,20	1,45	5,80	80,70	0	77,40
9	**6585299**	303	3,00	84,04	1,74	6,96	81,04	0	77,08
10	**6585290**	351	2,95	87,76	1,31	5,24	84,81	0	82,52

Table 5.15: Predicted magazine tool wear before and after OE1

MachiningOrderID	Qty	C_H[€]	C_M[€]	C_T[€]	C_P[€]
OD	10	50,00	8,40	0	58,40
OE	4	20,00	7,88	20	**47,88**
OE1	4	20,00	7,88	0	27,88

Table 5.16: Predicted machining cost for test scenario III

OrderAgentID	Qty	NCProgrammID	Maximum production cost	Maximum order execution time	f	Status
OF	8	NC12072019	59	100	1.45	machining
OG	4	NC12072019	30	150	1.30	preparing
OH	4	NC12072020	80	200	1.30	open
OI	4	NC12072018	80	200	1.30	open

Table 5.17: OrderAgents for test scenario IV.

Magazine Load for scenario IV						
Magazine Pos.	T ID	Tool Descript.	T Inv.No	Crit.TLifetime [min]	Cat.TLifetime [min]	Rem.Lifetime [min]
1	**6585290**	DrillD13	357	2,00	100,00	92,00
2	**6585298**	ShellMillD66	358	2,95	93,00	89,00
3	**6585304**	MillingCutterD31,71	299	2,90	59,00	59,00
4	**6585292**	FaceMillD60	300	3,50	85,00	80,00
5	**6585289**	ThreadMillM5X8	301	4,00	70,00	63,00
6	**6585300**	MillingCutterD16	302	2,50	96,00	89,00
7	**6585299**	MillingCutterD10	303	3,00	98,50	91,00
8	6585297	SlottingCutterD124	351	2,00	100,00	89,00
9	6585301	CountersinkD9	359	2,80	98,00	88,00
10	6585295	CylindricShaftMillD10	360	2,95	79,50	75,50

Table 5.18: Magazine load for scenario IV

5.2.5 Test Scenario IV

Objective: The last scenario tests if the MAS can sort the OrderAgents in a suitable sequence. This is accomplished by the negotiation between the OrderAgents and the MachineAgent. This scenario is built on the already known NC-programs, but different than the other test scenarios, it has four instead of three OrderAgents. Since two OrderAgents are in the frozen zone (OF and OG), two additional OrderAgents are required to test the scheduling function. For this test scenario four OrderAgents are generated, as displayed in table 5.17.

Initial Setting:
Table 5.18 lists all cutting tools stored in the magazine before the arrival of order OF. For this test scenario it is assumed, that all required tool instances are on stock and no assembly is necessary. Table 5.19 lists all required tools at the storage.

Tools at stock for scenario IV						
Magazine Pos.	T ID	Tool Descript.	T Inv.No	Crit.TLifetime [min]	Cat.TLifetime [min]	Rem.Lifetime [min]
1	**6585264**	FaceMillD60	335	4,00	80,00	66,68
2	**6585306**	DrillD12,7	311	1,80	57,50	57,50
3	**6585305**	RoundShankDrillD8,5	310	1,85	58,50	58,50
4	**6585293**	FaceMillD160	340	3,00	83,00	83,00

Table 5.19: Tools at stock for scenario IV

OF and OG are in the frozen zone and use the same NC-program, NC12072019. OH and OI have different NC-programs (see table 5.17) and need to be sorted. Table 5.1 at the beginning

of this section, lists all required cutting tools for the three NC-programs. Both OrderAgents have a workpiece quantity of four, which sets the fixed cost for both orders to an equal value of 20 Euros. The machining cost for OH are 4,15 Euros and 3,16 Euros for OI, which reflects the time the order occupies the machine tool. The difference is low compared to the tooling cost. The predicted magazine load and the remaining lifetimes before processing OrderAgent OF are illustrated in table 5.21. The predicted magazine load and the remaining lifetimes of the tools in the magazine before machining OrderAgent OG are displayed in table 5.22. The first possible sequence is called A and lines up as follows: OF, OG, OH and OI. The second possible sequence is B with the following OrderAgent line up: OF, OG, OI and OH. Table 5.21 and 5.22 describe the predicted magazine state for the frozen zone and serve for both possible sequences A and B as the calculation entry point. Table 5.23 shows how OH in the third position of sequence A would effect the magazine state and which tool setups would be required. The cutting tools in magazine position 6, 7 and 9 (colored green) are cutting tools from the preceding OrderAgent OG, that OH is also using. The cutting tools colored red and marked with "instead" are cutting tools required for the OH OrderAgent but not yet available at the magazine. Position 1 to 4 in the table (colored grey) are tools, which will be replaced by the required tools (colored red). Offering OH the third machining queue position requires four tools to be set up. Table 5.24 displays the magazine state after OH is machined and before IO arrives in the fourth machining queue position. IO shares only one tool type with OH (colored green) but requires three tools to be set up (colored red). Those three tools replace the cutting tools in magazine positions 1, 2 and 3. Adding the tool setup for OH in the third and IO in fourth position, sequence A requires a total sum of seven tools to be set up.

The B sequence (see table 5.25) sets OI in third position. OrderAgent OI shares three tool IDs with OG (colored green) and needs only one different tool type to be set up. Which makes clear that OI on the second position in the queue needs only one tool to be set up, while OH would needs four tools to be changed for the same queue position. OH in fourth position (see 5.26) requires four tools to be set up, which makes a sum of five tools to be set up. The MachineAgent is designed to not differentiate between cutting tools, which are replaced due to an expiring lifetime or tools that are required for the next job. The MachineAgent focuses on minimizing the number of tools to be set up. This means that the sum of required tool-changes weights heavier than the number of similar tools. For example, in the sequence A, OG and OH share three same tool types, but OH has four different tools compared to OG. Sequence B requires a sum of five tools to be set up, which is smaller than the number of tools required to be set up for the sequence A (seven tools). The summary of the individual costs is displayed in table 5.20. It shows that the desired output of this test is the sequence: OF, OG, OI and then OH. It also displays the revised offer, that the MachineAgent will make to the OrderAgent OH to waive the third position, if OH has already accepted this position. The revised offer always includes a price reduction. This price reduction is always the 50% of cost savings achieved by a smaller tool setup.

Comparing the Production cost of Sequence A and B								
OrderAgent	NC-program	Qty	C_M	C_H	Tools Qty changed	C_T	C_P	NewOffer
OF	NC12072019	8	14,41	40	0	0	54,41	None
OG	NC12072019	4	7,21	20	0	0	27,21	None
OH	NC12072020	4	4,15	20	4	40	64,15	None
OI	NC12072018	4	3,16	20	3	30	53,16	None
OF	NC12072019	8	14,41	40	0	0	54,41	None
OG	NC12072019	4	7,21	20	0	0	27,21	None
OI	NC12072018	4	3,16	20	1	10	33,16	None
OH	NC12072020	4	4,15	20	4	40	64,15	54,15

Table 5.20: Comparison of the production cost caused by sequence A and B

	Magazine state								
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**357**	2,00	**92,00**	0,92	7,36	90,00	0	84,64
2	**6585298**	**358**	2,95	**89,00**	1,31	10,48	86,05	0	78,52
3	**6585304**	**299**	2,90	**59,00**	1,45	11,60	56,10	0	47,40
4	**6585292**	**300**	3,50	**80,00**	1,74	13,92	76,50	0	66,08
5	**6585289**	**301**	4,00	**63,00**	0,96	7,68	59,00	0	55,32
6	**6585300**	**302**	2,50	**89,00**	1,45	11,60	86,50	0	77,40
7	**6585299**	**303**	3,00	**91,00**	1,75	14,00	88,00	0	77,00
8	**6585297**	**351**	2,00	**89,00**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
9	**6585301**	**359**	2,80	**88,00**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
10	**6585295**	**360**	2,95	**75,50**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.

Table 5.21: Predicted magazine tool wear before processing OF on position 1

	Magazine state								
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**357**	2,00	**84,64**	0,92	3,68	82,64	0	80,96
2	**6585298**	**358**	2,95	**78,52**	1,31	5,24	75,57	0	73,28
3	**6585304**	**299**	2,90	**47,40**	1,45	5,80	44,50	0	41,60
4	**6585292**	**300**	3,50	**66,08**	1,74	6,96	62,58	0	59,12
5	**6585289**	**301**	4,00	**55,32**	0,96	3,84	51,32	0	51,48
6	**6585300**	**302**	2,50	**77,40**	1,45	5,80	74,90	0	71,60
7	**6585299**	**303**	3,00	**77,00**	1,75	7,00	74,00	0	70,00
8	**6585297**	**351**	2,00	**89,00**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
9	**6585301**	**359**	2,80	**88,00**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
10	**6585295**	**360**	2,95	**75,50**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.

Table 5.22: Predicted magazine tool wear before processing OG

Magazine state									
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**357**	2,00	**80,96**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
2	**6585298**	**358**	2,95	**73,28**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
3	**6585304**	**299**	2,90	**41,60**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
4	**6585292**	**300**	3,50	**59,12**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
5	**6585289**	**301**	4,00	**51,48**	0,53	2,12	47,48	0	49,36
6	**6585300**	**302**	2,50	**71,60**	0,95	3,80	69,10	0	67,80
7	**6585299**	**303**	3,00	**70,00**	0,62	2,48	67,00	0	67,52
8	**6585297**	**351**	2,00	**89,00**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
9	**6585301**	**359**	2,80	**88,00**	0,16	0,64	85,20	0	87,36
10	**6585295**	**360**	2,95	**75,50**	Not Req.	Not Req.	Not Req.	Not Req.	Not Req.
instead 1	**6585293**	**340**	3,00	**83,00**	0,89	3,56	80,00	0	79,44
instead 2	**6585305**	**310**	1,85	**58,50**	0,64	2,56	56,65	0	55,94
instead 3	**6585306**	**311**	1,80	**57,50**	0,48	1,92	55,65	0	55,58
instead 4	**6585264**	**335**	4,00	**66,68**	1,34	5,36	62,68	0	61,32

Table 5.23: Sequence A: Predicted magazine tool wear before processing OH on position 3, require setup, use same tools test

Magazine state									
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585293**	**340**	3,00	**79,44**	Not Req.	Not Req.	Not Req.	0	Not Req.
2	**6585305**	**310**	1,85	**55,94**	Not Req.	Not Req.	Not Req.	0	Not Req.
3	**6585306**	**311**	1,80	**55,58**	Not Req.	Not Req.	Not Req.	0	Not Req.
4	**6585264**	**335**	4,00	**61,32**	Not Req.	Not Req.	Not Req.	0	Not Req.
5	**6585289**	**301**	4,00	**49,36**	Not Req.	Not Req.	Not Req.	0	Not Req.
6	**6585300**	**302**	2,50	**67,80**	Not Req.	Not Req.	Not Req.	0	Not Req.
7	**6585299**	**303**	3,00	**67,52**	Not Req.	Not Req.	Not Req.	0	Not Req.
8	**6585297**	**351**	2,00	**89,00**	Not Req.	Not Req.	Not Req.	0	Not Req.
9	**6585301**	**359**	2,80	**87,36**	Not Req.	Not Req.	Not Req.	0	Not Req.
10	**6585295**	**360**	2,95	**75,50**	0,74	2,96	72,55	0	72,54
instead 1	**6585302**	**337**	2,50	**68,00**	0,88	3,52	65,50	0	64,48
instead 2	**6585290**	**357**	2,00	**80,96**	0,54	2,16	79,96	0	78,80
instead 3	**6585298**	**358**	2,95	**73,28**	2,25	9,00	70,33	0	70,33

Table 5.24: Sequence A: Predicted magazine tool wear before processing OI on position 4

					Magazine state				
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**357**	2,00	**80,96**	0,54	2,16	78,96	0	78,80
2	**6585298**	**358**	2,95	**73,28**	2,25	9,00	70,33	0	64,28
3	**6585304**	**299**	2,90	**41,60**	Not Req.	Not Req.	Not Req.	0	Not Req.
4	**6585292**	**300**	3,50	**59,12**	Not Req.	Not Req.	Not Req.	0	Not Req.
5	**6585289**	**301**	4,00	**51,48**	Not Req.	Not Req.	Not Req.	0	Not Req.
6	**6585300**	**302**	2,50	**71,60**	Not Req.	Not Req.	Not Req.	0	Not Req.
7	**6585299**	**303**	3,00	**70,00**	Not Req.	Not Req.	Not Req.	0	Not Req.
8	**6585297**	**351**	2,00	**89,00**	Not Req.	Not Req.	Not Req.	0	Not Req.
9	**6585301**	**359**	2,80	**88,00**	Not Req.	Not Req.	Not Req.	0	Not Req.
10	**6585295**	**360**	2,95	**75,50**	0,74	2,96	72,55	0	72,54
instead 3	**6585302**	**337**	2,50	**68,00**	0,88	3,52	65,50	0	64,48

Table 5.25: Sequence B: Predicted magazine tool wear before processing OI on position 3, require setup, use same tools

					Magazine state				
MagazinePos	**ToolID**	**InvNo**	**CritLT**	**RemLT**	$Tuse_transl$	$TReq_Gross$	$RemLT - CritLT$	$TReq_NetMach$	**Rem.LTPostOrder**
1	**6585290**	**357**	2,00	**78,80**	Not Req.	Not Req.	Not Req.	0	Not Req.
2	**6585298**	**358**	2,95	**64,28**	Not Req.	Not Req.	Not Req.	0	Not Req.
3	**6585302**	**337**	2,50	**64,48**	Not Req.	Not Req.	Not Req.	0	Not Req.
4	**6585292**	**300**	3,50	**59,12**	Not Req.	Not Req.	Not Req.	0	Not Req.
5	**6585289**	**301**	4,00	**51,48**	0,52	2,12	47,48	0	49,36
6	**6585300**	**302**	2,50	**71,60**	0,95	3,80	69,10	0	67,80
7	**6585299**	**303**	3,00	**70,00**	0,62	2,48	67,00	0	67,52
8	**6585297**	**351**	2,00	**89,00**	Not Req.	Not Req.	Not Req.	0	Not Req.
9	**6585301**	**359**	2,80	**88,00**	0,16	0,64	85,20	0	87,36
10	**6585295**	**360**	2,95	**72,54**	Not Req.	Not Req.	Not Req.	0	Not Req.
instead 1	**6585293**	**340**	3,00	**83,00**	0,89	3,56	80,00	0	79,44
instead 2	**6585305**	**310**	1,85	**58,50**	0,64	2,56	56,65	0	55,94
instead 3	**6585306**	**311**	1,80	**57,50**	0,48	1,92	55,65	0	55,58
instead 4	**6585264**	**335**	4,00	**66,68**	1,34	5,36	62,68	0	61,32

Table 5.26: Sequence B: Predicted magazine tool wear before processing OH on position 4, require setup, use same tools

CHAPTER 6

Critical reflection and outlook

This chapter critically reviews the DM-MESD in four stages. First it examines if the design methodology fulfills the domain specific restrictions, and the requirements for a design methodology. In the third stage the final MAS design is reviewed by its resulting MAS design. The last stage offers the reader a critical summary and an outlook to future possibilities.

6.1 Examining the list of requirements

6.1.1 Domain specific requirements

The DM-MESD must fulfill the predefined domain specific requirements, which a MAS design methodology for the manufacturing execution system domain must fulfill. By considering those domain specific requirements the DM-MESD differs from other the common MAS design methodologies. This section reviews if the DM-MESD has met all the domain specific requirements.

Highly process controlled
The DM-MESD prioritizes the core process and ensures its flow. Right at the beginning, the design methodology defines and maps the core process and assure its proper operation during the design process. Although the DM-MESD also considers objects, its design derives from focusing on tasks and functions. This priority forces the designer to capture the required functions of the core process and ensures them. The DM-MESD also captures the required inputs and outputs, which include constraints. Defining a core process and focusing on the well-functioning of its sub steps and tasks centralizes the process perspective instead of a role or responsibility perspective.

Mutual dependency
The design methodology must incorporate concepts like autonomy, proactiveness and goal orientation, without risking the process operation. The DM-MESD considers the mutual dependencies occurring during the core process. By capturing all required inputs and outputs of the tasks and functions and building services upon those structures, mutual dependencies are considered, while at the same time encapsulation is promoted. The resulting services facilitate the integration of MAS concepts, such as autonomy, proactiveness and goal orientation, and consider the mutual dependencies of the original core process.

Sensitive System
The design methodology and its resulting MAS design must be located on a high MES layer. The design methodology thereby realizes a loose coupling of the process relevant hardware and leaves a buffering zone between the MAS and safety relevant mechanisms. This approach defines an investigation area, which meets those demands. This investigation area is integrated into the evaluation phase of the DM-MESD, which implies that the application domain is tested right at the beginning, if it is located on a higher MES layer. The DM-MESD therefore meets this domain specific requirement.

Highly heterogenic hard and software structure
This requirement demands that a MAS must be either applied on a platform independent solution or on a platform with multiple interconnectivity possibilities. The DM-MESD neither supports nor neglects this issue, as it only offers a pure design methodology without any implementation suggestions. It does not specify the agent's internal structure, nor does it specify the interaction by offering protocol templates. The programmer can therefore choose the platform and the implementation freely or consider future technical requirements for the application.

The DM-MESD methodology fulfills all requirements in a sufficient manner.

6.1.2 General requirements for a design methodology

Model fidelity
The model fidelity validates if the model suits its purpose. Early in this thesis the author stated that the model fidelity is being verified through the capability of the resulting MAS to maintain the core process. As the examined test scenarios show, the final MAS design maintains the core process and extends it in beneficial manner. This implies, that the DM-MESD maps the process in a satisfactory manner. The DM-MESD therefore complies with the requirement of model fidelity.

Prescriptiveness
The DM-MESD guides the designer through all six phases. It describes the purpose of each phase, as well as its starting point. The DM-MESD offers the designer a designing process building on the results of the previous step. The continued assistance is achieved by a guided procedure, where the result of the previous step is the input for the next step. The design methodology presented here offers not only a precise description of the single design operations, but also of the initial conditions and the final outputs. The DM-MESD offers, different from the other MAS design methodologies, an evaluation phase, so that the designer can be sure that the DM-MESD is a proper fit for the chosen application process.

Feasibility for industrial and mechanical engineers
In the investigation area that the DM-MESD defines, mechanical and industrial engineers are commonly the stakeholders. Its design strategy and the chosen modeling tools are suitable for this target group. By using textual description and structural analysis the DM-MESD corresponds to their commonly minimal modeling knowledge. Further, this approach favors the familiar process-oriented perspective during the analysis phase, instead of using the object-oriented perspective. Although the DM-MESD relies on textual description and structured analysis, it provides proper models and methods, as the application on the integrated tool cycle showed. The designer is not expected to consider any implementation relevant aspects during applying the DM-MESD, as it is a pure design methodology. The DM-MESD therefore fulfills this requirement.

Capability to model agent-oriented systems in the MES layer
The DM-MESD constitutes MAS concepts in its methods, without expecting any previous MAS knowledge from the designer. Such concepts are encapsulation, service orientation, pro-activeness, cooperation, and competitiveness. Encapsulation is realized during the use of the DM-MESD, by structuring the system according to tasks and functions of the core process. Those tasks and functions are later expanded to self-containing services. The process-perspective and the building of self-containing services from tasks and functions underlines its service-oriented nature. In defining objectives while structuring service clusters, the DM-MESD lays the foundation for proactive behavior. However, it must be stated that although pro-activeness is a design choice, its realization is rather an implementation than a design matter. In designing service clusters and their resulting agents to cooperate and compete to realize set objectives, the DM-MESD embeds those concepts. Although the DM-MESD does not consider the full range of all MAS related concepts, it can be said, that it at least supports and realized the above mentioned. This requirement is therefore considered fulfilled.

6.2 Review of the final MAS design

The assessment phase and analysis phase of the DM-MESD ensures the original functionality of the integrated tool cycle, while the abstraction and the clustering phase-initiated improvements. The aim at the beginning of the design process was to utilize the possibilities that the integrated tool cycle had to offer. The author has chosen a multiagent system as a control structure. While developing a MAS the author quickly realized the need for a MAS design methodology conceptualized and designed for the manufacturing domain and its stakeholders.

During the application of the DM-MESD methodology on the integrated tool cycle, the dependencies between the process entities were captured in a precise textual manner as input and outputs. The new services, such as "inquiring an offer", "proposing offer" or "managing setup", that have been developed during the design process with the DM-MESD, are a beneficial extension of the original process functionality. These services suggest that an advantageous process transformation can be achieved with this method and its integrated MAS concept. The services "inquiring an offer", "proposing offer" and "accepting offer" are services, which result from the negotiation as a multiagent concept.

By creating services with clearly defined inputs and outputs, the concept of encapsulation was also implemented and applied by the methodology. Building on tasks and functions, the small building blocks from which services are created, includes in its essence the service-oriented approach. But only the concepts of proactivity, cooperation and competitiveness make it possible to tap the potential of the developed services. These concepts are determined by conceptualizing the agents' response to their environment and other agents.

The creation of the new services was initiated by the definition of the KPI and the overall objectives. By setting the overall goal and gradually approaching it, the use case shows, that the methodology's perceptiveness can be concluded. Based on these measurable KPIs, it can be evaluated whether the hoped-for advantages could be achieved through a MAS design. During the application of the DM-MESD, a flexible order schedule based on the minimal tooling cost was defined as KPIs. The test scenario IV shows, that the final MAS can sequence the machining order in a manner that minimizes the tooling costs. The flexible machine utilization is achieved by the previously mentioned MAS concepts and through combining the new developed services. While newly developed services such as, "Inquiring offer", "Accepting offer" and "Proposing offer" were responsible for finding the best possible schedule by negotiation, already existing services such as, "Managing machining" and "Managing setup" ensured the provision of resources. In

addition to the advantages achieved by the MAS created with DM-MESD, the methodology is also judged by the documentation of the final MAS design. The final design is achieved through the description of the individual agent types and their functions and services. The interaction between the agents is represented by a textual sequence, sequence diagram and an interactions diagram. While the agents, their functionalities and their interactions were mapped, the resulting MAS design leaves room for the implementation choices made by the programmer.

6.3 Conclusive summary

The MAS design methodology presented here utilizes two UML diagrams as optional support. It is however centered on textual description and a structured analysis mechanism. The combination of both enables the designer to easily map the complete system and in a most natural way, which would require an experience modeler if for example UML would be used. This methodological path was chosen, since process stakeholders from the manufacturing domain are the future designers. Those stakeholders are most likely industrial and mechanical engineers, with little to no modeling experience. The DM-MESD is based on textual description and structured analysis, such as Gaia, to avoid the required model language extensions. PASSI for example uses UML agent extension (AUML). To ease the modeling for the target audience, the DM-MESD does not use UML, AUML or any other modeling language. This is reasoned in the effort required to specify or to extend a modeling language. Doing so requires adapting meta models, and a detailed description of notations and syntax checks. The methodological tools, such as the textual description is practiced easily and without further experience. Further, the DM-MESD enables the designer to map the process completely and prevents misunderstandings. In fact, the use of the textual description does not require any modeling knowledge and enables the process stakeholder to design a multiagent system. The DM-MESD also enables several persons to work simultaneously on a design, as the textual description can be written by the process stakeholder and then be handed over to the designer. The designer could then base her or his design process on the detail process captured in text. If required in further steps the designer could consult with the process stakeholder, for example during the clustering phase.

Like Gaia, this methodology does not offer or suggest an implementation mechanism nor a pattern. It neither offers protocol templates, to be transformed into FIPA ACL protocols. Offering protocol types in the DM-MESD would require a deeper understanding of the ACL from the designer. Trying to fit interactions into fixed communication protocols would influence the MAS design at an early stage and is not suitable for the target audience. It was therefore assumed that the final MAS design will be implemented by a computer scientist or a programmer with the capabilities and the experience to do so. Different from PASSI, the DM-MESD makes no implementation suggestions nor does it encounter any implementation mechanism, which would limit the implementation possibilities of the programmer. The DM-MESD does not dictate any protocol specifications. Although MAS for integrated tool cycle was realized in JADE, any other MAS platform could have been used.

The DM-MESD takes the manufacturing restrictions such as limited resources, process sequences, interfaces, etc. into account. Due to its focus on the core process and its process analysis phase, it also considers the given process restrictions and limitations. It offers the designer a methodological path including mechanisms and guidelines throughout the complete designing process. The DM-MESD offers the designer an evaluation phase right at the beginning, to verify the proper suitability for the application domain and to save time and effort. During the designing steps, new perspectives on the application process were elaborated and offered the possibility of a process transformation. Offering mechanisms to restructure the process, such as

developing KPI and considering them during the creative design phase enhanced the process. Differing from other methodologies, this methodology values the core process functionality above all and ensures its implementation right from the beginning. This is achieved by considering the required tasks, the functions, inputs and outputs during the analysis and the design phase. The DM-MESD also offers multiple mechanisms, such as KPI, service identification and service clustering to improve and transform the process.

6.4 Outlook

As a next step the DM-MESD can be applied to different application processes to document its performance and to improve it. This additional practice would clarify if application processes exist, which fit the investigation area, but still are not suitable for the DM-MESD. The additional use would evaluate if the investigation area of the DM-MESD requires improvement. In the future, the DM-MESD could be presented to industrial partner companies to gather field information directly from the stakeholders. This next step would help improve and develop the DM-MESD further according to the stakeholders' needs. Additional use cases would also help gather information from programmers to evaluate if implementation guidance from DM-MESD is desired.

As a future step the resulting MAS can be improved to map the different parts of the tooling cost in detail. Detailing the tooling costs offers the production a new insight into unexploited saving potentials.

Another future task is the improvement of the agent types and their interactions. The OrderAgents could be improved to compete directly against each other. Their algorithm could be enhanced so that they base their decisions on the reactions of the competition. Instead of either accepting or denying an offer, the future OrderAgent could start bargaining, for example offering to wait a bit longer for a better price. The MachineAgent on the other hand could be improved in three ways. Those three ways are either improving the competition between the MachineAgents or improving the negotiation between the OrderAgent or improving the tool management of the MachineAgent. In the first possibility, the direct competition between more than one MachineAgent could be enhanced. Two competing MachineAgents could adapt their prices to get the bid, as a form of direct competition. Another form of competition between two MachineAgents could also be indirect competition, due to their availability or current magazine load. In the second possibility, the algorithm of the MachineAgent could be enhanced to enable bargaining with the OrderAgent in order achieve either a better machine utilization or to save production costs. The last option of improvement could be to use the MachineAgent and the MAS simulation to elaborate an optimal tool changing strategy for the magazine. Such a simulation with MAS could also be used to elaborate a suitable tool disposition strategy in advance.

List of Figures

List of Tables

Bibliography

[AIST10] Michael Andreev, Anton Ivaschenko, Petr Skobelev, and Alexander Tsarev. A multi-agent platform design for adaptive networks of intelligent production schedulers. IFAC Proceedings Volumes, 43(4):78–83, 2010.

[BC02] Piermarco Burrafato and Massimo Cossentino. Designing a multi-agent solution for a bookstore with the passi methodology. AOIS@ CAiSE, 57, 2002.

[BCG07] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. Developing multi-agent systems with JADE, volume 7. John Wiley and Sons, 2007.

[BD04] Adil Baykasoglu and Türkay Dereli. Heuristic optimization system for the determination of index positions on cnc magazines with the consideration of cutting tool duplications. International Journal of Production Research, 42(7):1281–1303, 2004.

[BHO+18] Darya Botkina, Mikael Hedlind, Bengt Olsson, Jannik Henser, and Thomas Lundholm. Digital twin of a cutting tool. Procedia CIRP, 72:215–218, 2018.

[BJW13] Stefan Bussmann, Nicolas R Jennings, and Michael Wooldridge. Multiagent systems for manufacturing control: a design methodology. Springer Science & Business Media, 2013.

[BNP+08] Alan Boyd, D. Noller, P. Peters, D. Salkeld, T. Thomasma, C. Gifford, S. Pike, and A. Smith. SOA in manufacturing—guidebook. MESA International, IBM Corporation and Capgemini Co-Branded White Paper, pages 24–29, 2008.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, 8(3):203–236, 2004.

[BPR99] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade a fipa compliant agent framework. In Proceedings of PAAM, number 97 in 3, page 33. London, The Practical Application Company, 1999.

[BS01] Stefan Bussmann and Klaus Schild. An agent-based approach to the control of flexible production systems. In ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 01TH8597), volume 2, pages 481–488. IEEE, 2001.

[CSC03] Massimo Cossentino, Luca Sabatucci, and Antonio Chella. Designing jade systems with the support of case tools and patterns. Special Issue on JADE of Telecom Italia Journal EXP of September, 2003.

[DBL09] Kanchan Das, Md Fazle Baki, and Xiangyong Li. Optimization of operation and changeover time for production planning and scheduling in a flexible manufacturing system. Computers & Industrial Engineering, 56(1):283–293, 2009.

[DD17] Deutschland and Statistisches Bundesamt Deutschland. Statistisches Jahrbuch Deutschland und Internationales. Statistisches Bundesamt (Destatis), October, 2017.

[Dee01] S Misbah Deen. Cooperating agents for holonic manufacturing. In ECCAI Advanced Course on Artificial Intelligence, pages 119–133. Springer, 2001.

[DeL99] Scott A DeLoach. Multiagent systems engineering: A methodology and language for designing agent systems. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH DEPT OF ELECTRICAL AND ..., 1999.

[DKS14] Berend Denkena, Max Krüger, and Justin Schmidt. Condition-based tool management for small batch production. The International Journal of Advanced Manufacturing Technology, 74(1-4):471–480, 2014.

[ElM09] Hoda A ElMaraghy. Changing and evolving products and systems–models and enablers. In Changeable and reconfigurable manufacturing systems, pages 25–45. Springer, 2009.

[EMC] EMCO. Maxxmill 750: Emco lathes and milling machines for cnc turning and milling. https://www.emco-world.com/en/products/industry/milling/cat/79/d/2/p/1000839%2C79/pr/maxxmill-750.html. (Accessed on 04/29/2019).

[fIPA03] Foundation for Intelligent Physical Agents. FIPA Specification SC00023, Agent management specification, 2003.

[FRU99] Marshall Fisher, Kamalini Ramdas, and Karl Ulrich. Component sharing in the management of product variety: A study of automotive braking systems. Management Science, 45(3):297–315, 1999.

[FTP19] Solmaz Mansour Fallah, Thomas Trautner, and Florian Pauker. Integrated tool lifecycle. Procedia CIRP, 79:257–262, 2019.

[Gei13] Thomas Geib. Geschäftsprozeßorientiertes Werkzeugmanagement. Springer-Verlag, 2013.

[GHS05] Paolo Giorgini and Brian Henderson-Sellers. Agent-oriented methodologies: an introduction. In Agent-Oriented Methodologies, pages 1–19. IGI Global, 2005.

[Gro13] Industrie 4.0 Working Group. Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group. acatech -Deutsche Akademie der Technikwissenschaften), 2013.

[GSS93] Ann E Gray, Abraham Seidmann, and Kathryn E Stecke. A synthesis of decision models for tool management in automated manufacturing. Management science, 39(5):549–567, 1993.

[HK16] Mior Uzair Mior Hassan and Aini Zuhra Abdul Kadir. Management and monitoring of rfid-based cutting tool information for integrated manufacturing. International Journal of Engineering and Technology (IJET), 2016.

[HLMS98] Alain Hertz, Gilbert Laporte, Michel Mittaz, and Kathryn E Stecke. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE transactions, 30(8):689–694, 1998.

[HSTD05] Brian Henderson-Sellers, QN Numi Tran, and John Debenham. An etymological and metamodel-based evaluation of the terms" goals and tasks" in agent-oriented methodologies. Journal of Object Technology, 2005.

[IGGV97] Carlos A Iglesias, Mercedes Garijo, José C González, and Juan R Velasco. Analysis and design of multiagent systems using mas-commonkads. In International Workshop on Agent Theories, Architectures, and Languages, pages 313–327. Springer, 1997.

[Jen00] Nicholas R Jennings. On agent-based software engineering. Artificial intelligence, 117(2):277–296, 2000.

[JMMLSF05] Sven Jacobi, Cristián Madrigal-Mora, Esteban León-Soto, and Klaus Fischer. Agentsteel: An agent-based online system for the planning and observation of steel production. In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pages 114–119. ACM, 2005.

[JS05] François Jammes and Harm Smit. Service-oriented paradigms in industrial automation. IEEE Transactions on industrial informatics, 1(1):62–70, 2005.

[Kin12] Steffen Kinkel. Trends in production relocation and backshoring activities: changing patterns in the course of the global economic crisis. International Journal of Operations & Production Management, 32(6):696–720, 2012.

[KM06] Jozef Konings and Alan Patrick Murphy. Do multinational enterprises relocate employment to low-wage regions? evidence from european multinationals. Review of World Economics, 142(2):267–286, 2006.

[KTW11] Sven O Krumke, Alfred Taudes, and Stephan Westphal. Online scheduling of weighted equal-length jobs with hard deadlines on parallel machines. Computers & Operations Research, 38(8):1103–1108, 2011.

[LBK15] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manufacturing Letters, 3:18–23, 2015.

[LCK16] Paulo Leitão, Armando Walter Colombo, and Stamatis Karnouskos. Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. Computers in Industry, 81:11–25, 2016.

[Lee08] Edward A Lee. Cyber physical systems: Design challenges. In 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), pages 363–369. IEEE, 2008.

[LKR+16] Paulo Leitao, Stamatis Karnouskos, Luis Ribeiro, Jay Lee, Thomas Strasser, and Armando W Colombo. Smart agents in industrial cyber-physical systems. Proceedings of the IEEE, 104(5):1086–1101, 2016.

[LL13] J Lee and E Lapira. Predictive factories: the next transformation. Manufacturing Leadership Journal, 20(1):13–24, 2013.

[Mon14] László Monostori. Cyber-physical production systems: Roots, expectations and r&d challenges. Procedia Cirp, 17:9–13, 2014.

[MSF96] John Paul MacDuffie, Kannan Sethuraman, and Marshall L Fisher. Product variety and manufacturing performance: evidence from the international automotive assembly plant study. Management Science, 42(3):350–369, 1996.

[MWH16] Guha Manogharan, Richard A Wysk, and Ola LA Harrysson. Additive manufacturing–integrated hybrid manufacturing and subtractive processes: economic model and analysis. International Journal of Computer Integrated Manufacturing, 29(5):473–488, 2016.

[Nyq08] Olof Nyqvist. Information management for cutting tools: information models and ontologies. PhD thesis, KTH, 2008.

[PM08] Michal Pěchouček and Vladimír Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. Autonomous agents and multi-agent systems, 17(3):397–431, 2008.

[PVB05] Michal Pechoucek, Jiri Vokrinek, and Petr Becvar. Explantech: multiagent support for manufacturing decision making. IEEE Intelligent Systems, 20(1):67–74, 2005.

[PW02] Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In International Workshop on Agent-Oriented Software Engineering, pages 174–185. Springer, 2002.

[RBM08] Luís Ribeiro, José Barata, and Pedro Mendes. MAS and SOA: complementary automation paradigms. In Innovation in manufacturing networks, pages 259–268. Springer, 2008.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William E. Lorensen, et al. Object-oriented modeling and design, volume 199. Prentice-hall Englewood Cliffs, NJ, 1991.

[RG+95] Anand S Rao, Michael P Georgeff, et al. Bdi agents: from theory to practice. In ICMAS, volume 95, pages 312–319, 1995.

[SAM17] Eva Schaupp, Eberhard Abele, and Joachim Metternich. Potentials of digitalization in tool management. Procedia CIRP, 63:144–149, 2017.

[SE89] Joseph Sharit and Sharad Elhence. Computerization of tool-replacement decision making in flexible manufacturing systems: a human-systems perspective. THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH, 27(12):2027–2039, 1989.

[SHYN06] Weiming Shen, Qi Hao, Hyun Joong Yoon, and Douglas H Norrie. Applications of agent-based systems in intelligent manufacturing: An updated review. Advanced engineering INFORMATICS, 20(4):415–431, 2006.

[SSS+13] V Shpilevoy, A Shishov, P Skobelev, E Kolbova, D Kazanskaia, Ya Shepilov, and A Tsarev. Multi-agent system smart factory for real-time workshop management in aircraft jet engines production. IFAC Proceedings Volumes, 46(7):204–209, 2013.

[Sta17] Wirtschaftskammer Österreich – Stabsabteilung Statistik. Statistisches Jahrbuch Österreich. Wirtschaftskammer Österreich, 2017.

[VMS+14] Pavel Vrba, Vladimír Mařík, Pierluigi Siano, Paulo Leitão, Gulnara Zhabelova, Valeriy Vyatkin, and Thomas Strasser. A review of agent and service-oriented concepts applied to intelligent energy systems. IEEE transactions on industrial informatics, 10(3):1890–1903, 2014.

[WIE] TU WIEN. Pilotfabrik der tu wien. http://pilotfabrik.tuwien.ac.at/en/. (Accessed on 04/29/2019).

[WJ95] Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. The knowledge engineering review, 10(2):115–152, 1995.

[WJK00] Michael Wooldridge, Nicholas R Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. Autonomous Agents and multi-agent systems, 3(3):285–312, 2000.

[WNY+09] Guoxin Wang, Hidehito Nakajima, Yan Yan, Xiang Zhang, and Lu Wang. A methodology of tool lifecycle management and control based on rfid. In Industrial Engineering and Engineering Management, 2009. IEEM 2009. IEEE International Conference on, pages 1920–1924. IEEE, 2009.

[ZJW03] Franco Zambonelli, Nicholas R Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM), 12(3):317–370, 2003.

[Zol18] Zoller. »venturion 450« premium presetting and measuring machine for tools of all kinds. https://www.zoller.info/us/products/presetting-measuring/vertical-devices/venturion-450.html?langswitch=1&cHash=1f5149df6a75386427ff55b33a639840, April 2018. (Accessed on 04/30/2019).