Agent-Oriented Programming of Manufacturing Control Tasks

Stefan Bussmann

Daimler-Benz AG
Research and Technology
Alt-Moabit 96a, D-10559 Berlin
email: Stefan.Bussmann@dbag.bln.daimlerbenz.com

Abstract

The success of agent-oriented concepts in various application domains, in particular in manufacturing control, creates the need for an agent-oriented analysis, design, and programming methodology. This paper presents a programming method that covers one step of the necessary methodology. Given a specification of the tasks to be performed, the method allows to program the corresponding agent in three steps: (i) programming of the individual tasks; (ii) synchronization of tasks to avoid concurrency problems; and (iii) specification of script execution on a single processor machine. The programming method was specifically designed for implementing manufacturing control agents and complies with the industrial requirements stated in this paper.

1 Introduction

Multi-agent systems has become the key information technology for the next generation of manufacturing control. Motivated by the inability of existing manufacturing systems (i) to deal with the evolution of products and (ii) to maintain a satisfying performance outside normal operation [13], manufacturing research and industry has proposed *holonic manufacturing systems (HMS).* This new manufacturing paradigm is supposed to overcome these deficits with the help of concepts like autonomy, cooperation, and self-similarity [14]. In a holonic manufacturing system, autonomous and self-reliant manufacturing units, called holons, cooperate in order to achieve the overall manufacturing goals. A system of holons is called a holarchy and may itself be a holon that acts as an autonomous and cooperative unit in another holarchy [7]. Both principles enable holonic manufacturing to flexibly organize and control the whole manufacturing process of a company.

Even though at first glance the ideas of HMS seem identical to those of multi-agent applications to manufacturing (cf. e.g. [4,10,12,17]), a thorough comparison reveals significant differences: HMS is an organizing principle for structuring and controlling manufacturing processes, whereas multi-agent systems is a software technology for realizing the information processing of a holonic manufacturing system [6]. That is, multi-agent systems is an enabling technology for holonic manufacturing. This thesis is supported by a comparison of the terms agent and holon: A holon contains the information processing and the physical processing part of a manufacturing entity, while an agent is some kind of software process. Even though the physical processing part is not mandatory for a holon, and sometimes the term agent is also used for physical entities, this characterization of agents and holons is fairly common.

The success of multi-agent concepts in the manufacturing domain creates new challenges for the technology. Apart from the (problem-solving) functionality, an industrially used manufacturing system must also satisfy properties such as reliability, fault-tolerance, maintainability, transparency, etc. But above all, for multi-agent technology to be widely used and accepted in industry, non-researchers must be enabled to apply agent-oriented techniques just as any other engineering method. This implies in particular that engineers are enabled to design and program an agent-oriented control system in a straight-forward and efficient manner. Thus, the success of multi-agent systems creates ultimately the need for an agent-oriented analysis, design, and programming methodology.

This paper makes a first step towards an agent-oriented methodology for manufacturing control of the shop floor. It presents a programming method that allows to program control agents once a specification of each agent has been derived in a previous design step. The programming method maps the control agent onto a computational architecture whose implementation is transparent to the programmer.

The method presented in this paper is the result of two research projects at Daimler-Benz in which agent-oriented techniques were applied to car manufacturing. The development of the control systems in cooperation with the plants revealed the need for a methodology that can be applied by engineers without a research background in agent technology.

The paper is organized as follows. The next section discusses general industrial requirements on the development of agent-oriented systems for manufacturing control. Section 3 presents the programming method for control agents in three steps: (i) individual programming of control tasks, (ii) synchronization of tasks to avoid concurrency problems, and (iii) specification of script execution on a single-processor machine. The programming method is evaluated with respect to the industrial requirements. Finally, previous work related to the aim of this paper is discussed before the paper is concluded.

2 Industrial Requirements

A multi-agent manufacturing control system usually requires the use of special reasoning and coordination techniques. Depending on the manufacturing goals and the type of manufacturing process, different kinds of control architectures and/or strategies might be necessary in order to optimally control the manufacturing process (cf. e.g. [1,2,4]). However, despite the special needs of a particular manufacturing application, any industrial control system must meet general requirements. These requirements cover functional and software-engineering aspects of the control system.

2.1 Functional Requirements

Manufacturing control systems are large, complex artefacts which are designed to perform a clearly-defined task in a well-structured, standardized environment. Even though manufacturing processes experience a lot of changes and disturbances, the degree of uncertainty and unpredictably is not comparable to that of space, traffic, or service applications. As a consequence, manufacturing applications require less mental and social deliberation than typical applications of multi-agent systems. This is particularly true for mental categories such as desires, intentions, or joint intentions, including their associated reasoning. Their use is possible, but not appropriate for most manufacturing control tasks.

Moreover, all manufacturing agents cooperate in order to achieve the overall manufacturing goals. With respect to these goals, an agent never deliberatively rejects the cooperation with another agent. Only when the requested actions are impossible or strongly disadvantageous to the manufacturing process, it refuses their execution. In this sense, the manufacturing agents are semi-autonomous. This leads us to the first requirement.

> Requirement I: Manufacturing control systems require semi-autonomous agents. The agents must reason about the behavior of the manufacturing system, but not about their own mental attitudes or that of other control units.

This requirement does not deny the creation of mental categories inside an agent, like intentions. It only states that the agent does not have to reason about them. Furthermore, this requirement may be invalid for virtual enterprises which are temporary joint ventures of autonomous companies.

The second requirement is concerned with the type of behavior a control unit must exhibit. Manufacturing control units are continuously faced with a high rate of repeated events that are known, but unpredictable. This flow of events must be handled timely and efficiently. The handling of the events can consequently be fixed beforehand with the help of routines, while only the initiation and execution of routines must be performed on-line. The set of events and their pattern of occurrence changes only slowly over time. These long-term changes are mainly caused by major product and production technology changes.

> Requirement II: Manufacturing control units mostly require a routine-based behavior that is both timely and efficient. This behavior should be either configurable or self-adaptive.

This requirement does not ban explicit reasoning such as planning or scheduling from manufacturing control, but emphasizes that for most control units routine-based behavior is sufficient (cf. [1,2] for examples of explicit reasoning included in autonomous and cooperative manufacturing systems). Self-adaptiveness, on the other hand, may require explicit reasoning about the behavior of an agent. This reasoning, however, is not concerned with the agent's reaction to a specific event, but supervises the behavior over time.

2.2 Software-Engineering Requirements

Apart from the functional requirements, any control system that is supposed to be used in a productive manufacturing environment must meet general industrial standards. These standards specify, among others, requirements for reliability, fault-tolerance, diagnosibility, and maintainability. In particular, control systems must reach a degree of reliability that guarantees continuous operation. This is equally true for the control software. Product reliability, however, is only achievable if the software development process is performed in an engineering-like manner, instead of an ad hoc fashion.

Moreover, as already argued in the introduction, the widespread application of agent-oriented techniques to industrial control requires that the software development process is supported by a methodology. This methodology should enable skilled engineers to develop the agent-oriented control system in a straight-forward and efficient manner.

A method for programming an agent should therefore meet at least the following minimal requirements:

Requirement III: Programming methods must provide encapsulation of data and procedures.

Requirement IV: Control programs must have a clear semantics. Additionally, the behavior of an agent should be completely specified by its control program.

Requirement V: A programming method or methodology should lead straight-forward from the control task to the agent program.

Agent-oriented programming methods must fulfill at least the above requirements in order to be applicable to industrial control problems.

3 Agent Programming

Manufacturing applications require a design methodology that derives the architecture and algorithms of the control system in a top-down approach from the overall manufacturing goals. First of all, during an analysis of the overall goals and the process characteristics, the methodology identifies global control strategies which optimally run the manufacturing process. In a second step, the control strategies are decomposed into single control tasks which can be executed locally with the partial knowledge of the shop floor. These control tasks are then grouped and assigned to the agent of a manufacturing unit. In this paper, we assume to have accomplished these steps and are now faced with the task of designing and implementing each individual agent as a control component. (Methodologies for analyzing and designing multi-agent systems and deriving a specification for the agents of the system have been proposed in [4,8,15,17].)

Given the control tasks one agent has to perform, an executable program is designed for each task. Because of the inherent concurrency of control strategies, the execution of tasks is synchronized in order to avoid data and action conflicts. Finally, the concurrent execution of tasks is mapped onto a single-processor machine. The result of these three steps is the operational specification of a control agent which solves the control tasks specified.

The following three subsections outline this agent-oriented approach to the programming of manufacturing control units. Starting from the individual tasks of an agent, they show how the corresponding scripts are programmed (subsection 3.1), synchronized (subsection 3.2), and assembled into a single-processor program (subsection 3.3). The final subsection then evaluates the programming approach with respect to the industrial requirements stated in section 2.

3.1 Task-Oriented Programming

Control tasks are implemented with the help of scripts. Scripts are procedures with parameters, local variables, and a list of commands, just as in most imperative or object-oriented programming languages. Local variables contain common data structures (number, strings, lists, etc.) and commands allow conditional execution (if-then-else), loops (for/while), and the invocation of other scripts.

Scripts and global variables are grouped into modules. Each module provides a certain functionality, like e.g. sensor interpretation, protocol management, or scheduling. A set of modules finally constitutes an agent. A typical functional agent architecture for manufacturing control is depicted in figure 1.

Modules in turn are organized in a module hierarchy which defines a non-reflexive, transitive subsumption relation between modules. An example hierarchy is given in figure 2. A script can call (i) any script of the same module, (ii) any script declared as *restricted* in super-ordinate modules, and (iii) any script declared as *public*. An analogous rule applies to the use of global variables.

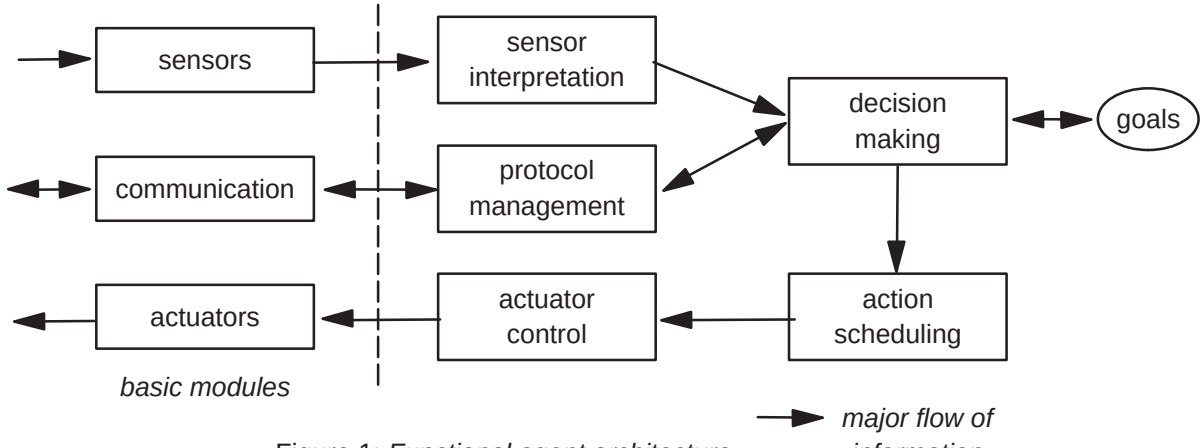

Figure 1: *Functional agent architecture.*

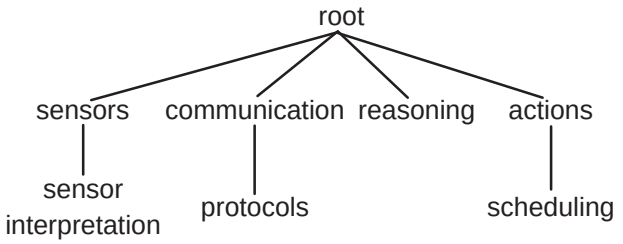

Figure 2: *An example module hierarchy.*

Scripts are invoked by events. If an event occurs, it is mapped to a script and the arguments of the event are passed as parameters to the script. The mapping is defined for each module and events are created with respect to a module. The events to be mapped can be external (coming from basic sensor or communication facilities) or internal (created by scripts). The activation of scripts (within the agent) is usually initiated by external events. The invoked scripts then either call other scripts or create internal events (cf. fig. 3). Exceptions are only agent initialization and timing events which both may start activation.

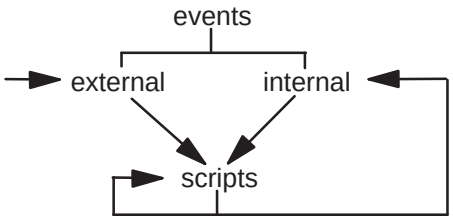

Figure 3: *Script invocation.*

3.2 Concurrency of Tasks

Control applications always require concurrent task execution. Drilling machines for example load and unload work pieces in parallel in order to maximize throughput. Concurrent execution, however, may create undesirable effects that either result in inconsistent data or critical control behavior. In general, conflicts between tasks may arise on two levels: (i) concerning data, and (ii) between scripts.

Data conflicts occur whenever multiple scripts read and/or write data variables concurrently. These problems are known from distributed systems and can be solved with standard techniques, such as critical section commands or semaphores.

Script conflicts are caused by task dependencies. Two types of dependencies are possible: (i) a set of tasks has to be executed sequentially, or (ii) a set of tasks is incompatible and at least one script has to be aborted. An example of the former are scripts that handle entering and leaving work pieces. Scripts initiated by the operator that override current actions are examples of the latter.

Task dependencies must be declared by the programmer on the script level, i.e., scripts get into conflict either always or never. At run time, script conflicts may then be avoided if the set of active scripts is kept free of conflicting scripts. Since the empty set of active scripts is always conflict-free, it is sufficient to make the set conflict-free after a new script has become active. The sequential ordering of scripts, respectively the choice of the script to be aborted may be declared a priori or computed at run time.

In summary, data conflicts are avoided through declaring mutual exclusion within script bodies, whereas script conflicts are avoided by specifying script relations.

3.3 Computational Model

The final step of the agent programming is to map the concurrent execution of tasks onto a single-processor machine. For this, it is necessary to bring the tasks into a (possibly interleaved) computational sequence, i.e., scripts must be scheduled. Moreover, the scheduling has to be applied across all modules because the behavior of an agent is determined by its handling of events within several modules. For instance, after its creation a sensor signal is first interpreted and classified by a script in the sensor interpretation module (cf. fig. 1). This script creates a corresponding event that triggers a script in the reasoning module. Depending on its decision, the event is either handled as a time-critical or a normal task in the two action modules (or it may even be dropped).

Consequently, the ordering of scripts has to be defined across all modules which is counter-intuitive to the modular programming of tasks. In order to resolve this conflict, we introduce a second view in addition to the modular view of task programming: event prioritization. Events are assigned a priority which is passed along with event creation and script invocation (cf. fig. 4). External events who initiate script activation are assigned a pre-defined priority. The priority of internal events is determined dynamically by the script creating the event, while subscripts inherit the priority of the calling script.

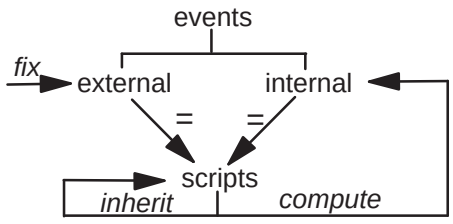

Figure 4: *Priority assignment.*

An example of a priority scheme is the following list of classes (listed with decreasing priority):

time-critical: Events that must be handled within a certain time-frame. The fast handling of these event guarantees the reactiveness of the agent.

task-oriented: Events that accomplish the normal operation of the agent (usually the actual control tasks) and that are not time-critical.

optional: Events that provide additional functionalities like monitoring or diagnosis.

The execution of scripts is then performed according to the following rules:

1. Classes are executed strictly in sequential order: First all scripts of class time-critical, then all scripts of class task-oriented, and finally all optional scripts are executed.

2. Between modules scheduling is done with a fair strategy (e.g., through round-robin).

3. Within a module the scheduling can be according to any appropriate strategy (round-robin, FIFO, LIFO, etc.).

It is possible to add more classes to the priority scheme. In some applications, for example, it is appropriate to distinguish between time-critical events that endanger the safety of the system and time-critical events that decrease the system's performance.

The computational architecture described above is much simpler than the functional architecture of an agent (cf. fig. 1). The computational architecture still consists of the basic modules, but in addition only requires an event control and a script execution module, as depicted in figure 5.

All events (internal or external) are sent to the event control. According to the rules described above it decides which script is to be executed next and hands this script to the script execution. This module either returns control to the event control because a script is finished or because it creates an internal event, or it is interrupted by an external event.

With the computational architecture we have described how task programs can be mapped onto a single-processor machine and have thus completed the specification of a control agent. The resulting specification can now be executed on a comparatively simpler architecture.

3.4 Evaluation

The programming method presented in this section employs concepts and techniques from modular programming and distributed systems. These techniques are combined in a specific approach to agent-oriented programming of manufacturing control tasks in order to meet the industrial requirements on the functionality of the control system and the software development process.

First, the programming approach meets the functional requirements stated in subsection 2.1. A typical functional agent architecture for manufacturing control requires modules for physical control (sensing and acting) and for agent interaction, but does not explicitly model the agent's mental attitudes or the mental attitudes of other agents (requirement I). The decision making of the agent may evaluate its own behavior or requests for cooperation with respect to its own goals, but it does so without considering the attitudes of other agents.

This conception of an agent architecture is supported by script programming approach. In particular, scripts allow to efficiently implement routine-based behavior that is both efficient and timely (requirement II). Configurability or self-adaptiveness, on the other hand, are not explicitly supported by the approach. To provide such functionality, the approach has to be extended in the future.

Second, the programming method also meets the software-engineering requirements stated in subsection 2.2. Scripts and variables are encapsulated in modules and have

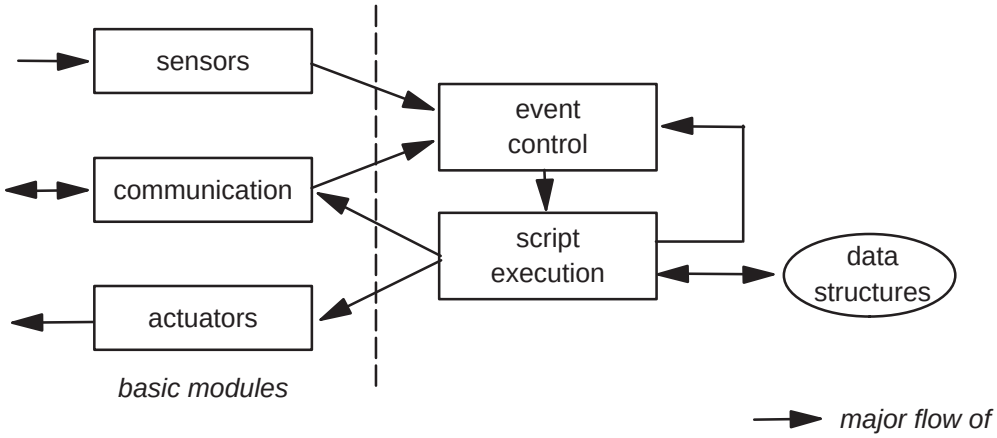

Figure 5: *Computational architecture.*

only limited access to variables and scripts in other modules (requirement III).

The programming method also provides a clear and complete semantics (requirement IV). The access of other variables and scripts is explicitly defined in the module hierarchy. The flow of control as well as the sequence of execution (of tasks) is specified by the script invocation associated with the priority assignment and the execution rules of the computational model. Finally, the declaration of data and tasks dependencies enables the programmer to avoid conflicts of tasks at run-time. All these aspects are explicitly specified and thus provide a clear and complete semantics of the agent's behavior.

Last but not least, the programming method proposed guides the programmer through the design process for an agent (requirement V). It starts with the tasks to be performed and derives scripts and modules for their execution. The scripts are then synchronized in order to avoid run-time conflicts. And finally, the execution of scripts is explicitly sequentialized on a single processor.

4 Related Work

So far, only few work has been concerned with agent-oriented analysis, design, or programming.

Burmeister [4] reviews object-oriented methodologies for analysis and design, and concludes that they are not adequate for the development of agent-oriented systems. First of all, the internal architecture as well as the interaction of agents is more structured than that of objects. And second, agents may decide autonomously whether or not to execute a request by another agent. Burmeister therefore proposes a methodology for analyzing and designing an agent system. This methodology describes how to identify the agents and their relationships, but does not explain how to program each agent.

Collinot et al. [8] proposed a methodology for specifying collective behaviors, like soccer playing. Elementary behaviors of agents are coordinated and organized in order to create the expected group behavior. This contrasts the approach of this paper that derives individual behaviors from the overall strategies. Furthermore, the methodology does not describe how the individual behaviors are implemented and how undesirable conflicts between elementary behaviors are avoided (only positive influences are considered).

Kinny et al. [15] proposed a methodology for designing BDI agents. The methodology allows to define the possible beliefs and plans of an agent. The plans are then executed on a BDI architecture, like the procedural reasoning system [11]. The methodology proposed by Kinny et al., however, does not support modularization. Nor does it explicitly specify in a transparent form how the plans are sequentialized on a single processor. For complex agents, the resulting behavior of the agent is therefore difficult to predict.

Braziet et al. [3] have developed a formal specification framework for complex reasoning systems that was applied to network management. The framework allows to specify primitive and complex tasks of a component and to define the interaction along information links. Task representation, however, is knowledge-based (cf. e.g. [9]) and is thus not appropriate for the domain considered in this paper where the overall manufacturing task is decomposed into operational strategies and routines.

A framework for agent-oriented programming was first proposed by Shoham [19]. In his approach, the state of agents is represented with the help of mental categories, such as beliefs, obligations, and capabilities. The states are changed by communication, actions, and commitment rules. Thomas [20] extended this framework by planning abilities that support the mental reasoning of an agent. For manufacturing control, however, both approaches are not appropriate because mental categories are inadequate primitives for expressing most control tasks (cf. section 2). AgentSpeak, on the other hand, is plan-oriented and thus more suitable for programming tasks [21]. However, the language does not provide mechanisms for modular agent design or execution control (except for plan priorities).

A lot of work has been done on agent architectures (see [16,22] for an overview). Even though many of these architectures include routine-based capabilities, hardly any architecture supports modular programming and execution control as described in this paper.

5 Conclusion

This paper presented a programming method for designing manufacturing control agents. Starting from a specification of the control tasks an agent has to perform, a program that exhibits the requested control behavior is derived in three steps: (i) for each task, a set of scripts is implemented that accomplishes the task, (ii) conflicting scripts are declared, and (iii) single-thread script execution is specified through event prioritization.

The programming method presented fulfills the industrial requirements for manufacturing control. First, semi-autonomy and routine-based behavior provide the appropriate agent model. Second, the programming method meets the software-engineering requirement. Programming is modular, extensible, and transparent to the programmer.

The programming method thus accomplishes one step in the development of an agent-oriented manufacturing control system. First, the overall control task is analyzed and a vision of the required control behavior is developed. The overall control behavior is then decomposed into strategies and individual control tasks. These control tasks are finally imple-

mented with the help of the programming method presented. Future work will be to extend the method to a framework that includes all analysis, design, and programming steps necessary to develop manufacturing control systems. Eventually, the resulting methodology will also include concepts for testing and diagnosing such systems.

References

[1] H. Baumgärtel, S. Bussmann, M. Klosterberg: "Multi-Agent Coordination of Material Flow in a Car Plant", Proc. of the 2nd Int. Conf. on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM'97) London, UK, 1997, pp. 227 – 236.

[2] L. Bongaerts, P. Valckenaers, H. Van Brussel, J. Wyns: "Schedule Execution for a Holonic Shop Floor Control System", in: Advanced Summer Institute (ASI'95) on Life Cycle Approaches to Production Systems, Lisbon, Portugal, 1995.

[3] F. Brazier, B. Dunin-Keplicz, N.R. Jennings, J. Treur: "Formal Specification of Multi-Agent Systems: a Real-World Case", Proc. of Int. Conf. on Multi-Agent Systems (ICMAS'95), AAAI Press/MIT Press, Menlo Park, USA, 1995, pp. 25 – 32.

[4] B. Burmeister: "Models and Methodology for Agent-Oriented Analysis and Design", in: K. Fischer (ed.), Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, DFKI Document D-96-06, Saarbrücken, Germany, 1996.

[5] S. Bussmann: "A Multi-Agent Approach to Dynamic, Adaptive Scheduling of Material Flow", in: J.W. Perram, J.-P. Müller (eds.), Distributed Software Agents and Applications (MAAMAW'94), LNAI 1069, Springer Verlag, Berlin, 1996.

[6] S. Bussmann: "An Agent-Oriented Architecture for Holonic Manufacturing Control", in: Proc. of the 1st Int. Workshop on Intelligent Manufacturing Systems, EPFL, Lausanne, Switzerland, 1998.

[7] J. Christensen: "Holonic Manufacturing Systems: Initial Architecture and Standards Directions", in: [14].

[8] A. Collinot, A. Drogoul, P. Benhamou: "Agent-Oriented Design of a Soccer Robot Team", Proc. of Int. Conf. on Multi-Agent Systems, AAAI Press, Menlo Park, USA, 1996, pp. 41 – 47.

[9] B. Dunin-Keplicz, J. Treur: "Compositional Formal Specification of Multi-Agent Systems", Pre-Proc. of Workshop on Agent Theories, Architectures, and Languages (ATAL-94), Amsterdam, The Netherlands, 1994, pp. 88 – 109.

[10] K. Fischer: "The Design of an Intelligent Manufacturing System", in: S.M. Deen (ed.), Proc. of the 2nd Int. Working Conf. on Cooperating Knowledge-Based Systems, University of Keele, Dake Centre, 1994.

[11] M.P. Georgeff, FF. Ingrand: "Decision-making in an embedded reasoning system", in: Proc. of the 11th Int. Joint Conf. on AI (IJCAI'89), Detroit, USA, 1989, pp. 972 – 978.

[12] S. Hahndel, F. Fuchs, P. Levi: "Distributed Negotiation-Based Task Planning for a flexible Manufacturing Environment", in: J.W. Perram, J.-P. Müller (eds.), Distributed Software Agents and Applications, LNAI 1069, Springer Verlag, Berlin, 1996.

[13] HMS Press Release, Revision 3.6, HMS Server, `http://hms.ifw.uni-hannover.de/`.

[14] First European Conference on Holonic Manufacturing Systems, European HMS Consortium, Hannover, Germany, 1994.

[15] D. Kinny, M. Georgeff, A. Rao: "A Methodology and Modelling Technique for Systems of BDI Agents", in: W. Van de Velde, J.W. Perram (eds.), Agents Breaking Away (MAAMAW'96), LNAI 1038, Springer Verlag, Berlin, 1996.

[16] J.P. Müller: "The Design of Intelligent Agents", LNAI 1177, Springer-Verlag, Berlin, Germany, 1996.

[17] V. Parunak, J. Sauter, S. Clark: "Toward the Specification and Design of Industrial Synthetic Ecosystems", in: M. Singh, A. Rao, M. Wooldridge (eds.), Intelligent Agents IV (ATAL'97), LNAI 1365, Springer-Verlag, Berlin, Germany, 1998.

[18] H.V.D. Parunak, J.F. White, P.W. Lozo, R. Judd, B.W. Irish, J. Kindrick: "An Architecture for Heuristic Factory Control", Proc. of the American Control Conference, Seattle, 1986.

[19] Y. Shoham: "Agent–oriented programming", Artificial Intelligence, 60, 1993, pp. 51 – 92.

[20] S.R. Thomas: "The PLACA Agent Programming Language", Pre-Proc. of Workshop on Agent Theories, Architectures, and Languages (ATAL'94), Amsterdam, The Netherlands, 1994, pp. 307 – 319.

[21] D. Weerasooriya, A. Rao, K. Ramamohanarao: "Design of a Concurrent Agent-Oriented Language", Pre-Proc. of Workshop on Agent Theories, Architectures, and Languages (ATAL'94), Amsterdam, The Netherlands, 1994, pp. 334 – 343.

[22] M. Wooldridge, N.R. Jennings: "Intelligent Agents:Theory and Practice", Knowledge Engineering Review, 10(2), 1995, pp.115 –152.