A Multi-Agent Approach to Dynamic, Adaptive Scheduling of Material Flow

Stefan Bussmann
Daimler-Benz AG
Research and Technology
Alt-Moabit 91b, D–10559 Berlin
email: bussmann@DBresearch–berlin.de

Abstract

Advanced manufacturing control still remains an important topic in current research. Especially aspects of dynamics and of failures in the production process are insufficiently taken into account by systems in use. This paper presents a multi-agent approach to scheduling material flow that shows dynamic and adaptive behaviour. Even though machine scheduling has found a thorough treatment in AI literature, there are only few investigations on the material flow problem. In this paper, it is argued that a decentralized architecture with centralized control fits well with the local and global aspects of the scheduling problem. The top-level algorithms of the scheduling process are outlined and further improvements required are sketched out.

1 Introduction

Advanced manufacturing control has long been an important topic in Artificial Intelligence research and has recently attracted interest of researchers in the field of Multi-Agent Systems because practical experience with existing approaches remains unsatisfactory. Classical scheduling methods typically aim at optimising the solution with respect to a global goal function and assume a static environment, i.e. all necessary information is known beforehand. Real manufacturing control, however, is much more dynamic due to the continual arrival of orders and various perturbations of the manufacturing process, such as breakdowns. This dynamic character of most manufacturing problems renders the long-term optimisation of schedules pointless and rather demands a more adaptive scheduling strategy.

This paper presents a dynamic and adaptive approach to material flow scheduling that is based on multi-agent techniques. The material flow problem has been scarcely treated in the multi-agent literature. Known to us is only the work in [PLJ+86] that implements a Kanban-like strategy. Concerning dynamic scheduling, some work has been done (cf. [BO92], [OSH88], or [DK93]) to tackle machine scheduling using contract net protocols [Smi80]. The basic idea is to use bargaining between autonomous agents in order to find an optimal order assignment, much in the spirit of supply and demand in economics. In all cases the principle is as follows: a broker announces an order to be scheduled; the transporting units make bids; and the broker awards the order to the best bidder. However, for global problems such as scheduling in the manufacturing domain (which, in general, is NP-hard) the idea of bargaining is inappropriate. To arrive at an optimal solution to the overall (global) assignment process, it may be necessary to assign orders to a transporter even though the assignment is locally suboptimal. Therefore, the process of bidding, which is basically a local decision making, cannot guarantee

published in Pre-Proceedings, MAAMAW-94, Odense, Denmark, 1994

a globally optimal solution. As we will argue in later sections, it is necessary to introduce more centralized control.

Our design of the scheduling process is along the principle of ”as decentralized as possible, as centralized as necessary”. During the discussion of the material flow problem, we will outline which decisions can be made locally and which must be taken with a global perspective. Following these considerations, we will present an architecture that distributes all local computations to the transport units and gives the responsibility for the global aspects to a coordinator. Additionally, we will analyse how the multi-agent techniques contributed to our solution of the material flow problem.

The remainder of the paper is organised as follows: the next section defines the scenario and the problem to be solved; section 3 highlights important aspects of the problem; section 4 describes the overall architecture and the main algorithm for the scheduling process; section 5 gives some more details on the implementation; and the last section evaluates the multi-agent approach to the material flow problem and outlines aspects that could be improved upon in future work.

2 Problem Definition

This section presents a *model of material flow* in a factory and defines the *transportation problem* to be solved.

Basically, the factory consists of a set of machines and a set of transporters. Machine orders specify *which machine* has to produce *which product* out of *which material* in *which time interval*. Implicitly, these orders define a material flow by listing produced material as consumed material in a later order. For transportation, the factory is equipped with transport units, such as forklifts or conveyer belts. The basic task of the material flow system is to compute a schedule for the transport units such that every piece of material reaches its next destination in time.

In the context of this paper, we assume machine orders to be given from outside. They are inserted *dynamically* into the scenario while previous machine orders are in execution. Furthermore, the execution of the schedule suffers from *perturbations*. On one hand, orders may change w.r.t. their specification after being introduced to the system (for instance, production steps are delayed or become more urgent); on the other hand, failures, such as sudden breakdowns of transporters, may occur. Due to these imposed characteristics of the manufacturing environment, the material flow problem is *dynamic* and *perturbed*.

2.1 The Model of Material Flow

We will now describe the model of material flow. It consists of a set **S** of storage spaces S_1, .., S_m, a set **M** of machines M_1, .., M_n, a set **T** of transporters T_1, .., T_q, a set **N** of material, an environment **E**, and a time model $\mathcal{T}$. We assume the environment to contain a (planar) graph in which machines and storage spaces are placed at the vertices, whereas transporters can freely move along the arcs of the graph. Figure 1 gives an example of such a graph for a small factory.

Raw material flows through the production process and is transformed into intermediate or final products. For reasons of simplicity, we will refer to raw material, intermediate and

final products simply as *material*. An instance of the set **N** of materials is of the form *(id, volume)* where id is the identification number of the specific material and volume is a positive, non-zero real number. Material can either be stored in a storage space, at a machine site, or on a transporter, all having limited storage capacity. This is expressed by the following functions ($\mathcal{P}$(**N**) denotes the power set of **N**):

capacity: $\mathbf{S} \cup \mathbf{M} \cup \mathbf{T} \rightarrow \mathbb{R}_+$
content: $(\mathbf{S} \cup \mathbf{M} \cup \mathbf{T}) \times \mathcal{T} \rightarrow \mathcal{P}(\mathbf{N})$

The content of a storage space, of a machine storage space, or of the loading space of a transporter may never exceed the storage capacity, i.e. the following global constraint must hold:

$$\forall\, t \in \mathcal{T},\ M_i \in (\mathbf{S} \cup \mathbf{M} \cup \mathbf{T}):\ \sum_{\forall n \in \text{ content } (Mi,t)} \text{volume}(n) \leq \text{capacity}(M_i) \qquad (2.1)$$

Any storage attempts that would exceed the storage capacity are declined.

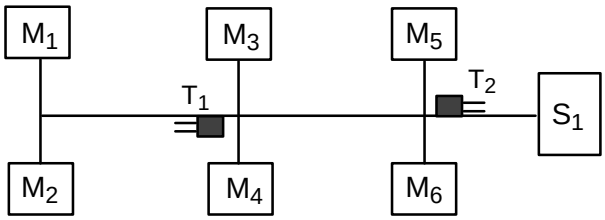

Figure 1: Factory Example

Machine Orders and Their Generation

A **machine order** specifies that a machine M_i or a stock room S_i is supposed to produce the material $n_p \in \mathbf{N}$ out of the material $n_c \subset \mathbf{N}$ beginning from the starting time t_s up to the finish time t_f ($t_s < t_f$). In the case of a storage space, it either consumes or produces material in order to model the material import and export of the factory.

A machine order **is executed** iff $n_c \subset \text{content}(M_i, t_s)$. Then it is true that $n_p \in \text{content}(M_i, t_f)$. This definition implies that all consumed material must reach the machine in time, otherwise the order could not be executed and the product would not be produced.

Machine orders are entered dynamically into the scenario, i.e. there exists a sequence $(t_i)_{i \in \mathbb{N}}$ in $\mathcal{T}$, such that at time t_i a set MO_i of machine orders is generated. The generation is random from the point of view of the model (it may be deterministic for an outsider). It may also alter already announced orders by introducing new versions of previously generated orders (which models changes in the specification of orders). A set MO_i obeys a global consistency constraint which states that every product is consumed by a machine order scheduled later on:

$$\forall\, i \in \mathbb{N},\ o_2 \in MO_i:\ n_c \in \text{consumed}(o_2) \Rightarrow \exists\, o_1 \in MO_i:\ n_c = \text{product}(o_1) \wedge \text{finish-time}(o_1) < \text{start-time}(o_2)$$

This constraint enables us to derive a set of **transport orders** for every material that is consumed, i.e. an order to transport *material* n_c from the site of *machine*(o_1) to the site of *machine*(o_2) between *finish-time*(o_1) and *start-time*(o_2). Consequently, for each element t_i of the sequence $(t_i)_{i \in \mathbb{N}}$ there is a set of transport orders TO_i which describes the material flow for the set of machine orders MO_i. A formal definition of a transport order is omitted, since it can be derived straight forward from what has been said above.

Transporter Actions

In order to achieve the material flow defined by the transport orders, the transporters execute a sequence of actions consisting of *move*, *load*, and *unload*. A *move* defines the movement of the transporter, beginning at a specified start time, from one position on the routing graph to a second. Any acceleration or deceleration phases are abstracted and a constant velocity is assumed (possibly different for each transporter). A *load* action describes that the transporter transfers the material from a machine or a storage space into its loading space. An *unload* is defined correspondingly. Of course, constraints concerning the consistency of the action sequence, the capacity limits of a stock, the existence of material for loads or unloads etc. apply. They are omitted here to avoid deviation from the main topic discussed.

2.2 The Material Flow Problem(s)

The basic task is to compute a transportation schedule which assigns an action sequence to every transporter such that every piece of material reaches its destination in time. This task is called the **fulfillment problem**.

Since the machine schedule is dynamic, the scheduling of transport orders must be done incrementally. At every t_i of the generation sequence (t_i), new orders must be incorporated into the existing schedule. Additionally, already existing schedules must be adapted because every generation step may change previously announced orders. Thus, the scheduling mechanism must be *dynamic* and *adaptive*.

For a specific generation sequence, it may be impossible to find a correct transportation schedule at each step such that *all* orders are executed in time. This may happen, for example, because the time to schedule an order is too short or because concurrent orders exceed the total transport capacity. In such cases, we want the system to fill as many orders as possible, i.e. the percentage of transport orders that are executed should be maximized.

An extension of the basic transportation problem includes the possibility of transporter breakdowns. Whenever a transporter breaks down, all orders assigned to it must be rescheduled. Furthermore, any material stored in a transporter that broke down must be picked up immediately and moved to its destination. The possibility of breakdowns stresses the need for adaptiveness in the scheduling process.

This problem extension is called the **failure-including fulfillment problem**. Formally, the problem is defined by a sequence in $\mathcal{T}$ such that at each time step certain transporters break down. The system receives the information about the breakdown immediately.[1]

3 Analysis of the Problem

This section discusses some aspects of the scenario that will be referred to in later sections. This concerns especially the computations necessary to schedule a transport order and the relation of a transportation schedule to the storage schedule at the machines.

1. More realistic would be a delayed notice.

3.1 Evaluation and Scheduling of a Transport Order

The execution of a transport order is depicted in figure 2. In this figure the actions of the transporter (shown on the upper axis) are related to the time constraints of the corresponding transport order (lower axis). In order to carry out an order, the transporter first drives to the source (**arrival phase**), loads the material (**loading phase**), moves to the destination (**transport phase**), and unloads the material (**unloading phase**). The last three phases must fall into the interval between start and finish time of the transport order. Furthermore, whenever the transporter picks up the material later than the start time, then it must be temporarily stored at the source. This interval is called **lay time**. Analogously, there is a lay time at the destination.

An important aspect of the transport execution is that the **active transport** (loading, transport, and unloading phase) has a fixed time length, given a specific transport order and a specific transporter. The time length of the active transport does not depend on the start of the execution, whereas the length of the arrival phase changes whenever the transporter's position at the beginning of the execution changes. This position is normally the destination of the last executed order. Consequently, a permutation of orders in the schedule changes the time it takes to arrive at the next source.

Many computations for scheduling orders are **local** to a transporter. First of all, the computation of the arrival phase is different for every permutation of orders. Second, every transporter has a different schedule for its already assigned orders, and because of that the arrival phase for a new order is different for every transporter (even if scheduled at the same time). And third, if the scenario consists of heterogeneous transporters, i.e. transporters have different characteristics regarding average velocity, transport capacity, or even possible routes, then also the active transport has different lengths for every transporter. Due to these reasons, the question of whether and how an order can be incorporated into the schedule of a transporter is a computation that involves information only concerning the specific transporter, i.e. it is *"local" to the transporter*.

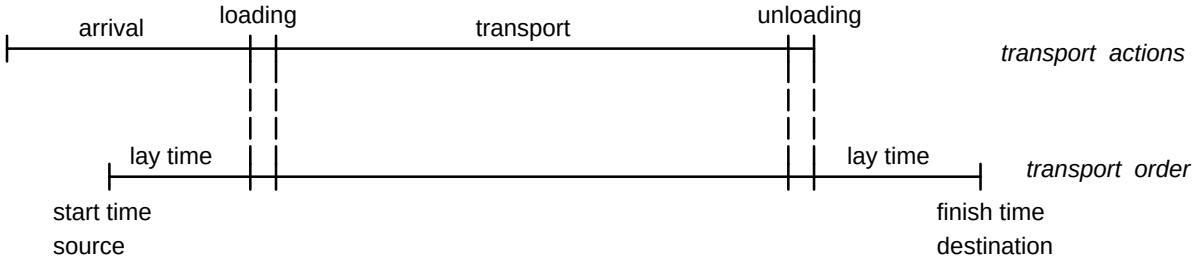

Figure 2: Transport Execution

A second aspect of the transport execution is its dependence on the *storage profiles* of the machines. Even though the active transport may be scheduled any time between the start and finish time of the transport order, different loading and unloading time points cause different lay times. This affects the transportation schedule because machines have limited storage capacity (cf. (2.1)). For instance, a machine runs out of storage space if a transporter unloads material for a machine order o_2 before the machine order o_1 has been started (o_1 is executed on the machine before o_2) and the total consumed material of o_1 and o_2 exceeds the storage capacity. As a consequence, the unloading of material for order o_2 is denied. These situations cause additional perturbations.

Furthermore, the dependency between the transportation schedule and the storage schedule is strong because any changes in the transportation schedule alter the storage profile on at least two machines. This effect also arises if the execution time of a transport order is changed only slightly.

3.2 Multiple Trips and Combination of Orders

A transporter is unable to execute an order if the volume of the material exceeds its loading space. If this is the case for all transporters, then the material has to be divided into packages and each package must be transported separately. The new set of transport orders is in most cases assigned to several transporters because a single transporter has to shuttle between source and destination and therefore would travel a longer distance. Such multiple trips increase the size of the assignment problem because the number of transport orders increases.

On the other hand, a transporter may have enough free loading space to take along additional material while executing an order. Figure 3 gives an example how to combine two orders by first picking up material at *source1*, then at *source2*, and finally dropping off the material at their destinations. Formally, the combination of transport orders is done by computing an action sequence that combines (and adopts) the action sequences necessary to carry out each single order. In contrast to multiple trips, the decision of whether to combine orders is local to a given transporter.

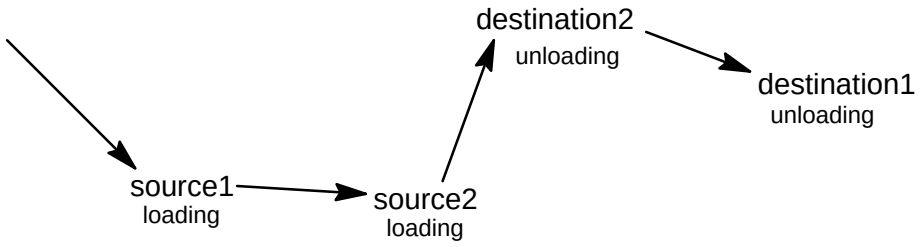

Figure 3: Combination of Orders

4 An Architecture for Material Flow

Centralized approaches to scheduling problems suffer from the combinatorical explosion of the search space. Already fairly simple scheduling problems may be NP-hard. For real world scheduling applications, the search space tends to be even larger because they include more choices and dependencies (cf. section 3.1 on the relation between the transportation and the storage schedule). Multi-agent systems offer two advantages here: (i) as distributed systems they allow to utilize concurrency; (ii) by encapsulating declarative and procedural knowledge the overall system becomes easier to handle. Both of these advantages apply in the case of the fulfillment problem as it was described above. The computations that are local to a transporter, i.e. to decide whether it is able to execute an order, can be done concurrently. Furthermore, any details about the characteristics of a transporter (for example, its current schedule, the route chosen etc.) can be encapsulated in the transporter and hidden from the overall assignment process.

However, a *totally decentralized* approach is not appropriate. The task of assigning orders to transporters is a global process because an order may only be assigned to *exactly one* transporter. Consequently, to globally optimize the scheduling process, fully autonomous agents would need a large amount of coordination. Especially in the material flow scenario, every transporter needs to be coordinated with (potentially) every other transporter. The high communication costs that would originate from such a design will outweigh the advantage of distributed computing. Consequently, it is reasonable to introduce a central component that is responsible for global aspects and that coordinates the assignment process. In particular, the evaluation should be done locally by the transporters and the following assignment globally by a coordinator, so that the schedule satisfies global optimality criteria. This has led to the architecture that will be described next.

4.1 The Principal Design

The architecture consists of an agent for every transporter and a *coordinator* that controls the overall scheduling process (see figure 4). The coordinator receives the new transport orders and keeps track of the order assignment; it announces, assigns, and retracts orders if necessary. On the other hand, each transporter has a (local) schedule for its assigned orders that contains the corresponding action sequence (cf section 2.1). On the basis of this schedule, it performs an analysis of new orders.

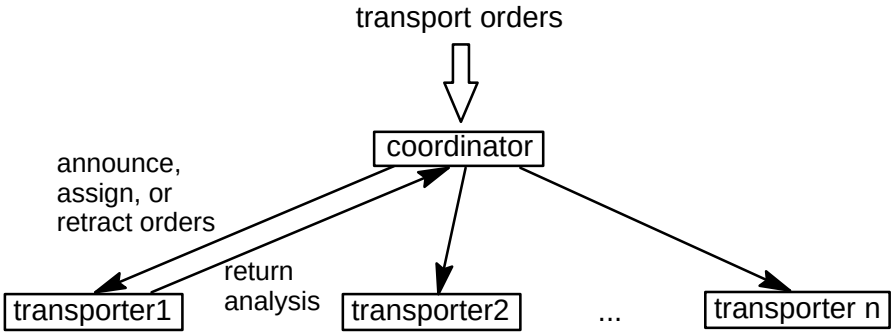

Figure 4: Interactions for the Scheduling Process

The top-level scheduling algorithm is shown in figure 5. The system is in a constant cycle of analyzing and assigning orders. Each cycle goes through four phases. Given a pool of new orders, the coordinator heuristically selects a subset and broadcasts it to the transporters. A good strategy is to announce only those with deadlines in the near future. In the second step, every transporter analyzes the orders with respect to its current situation and its specific abilities (for more details see section 4.2). The results are sent to the coordinator who evaluates them with respect to global criteria. In particular, the coordinator searches for an assignment that covers a maximum number of orders. Then the transporters are informed of the assignment and all orders not yet assigned are either declared as *impossible* or must be splitted to fit the transport capacity of the transporters (for more details see section 4.3).

The principal structure of the scheduling algorithm is similar to the contract net protocol as it was first proposed by Smith [Smi80]. In his proposal, a manager announces tasks

to contractors; these bid for suitable tasks; and finally the manager assigns the tasks to the best bidders. The main difference is that the contract net protocol is based on *bargaining* between manager and contractors, whereas in our approach transporters only perform local computations and leave the decision making to the coordinator. In particular, the agents return the result of the evaluation to the coordinator instead of making bids, i.e. they do not bargain with the coordinator.

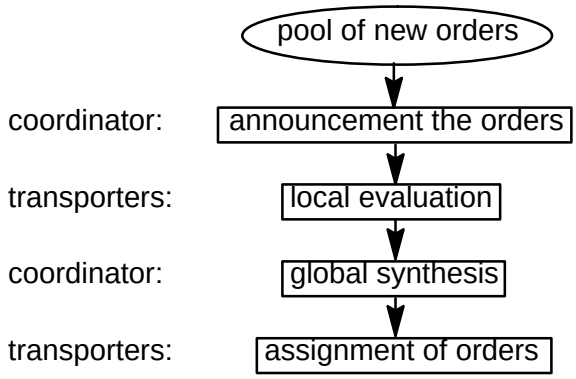

Figure 5: Phases of the Scheduling Algorithm

4.2 Local Evaluation by Transporters

The local analysis of new orders is done on the basis of the transport abilities of the transporter and its current transport schedule. The transport abilities concern, for instance, the average speed, the loading space capacity, and any other special constraints on the transportation process (e.g. routing restrictions).

The analysis is divided into two steps. In a first step, the transporter considers each transport order separately and evaluates it according to the following criteria:

1. Is it possible to execute the order?
   a. Can the active transport be done between start and finish time?
   b. Is the loading space capacity sufficient?
2. Is there a free interval large enough to execute the order?

In a second step, the transporter checks whether combinations of the announced orders can be incorporated into its current schedule since a combination of orders introduces additional constraints. But considering every combination of orders would result in an exponential algorithm (exponential w.r.t. the number of announced orders). Furthermore, even if enough computational power were available, such an effort would be in vain due to the frequent perturbations in the manufacturing process. Therefore, the transporter only estimates the possibility of a conflict by computing a conflict probability between two orders:

For every two orders compute the percentage of scheduling possibilites that result in a conflict.[2]

The result of the order evaluation is returned to the coordinator in a condensed form, mainly stating only the qualitative result of the analysis. Nevertheless, this result is incomplete because it contains only a conflict probability for combinations.

The consequence of this incompleteness is that the coordinator may assume assignments to be feasible that cannot be scheduled. In such a case, the coordinator is informed about the failure and puts the orders that were not scheduled back into the pool of transport orders. Note that a transporter can always schedule at least one order since the evaluation of a single order is correct.

4.3 Global Synthesis by the Coordinator

The coordinator combines the results of the local analyses of all transporters and searches for an assignment covering a maximum number of orders. If several maximal assignments exist, it minimizes the (average) conflict probability per transporter. For every order that has not yet been assigned, it analyzes the reasons and takes appropriate actions:

IF for every transporter it is impossible to execute the order
THEN it cannot be executed by the system.
ELSE
IF for every transporter the loading capacity is exceeded
THEN divide the order into smaller orders and re-announce them.

Afterwards, the coordinator continues with a new cycle of the top-level algorithm (cf. 4.1).

4.4 Evaluation of the Design and the Algorithm

The architecture proposed exhibits the structure that was demanded at the beginning of this section. Local computations, such as the evaluation and the scheduling of new orders, as well as transportation characteristics are encapsulated in the transporter agents. The agentification additionally allows concurrent processing. On the other hand, the coordinator has a global view on the scheduling process. It keeps track of the order assignment and receives an abstract analysis from every transporter. On the basis of this view, it computes a (maximal) assignment that meets global criteria.

The scheduling algorithm is *dynamic and adaptive by design*. The system incrementally adds new orders to the existing schedule(s). By the same mechanism, changed orders are *re*-scheduled. The coordinator first retracts the old version from the corresponding transporter and then announces the new version. In total, the system designed solves the *fulfillment problem*.

The architecture also solves the *failure-including fulfillment problem* provided that

2. An order can be scheduled at any (discrete) point of time between start and finish time. A conflict arises if the specific transportation intervals (including the arrival phase) overlap. The total set of possibilities for two orders is their cross product.

breakdowns only occur at empty transporters.[3] In this case, assigned orders are retracted and put back into the pool of new orders, just the same way changed orders are treated. Nevertheless, such events demand more timely rescheduling.

The demand for real-time behaviour has not yet been adequately met. In principle, real-time behaviour can be achieved by managing the computational resources of the agents appropriately. The computational time for a scheduling cycle mainly depends on the time for a single analysis, on the total time until all analyses are returned, and on the local scheduling. The time to find an assignment and any communication costs can be neglected. To speed up this process, the number of orders announced must be reduced to the most urgent ones (this reduces the number of combinations). Furthermore, the coordinator must assign urgent orders as soon as it receives a positive analysis if the system runs out of time to schedule these orders. Note that the transporters must analyze the orders in order to be able to guarantee their execution.

5 Implementation

In this section we will describe the implementation tool, the system performance, as well as the experience gained during the implementation phase.

5.1 Multi-Agent Tools and a Test Run

The scenario depicted in section 2 and the architecture described in the previous section have been implemented in the DASEDIS testbed for multi-agent systems [BS92]. The main advantage of this testbed is the built-in agent architecture. This architecture represents the actoric, sensoric, communicative, intentional and cognitive aspects of an agent in separate modules. Except for the module COGNITION, representing the cognitive aspects, all other modules are simulated according to the application in question. COGNITION itself is implemented as a knowledge-based system whose problem solving is based on scripts and cooperation protocols. A script encodes a stereotypical course of actions which an agent may take in order to respond to its surrounding or to achieve a goal. A cooperation protocol represents the possible flow of messages between two agents, abstracting from the low-level communication details. The structure and algorithms for scripts and protocols are generic. For the application programmer it remains to implement the knowledge of the agents in terms of scripts, resources necessary to execute a script, means to acquire the resources (possibly via protocols), and the relation of scripts to goals.

In the prototype implemented for the material flow scenario, the user creates agents for machines, storage spaces, and transporters, after having defined an environment including a graph and a production program; the coordinator is created automatically. All agents can be configured according to their characteristics, i.e., for example, the machine type, the storage capacity, or the average velocity. Once the scenario is started, the coordinator randomly computes new machine orders for a given time horizon. The structure of these orders is taken from the production program that prescribes possible order sequences for machine types. Machine orders are executed while new orders are scheduled. From the machine orders the coordinator derives the corresponding transport orders and announces them to the transporters. Then the assignment process evolves as described in the previous section.

3. For transporters carrying material, additional orders that recover the lost material must be generated.

An example test run with five machines and five transporters produced the following results: A total number of 2406 transport orders was introduced to the system in 50 steps. 780 of these were revised later on. Of the remaining 1626 orders the system was able to schedule 89 %. 9 % of orders were impossible to schedule (not enough transport capacity) and with the remaining 2 %, the system ran out of time.

5.2 Experience with a Multi-Agent Tool

The multi-agent part of the scheduling system could be readily implemented in the DASEDIS testbed. As mentioned above, DASEDIS supplies a complete agent model and high-level interaction protocols onto which the agents and the interactions of the scheduling system could easily be mapped. In particular, it was only necessary to program the knowledge base, i.e. scripts, resources, and execution conditions of scripts. Services like communication (even protocols), concurrency, and execution were provided by DASEDIS.

On the other hand, multi-agent techniques did not cover all aspects of the scheduling problem. Agents perform locally complex (cognitive) computations, such as analysing, scheduling (both done by the transporter), and assigning orders (done by the coordinator). These were implemented by classical conventional algorithms for which the programming consumed a considerable amount of time. For these subproblems, it is reasonable to employ other techniques, such as constraint-based reasoning or even Operations Research methods. The order assignment, for example, can be formulated as a clear mathematical problem.

Thus with the help of multi-agent techniques the scheduling problem was decomposed into smaller problems which were solved with other techniques.

6 Conclusion

This paper presented a multi-agent approach to material flow scheduling. In the first part, the material flow was described by a formal model. The scheduling problem to be solved was identified as being *dynamic* and *perturbed*. It turned out that many computations are local to a single transporter. On the other hand, global optimality criteria must be met. This led us to a mixed architecture, i.e. a distributed system with centralized control.

The main idea was to leave the evaluation and the scheduling of new orders to each transporter, whereas the assignment is in the responsibility of the coordinator. The coordinator receives a global view of the scheduling process with the help of a condensed result of the transporters' analyses.

The design process and the implementation were based on multi-agent techniques and tools. After identifying the need to distribute computation, a multi-agent approach provides an agent and a cooperation model that allowed the system to be designed at a *more abstract and conceptual level*. Terms such as *announce* or *assign* fit perfectly well into these models. Furthermore, multi-agent techniques supplied methods to *coordinate* the scheduling process. On the implementational level many mechanisms, such as concurrency or interaction protocols, were provided by the existing multi-agent environment DASEDIS. Thus, our experience with applying multi-agent techniques to the material flow problem can be summarized as follows: multi-agent techniques support the design process and available tools facilitate rapid prototyping and experimentation.

Our experience has shown that dynamic and adaptive scheduling of transportation demands a more 'reactive' system, in the sense that schedules are computed more rapidly. In many circumstances, for instance, new changes render the high communication effort put into the analysis obsolete. In this case, it seems more reasonable to quickly compute a preliminary schedule (for example, schedule one order at a time) and, if the situation allows, improve the schedule afterwards.

Finally, it seems promising to combine machine and transportation scheduling. First of all, the transportation problem cannot be isolated, since it is heavily dependent on machine scheduling. Additionally, a feasible machine schedule should in principle guarantee a feasible transportation schedule. Consequently, coordination of both schedules would result in better performance in comparison to the sequential scheduling of machine jobs and transportation, as is the case in current manufacturing control. Although the overall system becomes more complex, an extension of the approach presented in this paper would keep the complexity of the interrelations at a minimum.

Acknowledgement

I am indebted to Kurt Sundermeyer and Afsaneh Haddadi for their valuable comments and discussions on the contents of this paper. I do really appreciate their efforts.

Bibliography

[BS92] B. Burmeister, K. Sundermeyer: "Cooperative Problem-Solving Guided by Intentions and Perception", in: E. Werner, Y. Demazeau (eds.), Decentralized A.I. 3, North-Holland, 1992, pp. 77-82

[BHS93] B. Burmeister, A. Haddadi, K. Sundermeyer: "Generic Configurable Cooperation Protocols for Multi-Agent Systems", Pre-Proc. MAAMAW–93, Neuchâtel, 1993

[BO92] J. Butler, H. Ohtsubo: "ADDYMS: Architecture for Distributed Dynamic Manufacturing Scheduling", in: A. Famili, D.S. Nau, S.H. Kim (eds.), *Artificial Intelligence Applications in Manufacturing*, AAAI Press/MIT Press, 1992, pp. 199–213

[DK93] W. Dilger, S. Kassel: "Sich selbst organisierende Produktionsprozesse als Möglichkeit zur flexiblen Fertigungssteuerung", in J. Müller (ed.): "Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz", DFKI Document D–91–06, Saarbrücken, 1993

[OSH88] P.S. Ow, S.F. Smith, R. Howie: "A Cooperative Scheduling System", in M.D. Oliff (Ed.): *Expert System and Intelligent Manufacturing*, 1988, pp. 43-56

[PLJ+86] H. Van Dyke Parunak, P.W. Lozo, R. Judd, B.W. Irish, J. Kindrick: "A Distributed Heuristic Strategy for Material Transportation", Proc. of the Conference on Intelligent Systems and Machines, Rochester, MI, pp. 305-310

[Smi80] R. Smith: "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Transactions on Computers, C-29(12), December 1980, pp. 1104–1113