Transportation Scheduling and Simulation in a Railroad Scenario: A Multi-Agent Approach

Jürgen Lind
German Research Center for AI
Im Stadtwald B36
D-66123 Saarbrücken
lind@dfki.de

Klaus Fischer
German Research Center for AI
Im Stadtwald B36
D-66123 Saarbrücken
kuf@dfki.de

Jörg Böcker
IVE, University Hannover
Appelstr. 9a
D-30167 Hannover
boecker@ive.uni-hannover.de

Bernd Zirkler
Deutsche Bahn AG, Forschung& Technologie
Weserglacis 2
D-32423 Minden
Bernd.Zirkler@bku.db.de

Abstract

This paper presents a multi-agent system for planning, plan optimization and plan execution monitoring in a real-time railroad transportation scenario. The planning process is implemented as incremental anytime algorithm that is capable of integrating new task specifications in the ongoing planning and plan execution process. An initial solution is obtained by executing the contract-net protocol for every new task, the resulting solution is then optimized by a simulated trading approach.

The execution of the resulting plans is monitored by a plan execution monitor that dynamically controls ongoing driving or coupling activities of transportation units and that handles deviations from the original plan in the case of external events, e.g. location route failures or train delays.

1 Introduction

Efficient transportation — be it of persons or goods — is a key issue in todays industrial world [3]. Because of the immense amount of transportation tasks, it is necessary to use the available resources most effectively. Thus, computer aided — or entirely controlled — scheduling systems are key technologies in the forthcoming century.

Conventional waggon load traffic as it is performed today is shown in Figure 1: a company that wants to ship something via railroad to its customers delivers the freight to the local freight center (usually a railroad station) where it is stored until enough freight from other companies has arrived to justify a train to the regional freight center. At the regional freight center, waggons from other local freight centers that have the same direction are assembled and sent to the next interregional freight center where another re-assemblance process takes place. The decomposition of the trains is achieved in reverse order.

This approach, however, has some serious drawbacks. First of all, the waggons of individual customers must wait at the local freight centers until enough freight is delivered to make a train to the next local freight center profitable. Secondly, the re-assemblance of trains in regional and interregional freight centers is a very time consuming process that introduced additional delays in the producer-to-customer route.

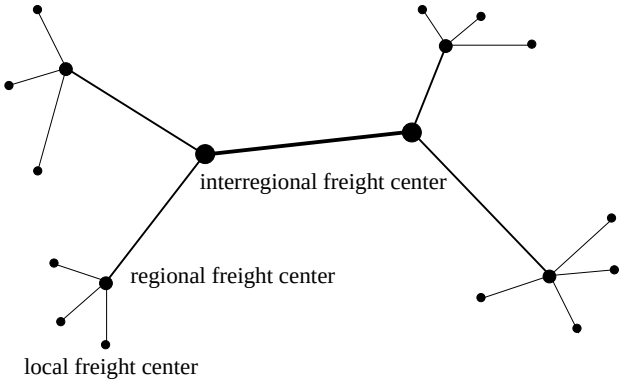

Figure 1: Hierarchical freight haulage

An alternative approach [9], [4], [15] to the classical freight transport process uses small railroad *transportation modules* (cf. [16]) instead of conventional trains. Whereas a normal train is made up of one locomotive and several freight waggons, a transportation module consists of two power units on either side of the module and three permanently coupled intermediate vehicles with a fixed number of loading spaces. Thus, a transportation module is a single unit of limited size. When a company wants to deliver some freight to a customer, it orders a transportation module at a local freight center and loads its goods on this module. The module itself is then responsible to find its way through the railroad network. The problem is now, that a location route in a railroad network cannot be used by two **independent** modules at the same time. Either a route is blocked by a single module or two (or more) modules share a route by hooking together at the beginning of a location route and splitting up afterwards. In order to use the underlying railroad infrastructure most efficiently, the railroad modules should share as many location routes as possible while taking care of their local constraints.

The main advantage of this approach is that it avoids a central planning authority that schedules all transportation modules. Instead, each module is responsible to achieve its goal which is to deliver its freight to some destination node in the network. Additionally, each module performs some local optimization of the network throughput by sharing as many location routes with other modules as possible. The local optimization process of all modules eventually leads to a high, though usually only suboptimal, degree of resource efficiency. Besides this major advantage, a decentralized approach implies less coupling operations during the train composition process, a high degree of customer accessibility and lower costs because of the effective location route usage. However, the re-organization of the traditional freight transport would require some major investments by the railroad companies and thus the potential of this idea must be carefully investigated in a simulation. The work presented in this paper is funded by the Deutsche Bahn AG in order to evaluate the pros and cons of the basic approach.

The coordination of the local optimization processes of many thousand modules in a practical scenario is a computationally demanding process and requires a sophisticated algorithmic framework. The major requirements towards a real-world system are listed in the following paragraphs.

First of all, a scheduling system should be capable of dealing with an incomplete problem specification. The classical operations research based approaches assume that the entire problem specification is given at the systems start-up time. Unfortunately, this assumption often does not hold in the real world! Transportation company's usually receive customer tasks over time and not only at the beginning of the planning process. Thus, the company cannot wait until all task specifications are available, then start the planning process in order to find an optimal plan and finally start to execute this plan. Instead, the company must overlap the planning process based on the available data and plan execution monitoring. Incoming tasks must then be integrated in the ongoing planning and execution process.

The second requirement is highly related to the first point and deals with the problem to establish a proper relation between the system and the real world it is supposed to model. A prominent example for the gap between research and reality are order dispatching systems for haulage companies: usually, the respective systems try to compute optimal routes and schedules for the companies trucks, but they fail to monitor the plan execution process and are thus not able to react to unforeseen situations such as mechanical failure of trucks or traffic jams making it impossible to maintain the original schedule.

Finally, a system should not necessarily try to find an optimal solution for a given problem. Although optimality is a desirable property of a solution, it is often the case that the computational effort to find an optimal solution is too high for realistic problems. Thus, a mechanism that is capable of finding a rather good solution quickly and then improving this solution if sufficient time is available, is an alternative to classical approaches. Algorithms of this type are usually referred to as *anytime algorithms*.

Multi-agent systems [8], [12] are particularly well suited for the transportation domain because of the inherent complexity and natural degree of distribution [5]. A lot of research work has be done in the past years [6], [7], [10] leading to fully operational systems eg. the TELETRUCK system presented in [2].

The system presented in this paper is fully implemented in Oz [14], a high-level programming language that combines constraint inference with concurrency. Oz is a dynamically typed language and supports object-oriented system development, finite domain and feature constraints. The implementation consists of approximately 74 classes with 18000 lines of code.

This paper is organized as follows: Firstly, we will describe the problem to be solved and the basic technical constraints to be maintained. We will then present the overall structure of our software system, the generic agent architecture used and the instantiation of the generic model to the specific scenario. Afterwards, the agents planning facilities, the plan optimization process and the functional units that monitor the execution of the resulting plans are explained. We then present first experimental results and conclude the paper with a short summary and an outlook on our future work.

2 Problem description

The overall goal of the system presented in this paper is to reduce the cost for a given set of transportation tasks in a railroad network. Each task is specified as a

tuple consisting of the origin and the destination node, the earliest possible departure time, the latest allowed arrival time and an additional time stamp indicating when the task is announced to the system. Thus, the set of tasks is not fully known to the system at start-up time, new tasks arrive during the planning process and may require a revision of the already assembled plan in order to reduce cost. A typical time profile for incoming tasks is depicted in Figure 2: at start-up time, 2000 tasks are known to the system; during the next 24 hours (=1440 minutes) additional tasks arrive, summing up to a total of 5000 tasks at the end of the 24-hour period.

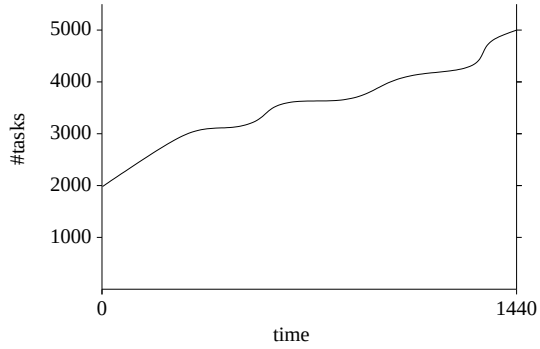

Figure 2: Task arrival time profile

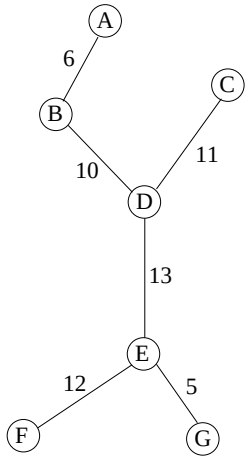

Figure 3: Example railroad network

A transportation task is served by a transportation module mentioned in Section 1, we assume that each task can be served by an individual module, i.e. there is no need to hook two or more modules together to serve a single task. Vice versa, we assume also that a module cannot serve more than one task at a time. All tasks occurring in the system are transportation requests in a railroad network; a network is graph consisting of several nodes connected via so-called *location routes*. In the current version of the system, we use an abstracted map of the German railroad network with approximately 250 nodes and 350 links. In this paper, however, we use the seven node network shown in Figure 3 to illustrate the basic ideas of our approach. The numbers on the routes in Figure 3 indicate the distance between two nodes connected via a location route.

Whenever a module serves a transportation task, it computes the path from the origin to the destination node with a shortest path algorithm. The module then rents the intermediate location routes for a certain time window from the *net manager*. The time window for each location route is uniquely determined by the earliest departure time and the latest arrival time of the transportation task. When a location route is allocated by a certain module, the route is blocked for other modules during this time interval. In order to reduce route blocking, however, two or more modules can decide to share a particular location route.

Route sharing means, that two or more modules hook together at the beginning of a location route (or of a sequence of consecutive routes) and split up afterwards. Route sharing has two advantages: firstly, it increases the average utilization of location routes because it enables more than one module to use a location route

at the same time. Secondly, the cost for renting a location route are reduced for an individual module by distributing the full cost among the participating modules.

To illustrate the idea of route sharing, consider the following example with two transportation task $T_1\langle B, F, 10, 60, 0\rangle$ and $T_2\langle C, G, 11, 150, 0\rangle$. Let the cost function for a location route be $c(w, n) = \frac{w}{n}$, i.e. the weight w of the route divided through number n of modules using it simultaneously. Now, if the two modules serving the respective tasks act independently, the transportation costs for the first module are 35.0 units and 29.0 units for the second. If, on the other hand, the two modules decide to cooperate and to share the common location route between node D and node E, the transportation cost reduces to 28.5 and 22.5, respectively. The computational problem in conjunction with location route sharing is to identify sets of tasks (i.e. their respective modules) that can share location routes. The limiting factors in this respect are the compatibility of transportation paths and time windows.

In the course of this paper, we will refer to sets of cooperating modules as *unions* where each union is determined by the participating modules. Thus, unions are-meta level concepts; a union emerges when at least two modules decide to cooperate by sharing location routes and it ceases to exists when all modules within the union have completed their respective tasks.

In this section, we have presented the requirements and goals of our system. In the next section, we will present the overall system architecture of our multi-agent system.

3 System Architecture

In this section, we give an overview over the architecture of our software system that solves the aforementioned scheduling task. The most important objects and their interconnections are depicted in Figure 4; the arrows between the objects denote the flow of control or the flow of data, respectively. Firstly, we will describe the building blocks of the architecture and then we will explain the data/control flow.

The *simulation engine* provides the system wide clock that is needed to synchronize the concurrently active objects. Furthermore, the simulation engine implements an alarm mechanism that allows the objects to suspend their execution until a particular clock cycle. Finally, the simulation engine supports system development and debugging by allowing the designer to run the system in single-step mode.

The *task manager* agent is responsible for handling incoming transportation requests. The task descriptions are externally specified; each task has a time stamp that indicates, when (i.e. in which clock cycle) the task should be announced to the system. This mechanism allows the simulation of time varying task arrival-times (see Section 2, Figure 2).

Each *union* agent shown in Figure 4 performs local planning and optimization, the interdependencies between several unions are handled in negotiation protocols that allow the agents to improve their local plans. Global optimization then emerges from the local optimization steps of the agents. In Sections 4 and 5, we will give a thorough description of the agent architecture and the negotiation protocols used.

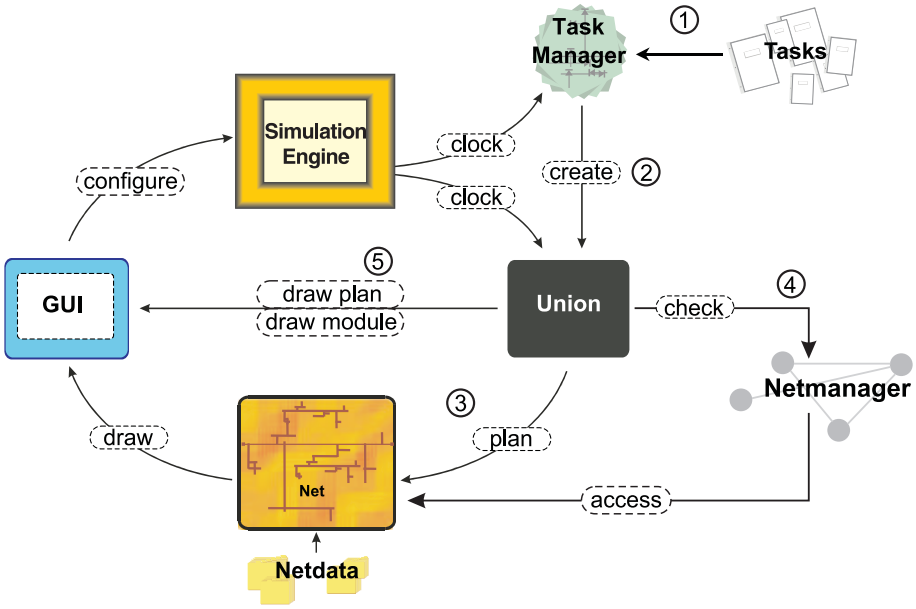

Figure 4: System Architecture

The *net manager* agent controls the location route usage by serving the route allocation requests of the unions. The net manager guarantees, that no location route is allocated to more than one union during a particular time window. Furthermore, the net manager can simulate additional traffic on the location routes (i.e. passenger trains) and thus limit the available time windows for some location routes. This feature is very important for the simulation of real-world transportation tasks. The net manager offers a stateless two-step protocol for the allocation of location routes:

check This message type sends a location route identifier, a time window during which the route should be allocated and the minimal speed by which the route should be travelled to the net manager. Note that the time window is usually larger than the time required by the querying union to travel the location route (the required time is determined by the minimal speed and the length of the route). It is now the task of the net manager to find a time window of the appropriate length that fits in the existing route allocations made by other unions. If it is possible to find such a window, the net manager returns a positive indication to the unions, otherwise a negative indication.

book This message type is used by the union to book a particular location route for a given time window. Note that it does not require a prior **check** operation although it is highly unlikely that a union will find a free time window on a route if it omits the prior check. Note also, that multiple **book** operations on the same location route but by different unions are handled in a first-come first-serve manner. This is due to the fact that the net manager is organized as stateless server, ie. it does not keep track of the order of check operations. The advantages of a stateless organization are that it limits the memory usage of the net manager agent and that is invulnerable to system failures.

free A location route that was allocated by a particular union can be freed by this union if the ongoing planning process makes the use of location route obsolete or when the time window during which the route is needed has changed.

The *net* agent contains the description of the underlying railroad network and provides a route planning mechanism that can be used by the agents to find an optimal route between any two nodes in the network.

The *graphical user interface (GUI)*, finally, is responsible for the visualization of the system activities and for the configuration of the system behaviour. The visualization of the system state is divided in a static component that allows the user to inspect the current status of the agents (e.g. their current plans etc.) and a dynamic component that allows the user to simulate the execution of the agents plans. Additionally, the GUI provides various configuration options that can be used to change the system behaviour. A screen-shot of the system is shown in Figure 5.

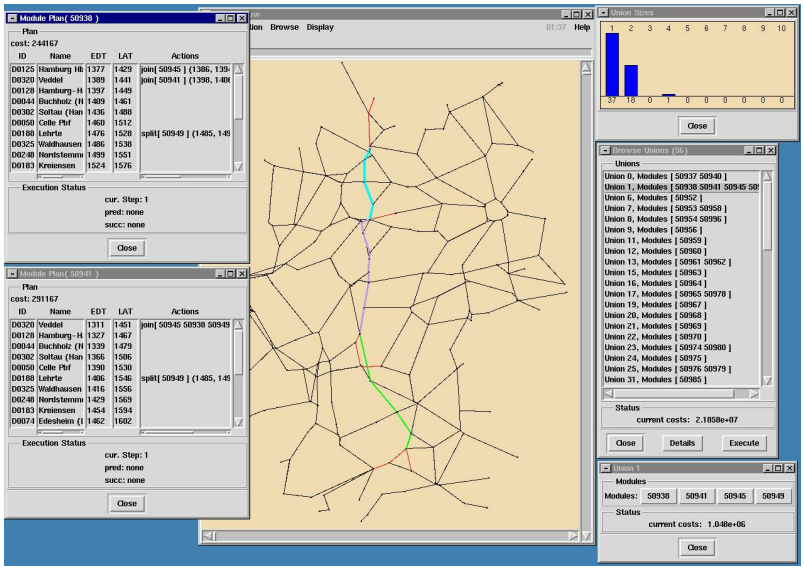

Figure 5: A screen-shot of the system with the plan browser, the map window and the union browser

Now that we have introduced the main objects of our system, we will give a brief description of the data/control flow indicated by the arrows in Figure 4. The numbers shown in Figure 4 illustrate the basic execution cycle that is executed whenever a transportation task is announced to the system. However, we have left out the plan optimization steps in this first overview. Plan optimization takes place in two additional steps that will be described later in the course of this paper.

The basic execution cycle is the following:

1. The task manager receives a task description
2. The task manager creates a new agent to handle the task
3. The new agent requests the optimal path from the net agent

4. The new agent checks the validity of the resulting plan by querying the net manager for the required location routes according to the previously specified protocol (**check**). If the routes are not available during the time windows of the plan, an alternative route is considered by backtracking to step 3.; otherwise the location routes are allocated (**book**). This process iterates until either a valid plan is found or the retry-bound is reached. In the later case, the new agent is deleted and an error message is raised.

5. The new agent starts to execute its plan when the time is up.

Now that we have introduced the overall system architecture, we will present the agent architecture used in this scenario in the next section.

4 Agents

Multi-agent systems are particularly well-suited for the scenario described earlier because they allow a very natural mapping from the entities occurring in the scenario to agents. In the problem domain presented in the previous section, we have two natural candidates for becoming the agents in the implementation of the system: the individual modules and the unions that emerge when two or more modules decide to share a location route. However, each additional abstraction increases the complexity of the resulting implementation and therefore, we have decided to model the unions as the agents in our system. A single module is encapsulated in a (so-called *degenerated*) union and thus we avoid the additional complexity in the system design. The advantage of applying this scheme is that we do not have to differentiate between modules and unions; every active entity in the system is a union and that 's it!

In the following sections, we present a generic agent architecture that was developed at the DFKI and the instantiation of the generic model to the specific scenario.

4.1 InteRRaP

InteRRaP [11] is a generic agent architecture for situated agents that integrates reactive behaviour and deliberation. The architecture was designed for agents that exist within multi-agent systems and thus some emphasis is on the communication aspect.

As depicted in Figure 6, InteRRaP is a layered architecture that consists of three concurrent layers:

Behaviour Based Layer (BBL) This layer implements the reactive behaviour of the agent, i.e. this layer reacts to external requirements without any explicit reasoning, thus it reacts very fast.

Local Planning Layer (LPL) This layer performs the planning process of an individual agent, it is also responsible to monitor the plan execution of the agents current plan.

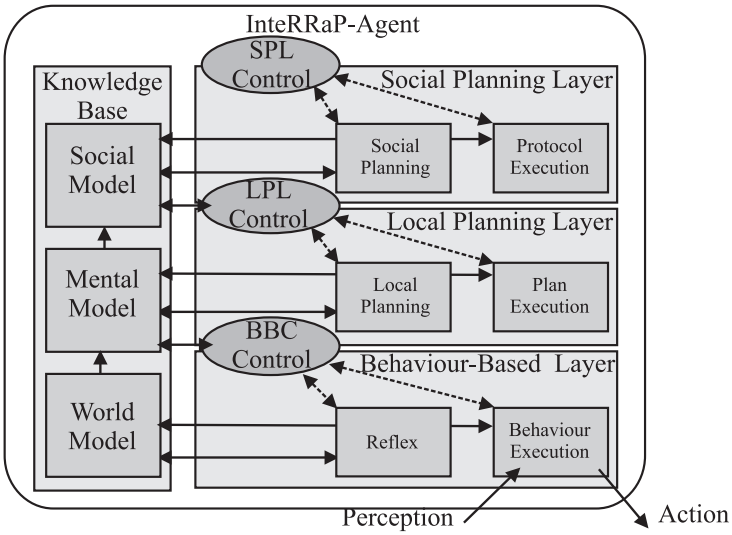

Figure 6: InterRRap

Social Planning Layer (SPL) This layer is responsible for the coordination with the other agents within a multi-agent system. The coordination with the other agents is achieved with explicit negotiation protocols.

All these layers run concurrently, the intra-agent coordination between the three layers is achieved via the knowledge-base. The knowledge base is conceptually divided in three layers (world model, mental model, social model), but each layer has access to the knowledge on every level. The conceptual discrimination, however, allows for a clearer design because most of the information stored in the knowledge base can be associated with a particular layer.

The InteRRaP architecture offers a generic framework for agent design that must be instantiated for the particular needs of a concrete scenario. Usually, some aspects of the generic framework are more interesting in a given scenario than others; the instantiation for the problem addressed in this paper is presented in the next section.

4.2 Union Agent Architecture

The union agents implement the generic model presented in Section 4.1 in the context of the specific scenario. The emphasis of the instantiation is on the local planning layer and the social planning layer, the behaviour based layer is not as important at the moment. However, in a later version of the system with a more realistic plan execution simulation, the BBL will gain more importance.

A functional decomposition of the relevant parts of the union agents is shown in Figure 7:

Communication Unit (CU) The communication unit is responsible for executing the negotiation protocols between the agents. These protocols are used for the local planning process, to optimize the local plan of each agent and to

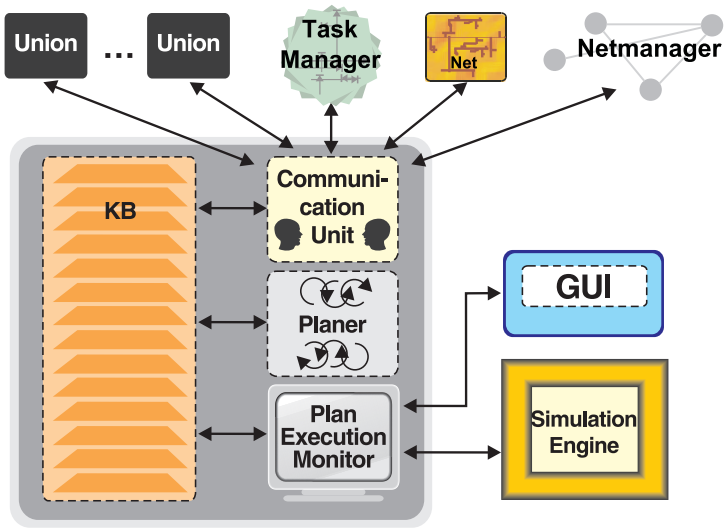

Figure 7: Union Agents

handle the location route allocation process. The protocols for the plan optimization steps will be explained in detail in the following section.

Planer The planer uses the shortest path service provided by the net agent and the route allocation protocol of the net manger to find the local plan for an agent.

Plan Execution Monitor (PEM) The plan execution monitor monitors the execution of the current plan of an agent and adapts the plan in the case of external events, e.g. location route failures etc.

The arrows between the agent and the other parts of the system shown in Figure 7 indicate the connections of an agent to the outside world. Each arrow between the communication unit and another agent represents a logical connection that is used to execute the negotiation protocols between the agents. The connections to the simulation engine and the graphical user interface, on the the other hand, are achieved by method invocation within the object system.

The goal of the local planning process of a union agent is to find an optimal connection from its source node to its destination node while preserving the time constraints given by the earliest departure time (EDT) and the latest arrival time (LAT) of the respective task. Local planning is achieved in several steps: firstly, the union requests the shortest connection from the source node to the the destination node from the net agent, then it calculates the time windows for each location route. A union must allocate the location route within the computed time window if it wants to preserve to overall constraints given by the EDT and LAT, respectively. Usually, the size of a time window exceeds the time needed to cover the location route and thus the union has some choice for the allocation of the route. A negotiation process between the union and the net manager is initiated where the two agents try to find a time window for the location route that satisfies the constraints of both agents. This negotiation process iterates for each location route; if all route

requests can be served, the union has found its local plan. If no plan can be found, an error condition is raised.

In this section, we have presented the instantiation of the generic agent architecture to the specific scenario and illustrated the local planning process of the union agents. In the next section, we will present how the local plans of the unions are optimized.

5 Optimization

Until now, the agents in our multi-agent system have performed a local planning process only, ie. each agent plans for its local goal without taking notice of the other agents. In this section, we will present an optimization process that is executed by the agents and that reduces their local cost as well as the total cost of the agent society. The key optimization technique in the scenario presented is to find agents that use the same location routes and to have these agents share the respective routes by coupling together at the beginning and by splitting up afterwards. This peer matching is achieved by negotiation processes between the agents in the agent society. In our system, we have combined two negotiation protocols to achieve the global optimization of the schedule.

A schedule is generated in the following way: an initial solution for the module schedule is obtained by running the *contract-net* [13] protocol whenever a new task is announced to the system. New tasks are incrementally integrated in the existing scheduling which guarantees, that always a solution for the problem (as far as it is known to the system) exists. However, this solution may be (and usually is) not optimal. In order to improve the quality of the existing solution, the *simulated trading* [1] protocol is run on the set of tasks (or the respective modules) currently known to the system. Unfortunately, executing the simulated trading protocol is a computationally expensive operation and so it is executed only periodically — either after a fixed number of new tasks has been added to the existing solution or explicitly triggered by a user request.

The hierarchical organization of the two negotiation protocols supports the aforementioned properties of incrementality and anytime behaviour and also implies a high degree of flexibility because of the possibility to asynchronously execute the simulated trading process at any time. In the following sections, we will present the instantiation of contract-net and simulated trading protocol for our specific scenario.

5.1 Contract-Net

In our system, the contract-net protocol [13] is executed whenever a new (degenerated) union has computed its local plan as described in Section 4.2. The union then initiates the contract-net protocol as the manager and offers the plan to the other currently active unions. These unions check if they contain one or more modules that are a potential sharing peers and if this is the case, they offer a sharing commitment to the new union. The new union collects these offers and selects the one that has the largest cost saving potential. It then transfers the module to the winning union and ceases to exist because it does not contain other modules.

If no union offers a sharing commitment, the new union remains active as degenerated union.

Thus, a major step in the contract net protocol is on the side of the bidders and it is to identify sharing opportunities with a newly created module. In the next section, we will therefore give a rather detailed explanation of the operator that is capable of finding a maximum degree of overlappings among an arbitrary number of plans. This operator is then used to compute the cost functions of the bidders.

5.1.1 Plan Integration operator

We will now develop a plan integration operator for the plans of n distinct modules. Plan integration means, that the operator takes the n plans as input and modifies these plans by inserting *join* and *split* actions such that the resulting n plans imply a maximum degree of location route sharing. The integration operator is used by the unions to decide whether they can integrate a new module in their set of modules or not.

DEFINITION 5.1 (NOTATION AND DATASTRUCTURES) *Each of the m modules has a unique identification number $i \in \{1,\ldots,m\}$ and is denoted by M_i. Furthermore, each union has a unique identification number $j \in \{1,\ldots,k\}$ as well. A union of n modules $M_{i_0},\ldots M_{i_n}$ is written as $U_j^{i_0,\ldots,i_n}$. A plan for an individual modules is a sequence of plan steps where each step consists of a node identifier, a time window and a list of actions to be performed during that plan step. Each of these actions is described by its type, the peer modules, its duration and the time window $\underline{t},\overline{t}$ during which it must be executed. Time windows used in the module plans have a lower bound ($\underline{t}$) and an upper bound ($\overline{t}$), location routes between any two nodes in the network are written as sequence of node names, eg. $\overline{AB}$, $\overline{BDEF}$,*

Each element of a plan be accessed via a dot notation scheme, list elements are accessed via their index number. For example let
[(NodeId:A, $\underline{t}$:10, $\overline{t}$:16, actions:nil)
(NodeId:C, $\underline{t}$:10, $\overline{t}$:16, actions:[(type:join peers:[M_2 M_3] duration:4 $\underline{t}$:11 $\overline{t}$:15)])
(NodeId:D, $\underline{t}$:10, $\overline{t}$:16, actions:nil)]

be the plan of module M. Then we have $M.A.\overline{t} = 16$ or $M.C.actions.1.peers$ =[M_2 M_3].

In the course of this section, we will use a rather simple example to illustrate the basic ideas and to give the reader rough feeling for what is going on during the plan integration process. In the example, we have two unions U_1^1 and U_2^2 with modules M_1 and M_2 serving tasks $T_1\langle A,F,10,60,0\rangle$ and $T_2\langle C,G,9,50,0\rangle$ respectively. Basically, plan integration is achieved in five steps:

Find location route matches The first step in the plan integration operation is to find an overlapping sequence of location routes in the plans. In the example, the path of module M_1 is $\overline{ABDEF}$ and the path of module M_2 is $\overline{CDEG}$. Thus, the two paths overlap in $\overline{DE}$. If no overlapping is found, the plan integration process is aborted. For n plans, the general overlap condition is $\forall i \exists j : overlap(M_i,M_j) \neq \emptyset$, i.e. for each module, there must exist at least one sharing peer in the union.

Generate joint actions If the overlap condition holds, the next step in the plan integration process is to generate *join* and *split* actions for the respective plans. These actions are referred to as *joint actions* because they require two modules to coordinate their individual actions. In the example, the two modules join in node D and split in node E, the actions to be inserted are therefore

M_1*.D.actions =[(type:join peers:*$[M_2]$ *...)]*
M_2*.D.actions =[(type:join peers:*$[M_1]$ *...)]*
M_1*.E.actions = [(type:split peers:*$[M_2]$ *...)]*
M_2*.E.actions = [(type:split peers:*$[M_1]$ *...)]*

Minimize number of joint actions The number of joint actions generated in the previous step is not optimal because the generation process only considers the local context of the action, i.e. only a single step in the plan. Due to prior actions of a module, however, some actions are obsolete and can be eliminated. To illustrate this situation, assume another Module M with task $T\langle B, G, 10, 60, 0\rangle$. Integrating the three modules M_1, M_2 and M, yields three overlappings

1. $(M_1, M_2) = \overline{DE}$
2. $(M_1, M) = \overline{BDE}$
3. $(M_2, M) = \overline{DEG}$

resulting in three bilateral actions pairs

1. (M_1, M_2): *join* at D, *split* at E
2. (M_1, M): *join* at B, *split* at E
3. (M_2, M): *join* at D, *split* at G

This results in a generation of two *join* actions for module M_2 at node D: one with M_1 and one with M. These two actions can be reduced to a single action M_2*.D.actions = [(type:join peers:*$[M_1 M]$ *...)]* because M_1 and M are already linked due to their prior *join* operation at node B. While this appears to be rather trivial in this context, this is not the case in more complex plans where previous *join* and *split* actions must be recursively traced for a large number of modules.

Specify joint action constraints In this step, the time windows of the newly generated actions are specified. The conditions that must hold in this respect are that actions must take place within the time windows of the plan steps of the respective modules and that the joint actions must take place simultaneously. In the example, the resulting constraints are

$M_1.D.actions.1.\underline{t} = M_2.D.actions.1.\underline{t} = max(M_1.D.\underline{t}, M_2.D.\underline{t})$
$M_1.D.actions.1.\overline{t} = M_2.D.actions.1.\overline{t} = min(M_1.D.\overline{t}, M_2.D.\overline{t})$

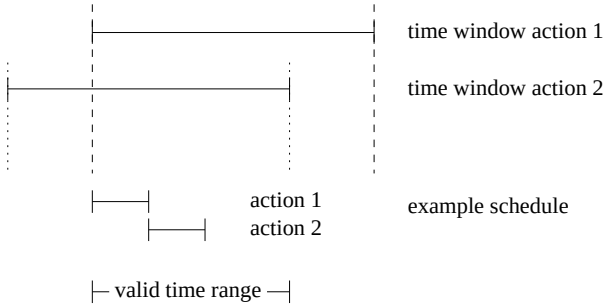

Figure 8: Action Scheduling

Find action schedule In the last step of the plan integration process, the operator must guarantee the existence of a schedule for all actions in each plan step of each module. This means, that all actions occurring in a plan step must be serialized in a way that the action executions do not overlap in time.

For example let M_i.*N.actions = [(type:join peers:*$[M_j]$ *duration:4* $\underline{t}$*:10* $\overline{t}$*:30)*
(type:join peers:$[M_k]$ *duration:4* $\underline{t}$*:14* $\overline{t}$*:34)]*

be the plan step of module M_i at node N. The time windows for the two actions are shown in Figure 8, the task of this step in the plan integration process is to arrange these actions within their respective time windows such that they do not overlap and that they take place in the time interval given by cutting the action execution time intervals. A valid schedule for the two actions is also shown in Figure 8.

Note that, as one can see in Figure 8, usually multiple schedules for the actions exist (eg. action 1 could as well be scheduled after action 2). However, the time windows of the actions are not committed to a particular schedule because this implies an unnecessary restriction imposed at planning time. At planning time, it is sufficient to know that a schedule exists, concrete commitment to a particular schedule are delayed until execution time giving the participating modules a maximum degree of flexibility. A delay of commitment until it is necessary is called a *least commitment strategy*.

After these steps of the plan integration process have been successfully performed, the output of the plan integration operator are n module plans that satisfy the overlap condition mentioned before.

5.1.2 Contracting

To decide whether is pays to integrate a new module into an existing union, we have used the following cost functions. The *insertion cost* c^+ of a module m into an existing union U computes to

$$c^+(U,m) = \sum_{r \in U'.routes \wedge |r.users|=1} \omega(r) \tag{1}$$

where U' is the is the union that emerges when the plan integration operator presented in the previous section is applied to the plans of the modules in U and

module m. $U'.routes$ denote the location routes used by U' and $|r.users|$ is the number of modules using location route r simultaneously. Thus, the insertion cost are the additional costs that occur because of location routes that are not already used by the union.

The *deletion costs* (or *savings*) c^-, on the other hand, occur when a module m is deleted from a union U and compute to

$$c^-(U,m) = \sum_{r \in m.routes \wedge |r.users|=1} \omega(r) \tag{2}$$

ie. the savings are the costs of the routes of module m that are not shared with other modules.

Thus, it pays to switch a module from an union U_1 to another union U_2 if the insertion costs for the module in U_1 are less than the costs that occur when the module is deleted from union U_2, ie. $c^+(U_2,m) < c^-(U_1,m)$. To decide which of the potential contractors $U_1,\ldots,U_k$ is the best from the point of view of the manager M, the manager selects the union U_i where $c^-(M,m) - c^+(U_i,m)$ has a maximum.

The contract net protocol presented in this section is used in our system to obtain an initial solution for the scheduling task. However, this solution is usually not optimal and can thus be improved further. This improvement process is done with a simulated trading approach presented in the next section.

5.2 Simulated Trading

The simulated trading protocol [1] is an algorithm designed to improve existing solutions, not to construct new solutions from scratch. In our case, the input and the output of the protocol are valid schedules where the cost of the output are always less or equal to the cost of the input. This is trivially true since the output can always be the input if no cheaper schedule exists. However, this property is nonetheless important because it guarantees, that the protocol can be aborted at any time and still yield a valid solution. Furthermore, if the protocol is given enough computation time, it is guaranteed to find the optimal solution. Now, how does this work?

The protocol is initiated by a special agent, the *stock manager*. In the course of the protocol execution, the agents (here called *traders*) perform several rounds of hypothetical trading, i.e. the traders either choose to sell some of their goods or to buy something from others. In our context, a sell operation corresponds to removing a module from a union and a buy operation corresponds to integrating a module in a union. Thus, the unions try to optimize their cost by exchanging unprofitable modules with better ones. The decision which module to sell depends on a probability distribution induced by the potential cost reduction if the module was sold. Vice versa, the decision to buy a module offered by another union depends on the potential cost reduction if the module would be integrated in the union.

Simulated Trading takes place in several rounds where each union decides whether to sell a module or to buy a module offered by another union. After a fixed number of trading rounds, the stock manager has collected the hypothetical sell and buy actions and it must find a valid *trading match* in the set of actions. There a several validity requirements for a trading match e. g. the difference between deletion

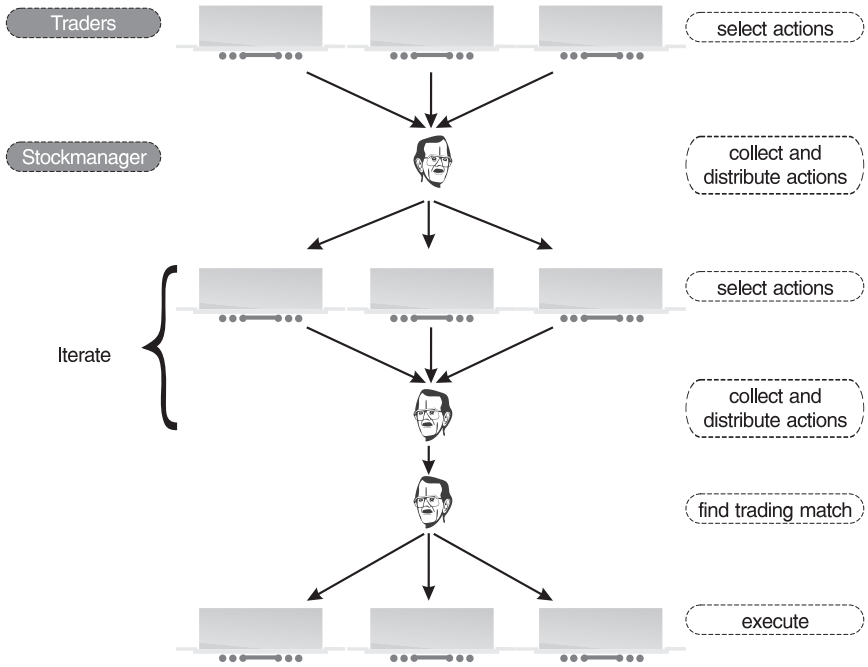

Figure 9: Simulated Trading

and insertion costs (Section 5.1.2) must be positive, or there must not be two buy actions on the same sell operation, etc. Finding a trading match is NP-complete [1] and accounts for the computational complexity of the simulated trading protocol. Usually, however, the size of the trading graph still allows to find a trading match in reasonable time [1]. If a trading match is found, the stock manager informs the traders which actions must be executed, i.e. which modules must be exchanged.

6 Plan Execution Monitoring

Each union has a functional unit that monitors the plan execution process of the modules in the union. The plan execution monitor (PEM) is the link between the planning unit and the real world (or a simulation engine that simulates the real world). The PEM controls the usage of location routes and the coupling activities of the modules. To illustrate how plan execution monitoring works, recall the plan of module M_2 from the example in the previous section:
[(NodeId:C, $\underline{t}$:9, $\overline{t}$:21, actions:nil)
(NodeId:D, $\underline{t}$:20, $\overline{t}$:32, actions:[(type:join peers:[M_1] duration:4 $\underline{t}$:11 $\overline{t}$:15)])
(NodeId:E, $\underline{t}$:33, $\overline{t}$:45, actions:[(type:split peers:[M_1] duration:3 $\underline{t}$:11 $\overline{t}$:15)])
(NodeId:G, $\underline{t}$:38, $\overline{t}$:50, actions:nil)]

When the module starts to execute this plan at time $t = 9$, it asks the PEM whether the location route from C to D is available as it should be because it was allocated at planning time. However, the route may not be available at execution

[1] For a 5000 task problem, the execution of the simulated trading protocol with four rounds takes approximately 40 minutes on a Sun UltraSparc.

time due to external reasons, e.g. route blocking due to mechanical failure of another module while using that particular route. It is the task of the PEM to communicate with the net manager in order to check if the route is available or not. If the module is allowed to use the route, it departs from node C. When it arrives at node D, it issues a coupling request to the PEM, indicating that it is waiting for module M_1 to join with it. If module M_1 has already arrived at node D, the coupling action can start if M_1 is not engaged in an ongoing coupling activity with another module. If M_1 has not arrived yet or is currently engaged, the coupling action of M_2 is stalled. If M_1 does not become available within the time interval specified in the *join* action of M_2, M_2 departs from D without coupling with M_1 because otherwise M_2 will miss its scheduled latest arrival time at the goal node. If the coupling activity can be completed as it is scheduled, the two modules depart from D after the coupling is finished. When they arrive at node E, the two modules inform the PEM of their parent union that they want to split and after completing the *split* action they depart from node E for their respective goal nodes.

This example is, again, highly simplified. If more than two modules are involved in *join* or *split* actions additional plan integrity constraints must be satisfied. If, for example, three modules $M_1M_2M_3$ are coupled in this order and M_2 must split from M_1 and M_3, an additional *join* action between M_1 and M_3 must be generated because M_1 and M_3 are supposed to remain coupled. However, no additional *join* action is necessary if the original module order is $M_2M_1M_3$. The decision of whether to generate additional coupling actions or not must be taken by the PEM upon plan execution time, depending on the actual coupling order of the modules.

7 Experimental Results

The interpretation of experimental data is a difficult issue when no comparable data is available — which is unfortunately the case with the problem presented in this paper. Because of the prototypical character of the system, no other systems – at least to our knowledge – that solve this railroad scheduling task exist. Because of the lack of comparable data, we thus have to evaluate our system compared to the case where no optimization is performed at all. However, even this limited evaluation yields some interesting insight in the value of a multi-stage optimization process.

The experimental data presented in this section is the results of running the system on a set of ≈ 5000 transportation tasks. These tasks were provided by the Deutsche Bahn AG and represent transportation orders that really occurred. All tasks were processed according to the conventional method explained in the introductory section; unfortunately no data about the costs for serving these tasks is available to us.

Table 1 shows the costs that occur when the system is run with different quality requirements on the solution and different levels of optimization. The optimization is either omitted, done by running the contract-net protocol whenever a new task occurs or by running the contract-net protocol and then the simulated trading protocol on the initial solution. The *minimal sharing length* admits only solutions, where two modules share at least the given number of kilometers. This requirement restricts the possible coupling activities and it has a strong influence on the optimization process as we can see from Table 1 and Figure 10. Especially Fig-

		min. sharing length		
		0	100	200
Optimization Type	none	$1.18212 \cdot 10^{10}$	$1.18212 \cdot 10^{10}$	$1.18212 \cdot 10^{10}$
	CN	$8.10898 \cdot 10^{9}$	$7.25189 \cdot 10^{9}$	$7.76054 \cdot 10^{9}$
	CN+ST	$7.91374 \cdot 10^{9}$	$7.14646 \cdot 10^{9}$	$7.76054 \cdot 10^{9}$

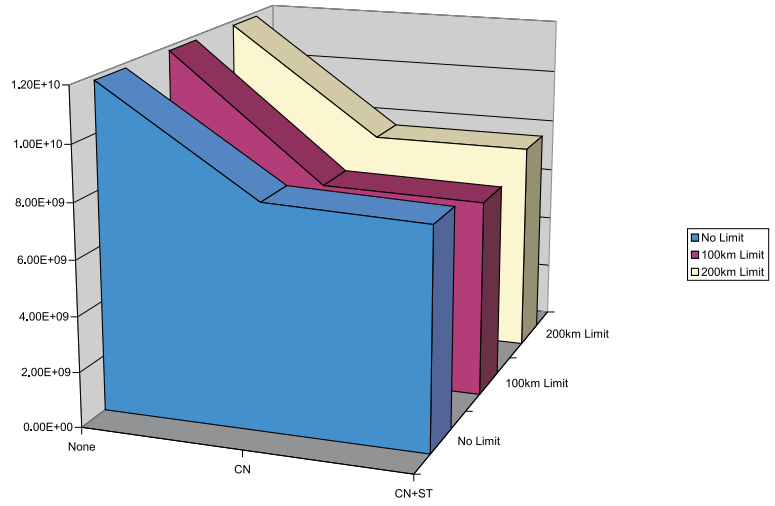

Table 1: Absolute Costs

ure 10 reveals an interesting fact: although the side constraints seem to be harder (because of the minimal sharing length requirement), the obtained solution is better than in the relaxed problem. The explanation for this behavior is that a minimal sharing length prevents some unions to commit themselves to sharing partners with a low cost saving potential (ie. with only a short common path) to early. These early commitments made in the case where no minimal sharing length exists make it impossible for the unions to benefit from better sharing peers that occur later within the integration process because they cannot satisfy the additional constraints and thus have to reject route sharing.

8 Conclusion and Outlook

In this paper, we have presented a scheduling system that optimizes the usage of the infrastructure in a railroad scenario. We have used a multi-agent approach for planning and plan execution monitoring in this real-time scenario because of the natural link between the theoretical concepts and the scenario. An incremental planning approach that takes incomplete tasks specifications into account uses the contract net protocol to obtained an initial plan. Then, a post-optimization of the initial solution is achieved by means of the simulated trading protocol. The link between the agents planning unit and the external world is drawn by the plan execution monitoring unit of the agent that dynamically reacts to external events and that can initiate a revision of the actual plan of the agent.

The focus of our future work is to extend the concept of location route overlap-

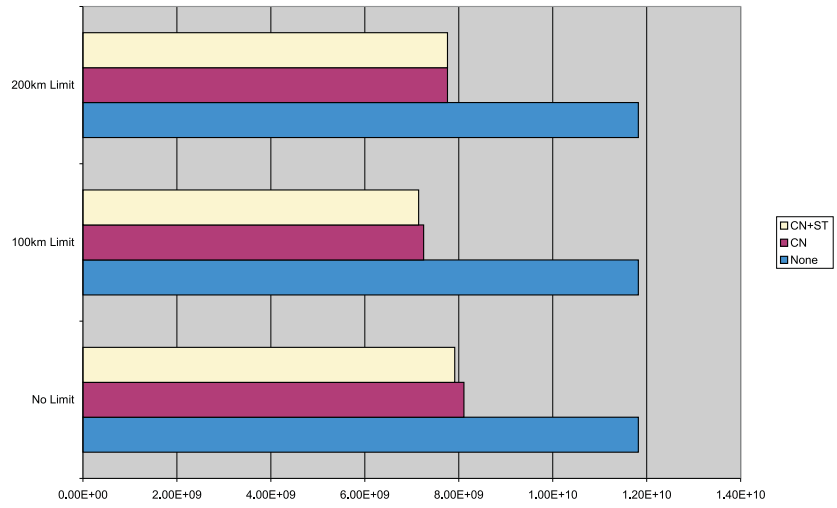

Figure 10: Effects of a minimal sharing length

ping. Currently, the agents seek for exact overlappings in their respective plans. An alternative to this concept is that the agents adapt their plans although their local plans will no longer be the shortest path for their individual tasks. However, the deviation from the shortest path is compensated by the gain obtained by sharing location routes. Optimal solutions to this common-path problem are known to be NP-complete and thus we seek to find powerful heuristics that reduce the computational effort but still yield good solutions. A first idea for such an heuristic is to compute the minimal distance between a new module and an existing union. The distance is defined as the minimum distance between the source and destination node of the new module and any node that is used by any module in the union. The new modules plan is then computed by linking the source and destination node to their nearest neighbours in the union.

As a final remark, we would like to add that the system presented here is not only applicable in the context of freight haulage, it can as well be used in a passenger transport scenario. Even scheduling of road trucks can be of interest when new technologies in road freight haulage that are currently developed by leading car companies become available.

References

[1] BACHEM, A., HOCHSTÄTTLER, W., AND MALICH, M. Simulated Trading: A New Approach For Solving Vehicle Routing Problems. Tech. Rep. 92.125, Mathematisches Institut der Universität zu Köln, Dezember 1992.

[2] BÜRCKERT, H.-J., FISCHER, K., AND VIERKE, G. Transportation scheduling with Holonic MAS, the TeleTruck approach. In *Proc. PAAM98* (1998).

[3] CARROUÉ, L. La ruinease maladie du ≪ tout-routier ≫ — une europe des transports menacée dembolie. *Le Monde Diplomatique* (Dezember 1997), 18–19.

[4] FABEL, P. Increasing the flexibility of freight traffic - using modular train units as an example. In *Proceedings of the World Congress of Railway Research (WCRR)* (Colorado Springs; USA, 1996).

[5] FISCHER, K., KUHN, N., MÜLLER, H. J., MÜLLER, J. P., AND PISCHEL, M. Sophisticated and distributed: The transportation domain. In *Proceedings of MAAMAW-93* (Neuchatel, CH, August 1993), Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World.

[6] FISCHER, K., KUHN, N., AND MÜLLER, J. P. Distributed, knowledge-based, reactive scheduling in the transportation domain. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence and Applications* (San Antonio, Texas, March 1994), pp. 47–53.

[7] FISCHER, K., AND MÜLLER, H. J. Cooperative problem solving in the transportation domain. In *Proceedings of the International Conference on Operations Research (OR'94)* (1995), U. Derigs, Ed., Springer-Verlag.

[8] GASSER, L. Introduction to multi-agent systems. In *Working Notes of Tutorial A at ICMAS95* (San Francisco, CA, June 1995).

[9] KRACKE, R., SIEGMANN, J., VOGES, W., BOECKER, J., AND ZIRKLER, B. Systemgestaltung des Schienengterverkehrs unter Einsatz der Strategie des Train-Coupling and -Sharing. Tech. rep., Universität Hannover, 1995. Studie im Auftrag der DB AG.

[10] KUHN, N., MÜLLER, H. J., AND MÜLLER, J. P. Simulating cooperative transportation companies. In *Simulation als betriebliche Entscheidungshilfe*, J. Biethahn, W. Hummeltenberg, B. Schmidt, and T. Witte, Eds. Vieweg Verlag, Braunschweig/Wiesbaden, 1994, ch. 18, pp. 263–264.

[11] MÜLLER, J. P. *The Design of Intelligent Agents: A Layered Approach*, vol. 1177 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Dec 1996.

[12] O'HARE, G. M. P., AND JENNINGS, N. R., Eds. *Foundations of Distributed Artificial Intelligence*. Wiley & Sons, New York, 1996.

[13] SMITH, R. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* (1980).

[14] SMOLKA, G. The Oz programming model. In *Computer Science Today*, J. van Leeuwen, Ed., Lecture Notes in Computer Science, vol. 1000. Springer-Verlag, Berlin, 1995, pp. 324–343.

[15] VOGES, W., AND MIERAU, U. Train Coupling & -Sharing. In *Proceedings of the World Congress of Railway Research (WCRR)* (Florence; Italy, 1997).

[16] WINDHOFF AG. CargoSprinter, 1996. http://www.windhoff.de http://www.fortunecity.de/anlagen/atlantik/23/sprinter.htm.