Exploiting a Virtual Environment in a Real-World Application

Danny Weyns, Kurt Schelfthout, and Tom Holvoet

AgentWise, DistriNet,
Department of Computer Science, K.U. Leuven,
Celestijnenlaan 200 A, B-3001 Leuven, Belgium
{Danny.Weyns, Kurt.Schelfthout, Tom.Holvoet}@cs.kuleuven.be

Abstract. In situated multi-agent systems (situated MASs), agents are explicitly placed in an environment. A situated agent does not not use long-term planning to decide what action sequence should be executed, but selects actions on the basis of its current position, the world it perceives and limited internal state. Situated agents exploit the environment to coordinate their behavior and to reach a common goal. In a recent project, we applied situated MASs to the control of an automated transportation system that uses automatic guided vehicles (AGVs) to transport loads in a warehouse. In contrast to traditional approaches where the AGVs are controlled by a central server, in this project we model the AGVs as agents in a situated MAS, aiming to improve flexibility and openness. Since the physical environment of AGVs is very restricted, it offers little opportunities for agents to use the environment. We introduce a virtual environment for agents to live in. This virtual environment (1) offers a medium that agents can use to exchange information and coordinate their behavior, and (2) serves as a suitable abstraction to shield low-level physical processing from the AGV agents. Since the only infrastructure available to the AGVs is a wireless network, the virtual environment is necessarily distributed over the AGVs. Synchronization of the state of the virtual environment is provided by ObjectPlaces, a middleware infrastructure that offers support to exchange and share information among nodes in mobile and ad-hoc networks. In this paper, we demonstrate how the environment is used creatively in the design of a MAS solution, helping to manage the complexity of engineering a complex real-world application.

1 Introduction

In the last fifteen years, multi-agent systems (MASs) have been put forward as a paradigm to tackle the increasing complexity of distributed applications. Our research focusses on situated MASs, i.e. MASs in which agents are explicitly placed in an environment. In situated MASs, agents and the environment are first-order abstractions [15]. Situated agents exploit the environment to coordinate their behavior and to reach a common goal. Example mechanisms for environmental coordination are marks [4], gradient fields [6] and digital pheromones [10].

In [16], M. Wooldridge lists benefits of situated MAS including efficiency, robustness and flexibility, but he also points to a number of limitations of situated MASs.

D. Weyns, H. Van Dyke Parunak, and F. Michel (Eds.): E4MAS 2005, LNAI 3830, pp. 218–234, 2006.

Wooldridge argues that situated agents take into account only local, current information and thus inherently must take a "short-time" view for decision making. However, complex problem domains suitable to apply agent-technology, such as ad-hoc networks or manufacturing control, are by their very nature distributed and highly dynamic. In such domains it is questionable whether it is feasible or even useful for agents to collect global information or have a "long-term" view on the situation. Another problem raised by Wooldridge is that there is no methodology to engineer situated agents, in particular with respect to desired overall behavior of the system. The relationship between local interactions of agents on the one hand and global behavior of the MAS on the other hand is indeed a complex open problem in need of extensive further research. An interesting approach is proposed in [3].

Situated MASs have been applied with success in practical applications over a broad range of domains. Some examples are: manufacturing control [9], supply chains systems [12], social simulation [5] and network management [1]. In an ongoing R&D project with Egemin[1], we apply the paradigm of situated MASs to the control of automatic guided vehicles (AGVs) that have to transport loads in a warehouse. In contrast to traditional approaches where the AGVs are controlled by a central server, in this project we model the AGVs as agents in a situated MAS, aiming to improve flexibility and openness. Flexibility refers to a system's capability to adapt its behavior with different environmental situations, and openness enables a system to cope with expansion (new agents that join the system) and reduction (agents that leave the system). Since the physical environment of AGVs is very restricted, it offers little opportunities for agents to use the environment. We introduce a virtual environment for agents to live in. This virtual environment (1) offers a medium that AGV agents can use to exchange information and coordinate their behavior, and (2) serves as a suitable abstraction to shield low-level physical processing from the AGV agents. Since the only infrastructure available to the AGVs is a wireless network, the virtual environment is necessarily distributed over the AGVs. Synchronization of the state of the virtual environment is provided by ObjectPlaces, a middleware infrastructure that offers support to exchange and share information among nodes in mobile and ad-hoc networks. In this paper, we demonstrate how the environment is used creatively in the design of a MAS solution, helping to manage the complexity of engineering a complex real-world application.

This paper is structured as follows. In Sect. 2, we elaborate on situated MASs, and we discuss opportunities that environments offer for situated MASs. Section 3 introduces the AGV application. We discuss the traditional centralized solution briefly, and then explain how we have modelled this application as a situated MAS. In Sect. 4, we zoom in on the virtual environment and illustrate how AGV agents exploit this environment to coordinate their behavior. Finally, in Sect. 5 we draw conclusions.

2 Situated MASs and Environments

2.1 Situated MASs

A situated MAS consists of a (distributed) environment populated with a set of agents that cooperate to solve a complex problem in a decentralized way. Situated agents have

[1] http://www.egemin.com/

local access to the environment, i.e. each agent is placed in a local context which it can perceive and in which it can act and interact with other agents. A situated agent does not use long-term planning to decide what action sequence should be executed, but selects actions on the basis of its current position, the state of the world it perceives and limited internal state. Intelligence in a situated MAS originates from the interactions between the agents, rather than from their individual capabilities.

Situated agents exploit the environment to share information and coordinate their actions. A digital pheromone, for example, is a dynamic structure in the environment that aggregates with additional pheromone that is dropped, diffuses in space and evaporates over time. Agents can use pheromones to dynamically form pheromone paths to locations of interest. Another example is a gradient field that propagates through the environment and changes in strength the further it is propagated. Agents can use a gradient field as a guiding beacon. The environment is thus a crucial part of any situated MAS: both agent and environment are first-order abstractions.

2.2 Opportunities for Exploiting the Environment

Inspired by research and our own experiences with situated MAS, we discuss opportunities that environments offer for MASs [15].

1. Structuring entity: the agents as well as the objects and resources of a MAS are dynamically related to each other. It is the role of the environment to define the rules under which these relationships can exist and can evolve. As such the environment acts as a structuring entity for the MAS. For MASs with an explicit spatial structure, the layout as well as the constraints associated with this layout are part of the environment.
2. Maintenance of shared state: an environment can serve as a robust, self-revising, shared memory for agents. This unburdens the individual agents from continuously keeping track of their knowledge about the system. The state of digital pheromones is an example of shared state that is maintained by the environment.
3. Service support: the environment can provide services for the situated agents to pursue their assigned goals. For example, the environment can provide support to propagate and maintain gradient fields in a distributed environment.
4. Coordination: the environment enables situated agents to coordinate their interactions. Communication required to coordinate can take very different forms: agents can communicate directly via message transfer, or communicate anonymously via a shared space, or communicate indirectly through marks in the environment.
5. Regulating entity: the environment can serve as a means to enforce system-wide constraints (laws) on all agents within a MAS. The environment, e.g., regulates the access to resources. In general, the environment defines the rules for, and enforces the effects of, the agents' actions.

3 The AGV Transportation System

We apply the approach of situated MASs to a real-world application in the domain of automating logistics services in warehouses and manufactories. This application is investigated in a joint R&D research project between the AgentWise research group and

Egemin, a manufacturer of automated warehouse systems. In this section, we first introduce the application and list the required functionalities. We discuss the traditional centralized solution briefly. Next, we discuss new quality requirements for the application and we give a high-level overview of the decentralized solution with a situated MAS. We introduce the virtual environment and illustrate how AGV agents exploit this environment to coordinate their behavior. In the next section, we explain the virtual environment in depth.

3.1 The AGV Application

An AGV transportation system uses unmanned vehicles (AGVs) to transport *loads* through a warehouse. Typical applications are repackaging and distributing incoming goods to various branches, or distributing manufactured products to storage locations. An AGV uses a battery as its energy source. AGVs can move through a warehouse, following a physical path on the factory floor, guided by a laser navigation system, or by magnets or cables that are fixed in the floor. An AGV can pick a load at a certain location and drop it at another location. An AGV can also park at particular locations when it is idle, and charge its battery at a charging station.

Functionalities. The main functionality the system should perform is handling *transports*, i.e. moving loads from one place to another. Transports are generated by *client systems*. Client systems are typically business management programs, but can also be other logistic machines, employees or service operators. A transport is composed out of multiple *jobs*: a job is a simple task that can be assigned to an AGV. For example, picking a load is a pick job, dropping it is a drop job and moving over a specific distance is a move job. A transport typically starts with a pick job, followed by a series of move jobs (probably interrupted by one or more wait jobs) and ends with a drop job.

In order to execute transports, the main functionalities the system has to perform are:

1. Transport assignment: transports are generated by client systems and have to be assigned to AGVs that can execute them.
2. Routing: AGVs must route efficiently through the layout of the warehouse when executing their transports.
3. Gathering traffic information: although the layout of the system is static, the best route for the AGVs in general is dynamic, and depends on the current conditions in the system. For example, some paths may be busy and cause more delay than a longer path that is not busy. Gathering traffic information concerns the monitoring of the current traffic status of the system to adapt the routing of the AGVs to these dynamic conditions.
4. Collision avoidance: obviously, AGVs may not collide. AGVs can not cross the same intersection at the same moment, however, safety measures are also necessary when AGVs pass each other on closely located paths.
5. Deadlock prevention: since AGVs are relatively constrained in their movement (they cannot divert from their path), the system must ensure that at least one of the necessary conditions for deadlock can never hold.

When an AGV is idle it can park at a neighboring park location; when the AGV runs out of energy, it has to charge its battery at one of the charging stations.

3.2 Traditional Approach

Traditionally, vehicles are controlled by one central server, using wireless communication. The server has global knowledge of the system and plans routes for AGVs according to incoming transports and instructs AGVs to perform the jobs. The server continuously polls the AGVs about their status. The low-level control of the AGVs, in terms of actuators, sensors, etc., is handled by the AGV control software called E'nsor[2]. To this end, the layout of the factory is divided into logical elements: *segments* and *nodes*. A logical segment typically corresponds to a physical part of a path of three to five meters. E'nsor is able to steer the AGV per segment, and the AGV can stop on every node, possibly to change direction. E'nsor understands five basic actions:

- Move(segment): instructs E'nsor to drive the AGV over the given segment. Each segment and node is identified by a unique identifier.
- Pick(segment): instructs E'nsor to drive the AGV over the given segment and to pick the load at the end of it.
- Drop(segment): the same as pick, but drops a load the AGV is carrying.
- Park(segment): instructs E'nsor to drive the AGV over the given segment and to park at the end of the segment.
- Charge(segment): instructs E'nsor to drive the AGV over a given segment to a battery charge station and start charging batteries there.

The physical execution of these actions, such as staying on track on a segment, turning, and manipulation of loads are handled by E'nsor. Reading out specific sensor data, such as the current position and the battery level is also provided by E'nsor. When the transport is finished, the server reports the completion of the transport to the corresponding client system.

New Quality Requirements for AGV Transportation Systems. The centralized approach has successfully been applied in numerous practical systems. The main quality properties of the traditional approach are efficiency, configurability and predictability. However, the evolution of the market put forward new requirements for AGV transportation systems.

First, customers request for flexibility of the transportation systems, AGVs should adapt their behavior with changing circumstances. In particular, AGVs should be able to exploit opportunities, e.g., when an AGV is assigned a transport and moves toward the load, it should be possible for this AGV to switch tasks along the way if a more interesting transport pops up. AGVs should also be able to anticipate possible difficulties, e.g., when the battery level of the AGV decreases, the AGV should prefer a zone close to a charge station. Another desired property is that AGVs should be able to cope with particular situations, e.g., when a truck with loads arrives at the factory, the system should be able to reorganize smoothly.

Second, customers expect that the AGV transportation system is open, i.e., the system should be able to deal with leaving AGVs, or new AGVs entering the system. One example is maintenance. Currently, maintenance of AGVs is based on fixed worst-case

[2] E'nsor ® is an acronym for Egemin Navigation System On Robot.

rules. However, the time an AGV should go into maintenance depends on the number of movements (turns, picks, drops, ...) the AGV has executed. Since this information can be locally collected on each AGV, it is more precise (and efficient) to allow each AGV to decide individually whether to go into maintenance or not. AGVs can then leave and enter the system at arbitrary moments. As another example of openness, some customers want to interact with the AGVs on the factory floor, by directly assigning a job to a particular AGV.

In summary, flexibility and openness are high-ranking quality requirements for today's AGV transportation systems.

3.3 A Decentralized Solution with a Situated MAS

The general idea of the decentralized approach is to put more autonomy in the AGVs allowing for improved flexibility and openness. In the decentralized solution, vehicles become autonomous agents which make decisions based on their current situation, and who coordinate with other agents to ensure the system as a whole processes transports.

Decentralized control of automated warehouse transportation systems is an active area of research. In [7], Ong gives an extensive overview of decentralized agent-based manufacturing control and compares the pros and cons of centralized versus decentralized control. According to Ong, the advantages of decentralized control are: (1) it is more economical w.r.t. required processing power, and (2) it is more reliable. Disadvantages of decentralization are: (1) performance of the system may be affected by the communication links between nodes, (2) while the distributed approach is designed to cope with disturbances, there is, in general, a trade-off between its performance and the reactivity of the system to disturbances, and (3) myopic decision may occur due to the lack of global information. Examples of other recent decentralized approaches are [8] that discusses a decentralized cognitive planning approach for collision-free movements of vehicles, and [2] that discusses a behavior-based approach for decentralized control of automatic guided vehicles. However, both approaches are validated only in simulations under a number of simplifying assumptions. In general, applications of decentralized control of automated transportation systems in real industrial settings are rarely discussed in literature.

Besides the advantages of decentralization listed by Ong, we believe that in principle, a MAS-based AGV transportation system also becomes more flexible. Since each AGV acts locally, it can better exploit opportunities and adapt its behavior under changing circumstances. On the other hand, the benefits of a decentralized approach do not come for free. Since an all knowing entity in the system does not exist, inter-AGV coordination becomes complex. Bandwidth must be considered carefully to ensure that the communication network does not become a bottleneck. Another important consequence of decentralization, not mentioned by Ong, is an increased complexity of debugging.

The general challenge in the project is to support the current functionality, while aiming to improve flexibility and openness, and keeping in mind the benefits of the centralized approach. So far, we have implemented AGV routing, information sharing and collision avoidance. We have validated the solution in a test setup with two physical AGVs, and in a number of advanced simulation cases with up to six AGVs. Many challenges lie ahead. Currently, we are developing architectural models to cope with

order assignment and deadlock prevention. Only when these models are implemented and tested, we can start the validation of the integral solution in an advanced setup.

High-Level Model of the Situated MAS. The situated MAS consists of two kinds of agents, *transport agents* and *AGV agents*. Transport agents are located at *transport bases*. AGV agents are located in AGVs that are situated on the factory floor. Figure 1 depicts a high-level model of the situated MAS with one transport base and two AGVs. A transport agent represents a transport that needs to be handled by an AGV. AGV

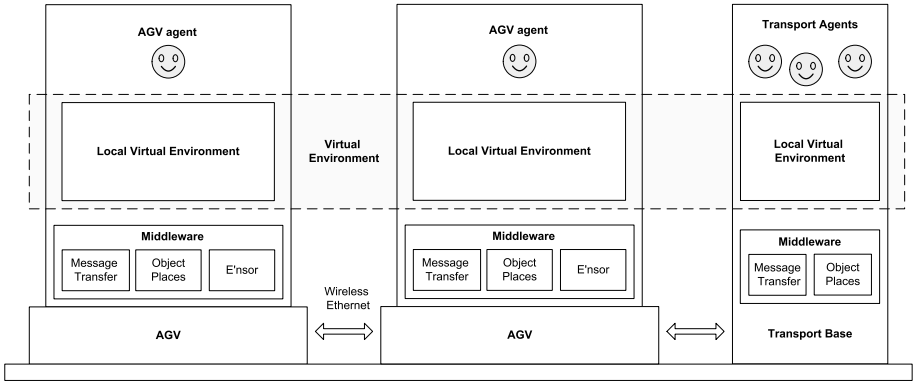

Fig. 1. High-level model of the AGV transportation system

agents are responsible for executing the assigned transports. We fully reused the E'nsor software that deals with the low-level control of the AGVs. As such, the AGV agents control the movement and actions of AGVs on a fairly high level. The communication infrastructure provides a wired network that connects client systems and transport bases, and a wireless network that enables mobile AGVs to communicate with each other and with transport agents on transport bases.

AGVs are situated in a physical environment, however, this environment is very constrained: AGVs cannot manipulate the environment, except by picking and dropping loads. This restricts how AGV agents can exploit their environment. We introduce a virtual environment for agents to live in. This virtual environment offers a medium that agents can use to exchange information and coordinate their behavior. Besides, the virtual environment serves as a suitable abstraction that shields the AGV agents from low-level issues, such as the physical control of the AGV.

In the AGV application, the only physical infrastructure available to the AGVs is a wireless network for communication. In other words, the virtual environment is necessarily distributed over the AGVs and transport bases. In effect, each AGV and each transport base maintains a *local virtual environment*, which is a local manifestation of the virtual environment. Local virtual environments are merged with other local virtual environments opportunistically, as the need arises. In other words, *the* virtual environment as a software entity does not exist; rather, there are as many local virtual environments as there are AGVs and transport bases. Some of these local virtual environments

may recently be synchronized with each other, while others may not. From the agent perspective, the virtual environment appears as one entity. The synchronization of the state of neighboring local virtual environments is supported by the ObjectPlaces middleware [11]. We elaborate on state management in the virtual environment in Sect. 4.

Responsibilities of Agents and the Environment. To describe how we apply a situated MAS to control an AGV system, we revisit the five core functionalities of the AGV application described in Sect. 3.1. We describe the main responsibilities of the two types of agents in the MAS, as well as the responsibilities of the virtual environment.

Transport Assignment. As stated above, transports are represented by transport agents that reside on transport bases. Transport bases receive transports from client systems. For each new transport, a new transport agent is created that is responsible to assign the transport to an AGV and to ensure that the transport is completed correctly. Each transport has a priority that depends on the kind of transport, the pending time since its creation, and the nature of other transports in the system. Therefore, transport agents interact with other related transport agents to determine the correct priority over time. Transport agents use the virtual environment to find AGV agents to assign the transports, and to follow the progress of the assigned transports. To assign a transport, the transport agent negotiates with AGV agents of AGVs near to the pick location of the load. Once the transport is assigned, the awarded AGV handles the transport. As soon as the transport is completed, the AGV agent informs the transport agent, that in its turn informs the client system after which the transport agent is removed. The transport agent guarantees the persistence of the transport in the system. If for some reason the assigned AGV is unable to complete the transport, the transport agent may negotiate with other AGVs to reassign the order.

Routing. For routing purposes, the virtual environment has a static map of the paths through the warehouse. This graph-like map corresponds to the layout used by E'nsor. To allow agents to find their way through the warehouse efficiently, the virtual environment provides signs on the map that the agents use to find their way to a given destination. These signs can be compared to traffic signs by the road that provide directions to drivers. At each node in the map, a sign in the virtual environment represents the cost to a given destination for each outgoing segment. The cost of the path is the sum of the static costs of the segments in the path. The cost per segment is based on the average time it takes for an AGV to drive over the segment. The agent perceives the signs in its environment, and uses them to determine which segment it will take next.

Gathering Traffic Information. Besides the static routing cost associated with each segment, the cost is also dependent on dynamic factors, such as congestion of a segment. To warn other agents that certain paths are blocked or have a long waiting time, agents mark segments with a dynamic cost on a *traffic map* in the virtual environment. Agents mark the traffic map by dropping pheromones on the applicable segments. When AGVs come in each others neighborhood, the information of the traffic maps is exchanged and merged to provide up-to-date information to the AGV agents. Since pheromones evaporate over time, outdated information automatically vanishes over time. AGV agents take

the information on the traffic map into account when they decide how to drive through the warehouse.

Collision Avoidance. AGV agents avoid collisions by coordinating with other agents through the virtual environment. AGV agents mark the path they are going to drive in their environment using *hulls*. The hull of an AGV is the physical area the AGV occupies. A series of hulls then describes the physical area an AGV occupies along a certain path. If the area is not marked by other hulls (the AGV's own hulls do not intersect with others), the AGV can move along and actually drive over the reserved path. Afterwards, the AGV removes the markings in the virtual environment. We zoom in on collision avoidance in Sect. 4.4.

Deadlock Prevention. The basic mechanisms for deadlock prevention provided in the traditional approach can be adopted in the MAS approach. E.g., when an AGV approaches a bidirectional path in the layout, the AGV agent can lock that path via the hull reservation mechanism, or when an AGV reaches an entry point of a critical area where only a limited number of AGVs are allowed, the AGV agent can instruct the AGV to wait there until the area is accessible. However, those rules only provide a partial solution to avoid deadlock. Currently, we study two additional tracks to deal with deadlock, one with a supervising MAS that monitors the AGV movements and provides feedback to the AGV agents, and another where AGVs themselves monitor their neighborhood and exchange information regarding deadlock threats via the environment.

4 A Virtual Environment for AGV Agents

This section describes the virtual environment in the AGV transportation application. We focus on the virtual environment from the viewpoint of AGV agents. First we give a broad overview of the structure of the virtual environment, in three parts. The first part gives a brief overview of the high-level model of an AGV and situates the virtual environment in this model. The second part describes how the local virtual environment synchronizes its state with other local virtual environments. The third part describes how the virtual environment handles perception, action and communication. Concluding with an example, we describe how the virtual environment is exploited by the AGV agents to avoid collisions.

4.1 High-Level Model of an AGV

Figure 2 depicts an overview of an AGV. The AGV agent is shown in the top layer of the model. We do not elaborate on the architecture of the AGV agent; it is based on the reference architecture discussed in [13] and [14]. The AGV agent is situated in the virtual environment, shown as a layer below the top layer. The virtual environment uses the middleware layer, that is composed of a Message Transfer System, the ObjectPlaces middleware [11] (both discussed later) and E'nsor. The operating system is located below the middleware. Finally, the bottom layer represents the physical infrastructure of the AGV, including a processor, communication infrastructure, actuators and sensors. We further elaborate on the virtual environment and the supporting middleware hereafter.

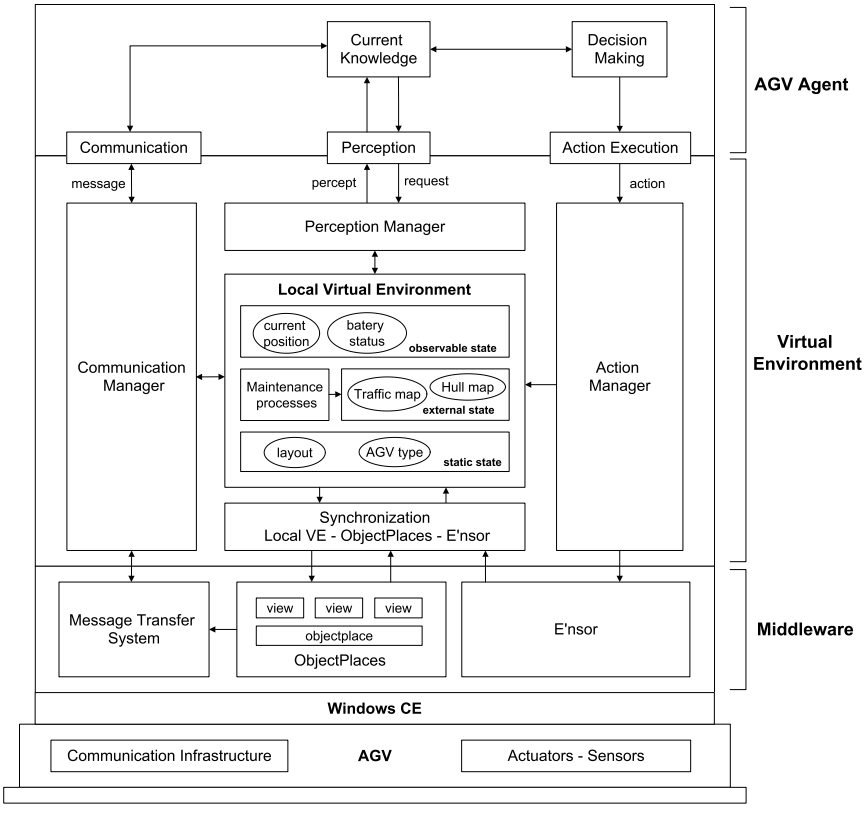

Fig. 2. High-level model of an AGV

4.2 Managing State in the Virtual Environment

Since the virtual environment is necessarily distributed over the AGVs and transport bases, each local virtual environment is responsible to keep its state synchronized with other local virtual environments. The state of the local virtual environment on an AGV is divided into three categories:

1. Static state: this is state that does not change over time. A typical example is the layout of the factory floor, which is needed for the AGV to navigate. Static state must never be exchanged between local virtual environments, since it is common knowledge and never changes.
2. Observable state: this is state that can be changed in one local virtual environment, while other local virtual environments can only observe the state. An AGV typically obtains this kind of state from its sensors directly. An example is an AGV's position. Local virtual environments are able to observe another AGV's position, but only the local virtual environment on the AGV itself is able to read it from its sensor, and change the representation of the position in the local virtual environment. No

conflict ever arises between two local virtual environments concerning the update of observable state.

3. Shared state: this is state that can be modified in two local virtual environments concurrently. So, two or more local virtual environments can conflict on what is the "right" state. The traffic map, containing dynamic costs associated with segments, is an example of shared state. Several AGV agents can modify the cost on the same segment concurrently. When the local virtual environments on these AGVs synchronize, costs of the local virtual environments' traffic maps are mutually exchanged and conflicts are resolved to generate an up-to-date traffic map in both local virtual environments.

In order to manage and maintain this state, the local virtual environment performs three basic activities. We describe each of these in turn.

The first activity is synchronizing the state of the local virtual environment with the AGV's sensors. The local virtual environment uses E'nsor to regularly poll the vehicles's current status and adjust its own state appropriately. For example, if the AGV's position has changed, the AGV position in the local virtual environment is updated.

The second activity the virtual environment performs is synchronizing the state of the local virtual environment with other AGVs. This is supported by the ObjectPlaces middleware. ObjectPlaces offers high-level abstractions to deal with communication in mobile and ad hoc networks. The local virtual environment uses the middleware by sharing objects in a tuplespace-like container, called an *objectplace*. Every AGV has one objectplace locally available. Objects in objectplaces on remote AGVs can be gathered using a *view*. The local virtual environment can define a view by (1) specifying which AGVs' objectplaces need to be included in the view (e.g. the objectplaces of all AGVs within a specific range), and (2) specifying what objects need to be included in the view (e.g. hull objects). Based on these specifications, the ObjectPlaces middleware then builds a local collection of objects reflecting the current contents of the remote objectplaces. In other words, a local virtual environment shares data with other AGVs by putting objects in the local virtual environment's objectplace. Local virtual environments gather data from other AGVs by defining a view on the objectplaces of those AGVs. For example, when an AGV agent marks a hull in the environment, this hull is published in the local virtual environment's objectplace. When the AGV agent wants to perceive hulls in its vicinity, the local virtual environment defines a view on all hull objects in objectplaces of AGVs within a certain physical distance from the AGV. The middleware then gathers the hull objects from the objectplaces on the appropriate AGVs. The local virtual environment can then use the hull objects to determine whether the requested path is free or not and return this results to the agent.

The third and last activity is maintaining the state of the virtual environment locally. This is done by *maintenance processes* in the local virtual environment itself. An example is the maintenance of pheromones. A change of local state possibly triggers an update of state in the local virtual environment's objectplaces, so that other virtual environments can synchronize with the new state.

In summary, the virtual environment deals with the management of state in the distributed system, hiding many aspects of distribution from the AGV agent. Agents can

act and perceive in the local virtual environment, which in turn contacts local environments on other AGVs to synchronize state.

4.3 Perceiving, Acting and Communicating

The virtual environment offers abilities for perception, action and communication to the AGV agent, shielding low-level details from the agent.

Perception is handled by the *perception manager*. The perception manager's task is straightforward: when the agent requests a percept, for example the current positions of neighboring AGVs, the perception manager queries the necessary information from the local virtual environment and returns it to the agent.

Actions are handled by the *action manager*. A first kind of actions concerns the physical actions of the AGV, for example moving over a segment or picking a load. These actions are handled fairly easily by passing them on to the E'nsor control software. A second kind of actions does not actually have an effect on the behavior of the AGV, but manipulates the virtual environment. Marking hulls is one example of this, which is described in detail in Sect. 4.4. Another is dropping a virtual pheromone. In general, an action can be handled by passing it down to Ensor, and/or by changing the local virtual environment which in turn may change the content of the objectplace.

Communication is handled by the *communication manager*. Agents can communicate directly with other agents through the virtual environment. A typical example is an AGV agent that communicates with a transport agent. Another example is an AGV agent that requests the AGV agent of a waiting AGV to move out of the way. The virtual environment is responsible for translating high level messages to messages that can be sent through the network (resolving agent names to IP numbers for example). For this, it uses the *message transfer system* in the middleware layer.

In summary, the virtual environment offers high level primitives to the AGV agent to act in the world, perceive the world, and communicate with other agents. The virtual environment shields the agent from having to deal with lower level issues.

4.4 A Specific Scenario: Collision Avoidance

We now describe a specific scenario, to illustrate how collision avoidance works. In the centralized approach, collision avoidance is realized as follows: for each AGV in the system, a series of *hulls* are calculated along the path each AGV is going to drive. When two or more such *hull projections* overlap, AGVs are on a collision course and all except one AGV are commanded to wait.

In a decentralized architecture, a central arbitrator does not exist. However, the virtual environment enables the agents to act as if they are situated in a shared environment, while the virtual environment takes on the burden of coordination. Figure 3 shows a series of screenshots of a simulation run in a realistic map. In Fig. 3(a), two AGVs are approaching one another. We call the AGV approaching from the top AGV A, and the other AGV B. Both are projecting hulls in the environment. At this point, no conflict is

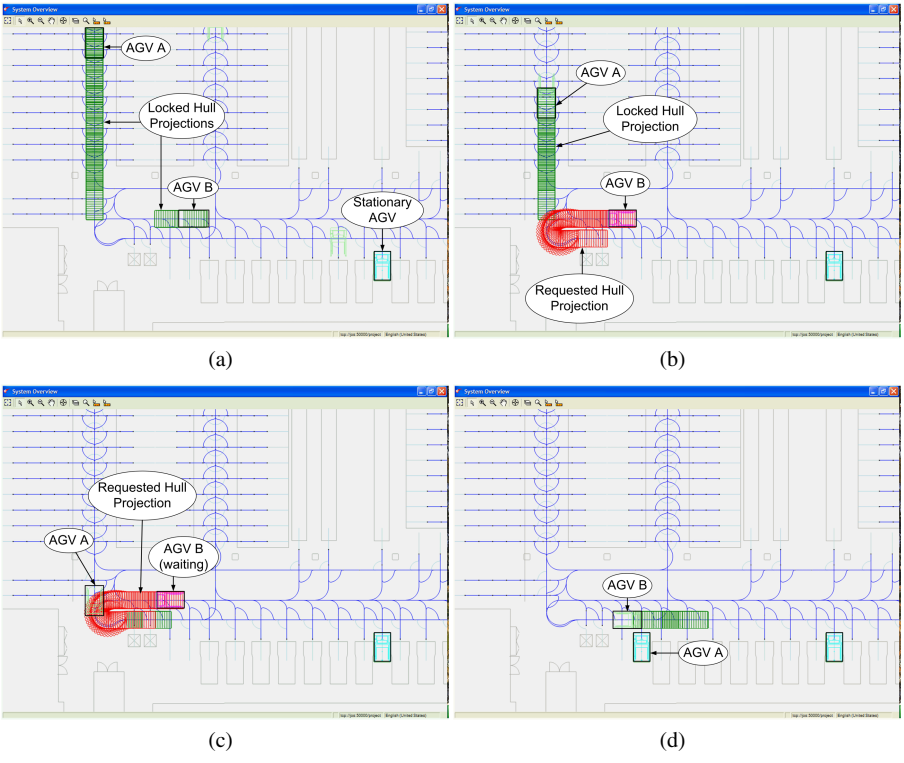

Fig. 3. (a) Two AGVs approaching, (b) A conflict is detected, (c) One AGV passes safely, (d) The second AGV can pass as well

detected. In Fig. 3(b), AGV B has projected further ahead, and is now in conflict with the hull projection of AGV A. However, since AGV A's hull projection was already locked, AGV B must wait. In Fig. 3(c), AGV A is taking the curve, passing AGV B. Finally, in Fig. 3(d), AGV A has parked at the bottom, and AGV B can start moving.

We now describe the collision avoidance mechanism in more detail. First, we focus on how the agent avoids collision without being aware of the actual underlying collision avoidance protocol, then we study the work behind the scenes (i.e. the protocol) in the virtual environment.

In order to drive, the agent takes the following actions:

1. The agent determines the path it intends to follow over the layout. The agent determines how much of this path it wants to lock. This is determined by a safe stopping distance on the one hand, and the application of basic rules for deadlock avoidance on the other hand. As an example of the latter, if the AGV tries to lock a bi-directional path, it must lock that path until the end, since otherwise another AGV might enter it from the other direction, leading directly to a deadlock situation.
2. The agent marks the path it intends to drive with a *requested hull projection*. This projection contains the agents id and a priority, that depends on the current transport the AGV is handling.

3. The agent perceives the environment to observe the result of its action.
4. The agent examines the perceived result. There are two possibilities:
   (a) The hull is marked as "locked" in the environment; it is safe to drive.
   (b) The hull is not marked as locked. This means that the agent's hull projection conflicted with that of another agent. The agent may not pass; at this point the agent may decide to wait and look again at a later time, or remove its hull projection and take another path altogether.

The virtual environment plays an important role in this coordination approach: it must make sure that a hull projection becomes locked eventually. To this end, the local virtual environment of the AGV agent that requests a new hull projection, executes a collision avoidance protocol with local virtual environments of nearby AGVs.

It is desirable to make the set of nearby AGVs not larger than necessary, since it is not scalable to interact with every AGV in the system. On the other hand, the set must include all AGVs with which a collision is possible: safety must be guaranteed.

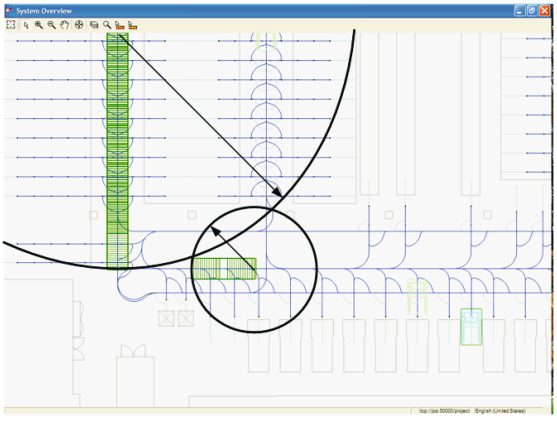

Fig. 4. Determining nearby AGVs

A solution to this problem is shown in Fig. 4. The local virtual environment of a *requesting AGV* will interact with the local virtual environments of other AGVs whose *hull projection circle* overlaps with the hull projection of the requesting AGV. The hull projection circle is defined by a center point, which is the position of the AGV itself, and a radius, which is equal to the distance from the AGV to the furthest point on its hull projection. If two such circles overlap, this indicates (to a first approximation) that the two AGVs might collide. We call the set of AGVs with overlapping hull projection circles the *requested AGVs*.

The local virtual environment of the requesting AGV executes the following protocol with the local virtual environment's of requested AGVs. The protocol is a variant on well-known mutual exclusion protocols based on voting.

1. The local virtual environment of the requesting AGV sends the requested hull projection to the local environments of all requested AGVs.

2. The local environments of requested AGVs check whether the projection overlaps with their hull projection. There are three possibilities for each of the requested AGVs:
   (a) No hull projections overlap. The local virtual environment of the requested AGV sends an "allowed" message to the local virtual environment of the requesting AGV.
   (b) The requesting AGV's hull projection overlaps with the requested AGV's hull projection, and the requested AGV's hull is already locked. The local virtual environment of the requested AGV sends a "forbidden" message to the local virtual environment of the requesting AGV.
   (c) The requesting AGV's hull projection overlaps with the requested AGV's hull projection, and the requested AGV's hull is not locked. Since each of the requested hulls contains a priority, the local virtual environment of the requested AGV can check which hull projection has precedence. If the hull projection of the requesting AGV has a higher priority than that of the requested AGV, the local virtual environment of the requested AGV replies "allowed"; it replies "forbidden" otherwise.
3. The local virtual environment of the requesting AGV waits for all "votes" to come in. If all local virtual environments of the requested AGVs have voted "allowed", the hull projection can be locked and the local virtual environment is updated. If not, the local virtual environment of the requesting AGV waits a random amount of time and then tries again from step 1.

If at any time, the agent removes the requested hull from the virtual environment, the protocol is aborted.

In this scenario, the virtual environment serves as a flexible coordination medium, which hides much of the distribution of the system from the agents: agents coordinate by putting marks in the environment, and observing marks from other agents.

5 Conclusions

Situated agents exploit the environment to coordinate their behavior and reach their goals. Research in this area almost invariably assumes the existence of an exploitable environment a priori that is accessible for all agents, either by centralizing or by providing infrastructural support especially for environments. A possible critique on this research is that it takes the access to the environment as a common shared entity for granted, whereas the absence of such an entity is the essence of many multi-agent based systems. On the contrary, we have shown that an environment does not need to be a common shared entity to be useful. We introduced the concept of virtual environment, a *decentralized* entity in an application where a centralized approach is undesirable and no shared infrastructure is available to deploy an environment as a common accessible entity. We also showed that offering an observable and moldable environment to agents living in a constrained physical environment, strengthens the MAS approach instead of diluting it. The virtual environment is a first-order abstraction in the designed solution of the AGV transportation application.

So far, we have implemented AGV routing, information sharing and collision avoidance. We have validated the solution in a test setup with two physical AGVs, and in a number of advanced simulation cases. The next challenges are order assignment and deadlock avoidance. With respect to order assignment we study two different tracks, one with an adaptive version of the Contract-Net protocol and one with a gradient field based approach. In this latter approach, each transport agent emits a gradient field in the virtual environment that attracts interested AGVs to the pick location of the load, while each interested AGV emits a gradient field that repels other competitor AGVs. The gradient fields guide idle AGVs toward the most appropriate transports, ensuring maximal flexibility (e.g., AGVs take into account opportunities –new transports that pop up– when they drive toward a load). To deal with deadlock, we also follow two possible approaches, one with a supervising MAS that monitors the AGV movements and provides feedback to the AGV agents, and another where AGVs themselves monitor their neighborhood and exchange information regarding deadlock threats via the virtual environment.

We are convinced that exploiting the environment in our ongoing AGV research case is an asset, and will continue our validation of situated MAS in this complex real-world application.

References

1. Bonabeau, E., Henaux, F., Guérin, S., Snyers, D., Kuntz P., Theraulaz, G.: Routing in Telecommunications Networks with "Smart" Ant-Like Agents. Intelligent Agents for Telecommunications Applications (1998)
2. Berman, S., Edan, Y., Jamshidi, M.: Decentralized autonomous Automatic Guided Vehicles in material handling. Transactions on Robotics and Automation 19(4) (2003)
3. De Wolf, T., Samaey, G., Holvoet, T., Roose, D.: Decentralised autonomic computing: Analysing self-organising emergent behaviour using advanced numerical methods. 2th International Conference on Autonomic Computing. IEEE (2005)
4. Ferber, J.: Multiagent Systems: An Introduction to Distributed Artificail Intelligence. Addison-Wesley (1999) 439-445
5. Macy, M., Willer, R.: From Factors to Actors: Computational Sociology and Agent-Based Modeling. Annual Review of Sociology 28 (2002)
6. Mamei, M., Zambonelli, F.: Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. International Conference on Pervasive Computing and Communication (2004)
7. Ong, L.: An investigation of an agent-based scheduling in decentralised manufacturing control. Ph.D Disseration University of Cambridge (2003)
8. Pallottino, L., Scordio, V.G., Frazzoli, E., Bicchi, A.: Decentralized cooperative conflict resolution for multiple nonholonomic vehicles. AIAA Conference on Guidance, Navigation and Control (2005)
9. Parunak, H.V.D., Baker, A.D., Clark, S.J.: The AARIA Agent Architecture: From Manufacturing Requirements to Agent-Based System Design. Workshop on Agent-Based Manufacturing (1998)
10. Parunak, H.V.D., Brueckner, S., Sauter, J.: Digital Pheromones ofr Coordination of Unmanned Vehicles. Post-proceedings of the First International Workshop on Environments for Multiagent Systems, Lecture Notes in Computer Science Series, Vol. 3374 (2005)

11. Schelfthout, K., Holvoet, T., Berbers, Y.: Views: Customizable abstractions for context-aware applications in MANETs. Proceedings of the fourth international workshop on Software engineering for large-scale multi-agent systems, St. Louis, USA. ACM Press (2005).
12. Sauter, J.A., Parunak, H.V.D.: ANTS in the Supply Chain. Workshop on Agent based Decision Support for Managing the Internet-Enabled Supply Chain, Seattle, WA (1999)
13. Weyns, D., Holvoet, T.: A Formal Model for Situated Multi-agent Systems. Formal Approaches for Multi-Agent Systems, R. Verbrugge and B. Dunin-Keplicz Eds., Special Issue of Fundamenta Informaticae 63(2-3) (2004)
14. Weyns, D., Steegmans, E., Holvoet, T.: Protocol Based Communication for Situated Multi-agent Systems. 3th International Joint Conference on Autonomous Agents and Multi-Agent Systems, New York (2004)
15. Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T., Ferber, J.: Environments for Multi-agent Systems, State-of-the-art and Research Challenges. First International Workshop on Environments for Multiagent Systems, Lecture Notes in Computer Science Series, Vol. 3374 (2005)
16. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley and Sons, UK (2002)