Decentralized Control of E'GV Transportation Systems

Danny Weyns, Kurt Schelfthout,
Tom Holvoet
AgentWise, DistriNet, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
<first name.name>@cs.kuleuven.ac.be

Tom Lefever
Egemin N.V.
Baarbeek 1, B-2070 Zwijndrecht, Belgium
tom.lefever@egemin.be

ABSTRACT

Egemin N.V. is a Belgian manufacturer of Automatic Guided Vehicles –named E'GVs– and control software for automating logistics services in warehouses and manufactories using E'GVs. In a joint R&D project, Egemin and the AgentWise research group are developing an innovative version of the E'GVs control system aimed to cope with new and future system requirements such as flexibility and openness.

In this project, we exploit principles and mechanisms known from situated multi-agent systems for modelling and implementing a *decentralized control system*. Instead of a centralistic approach, where one computer system is in charge of numerous complex and time-consuming tasks (such as routing, collision avoidance, deadlock avoidance, etc.), we aim to provide the E'GVs with a considerable amount of autonomy. This allows to obtain a system that is far more flexible than the current software – the E'GVs adapt themselves to the current situation in their direct vicinity, order assignment is dynamic, the system can cope with E'GVs leaving the system (e.g. for maintenance) or adding new E'GVs, and so on.

In this paper, we describe the architecture that is employed for implementing the control of the E'GV system. The implementation of the control software seamlessly integrates with E'nsor, the low-level vehicle control system. We illustrate a concrete control scenario in which two E'GVs coordinate their behavior to avoid collisions.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design

Keywords

Automatic guided vehicles, situated multiagent systems

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

1. INTRODUCTION

In March 2004 the AgentWise research group started a joint R&D project with Egemin, a Belgian manufacturer of automated logistic service systems for warehouses and manufactories. Egemin builds automatic guided vehicles, called E'GVs, and deploys and maintains complete installations of multiple E'GVs (and other logistic machines) for their clients, see Fig. 1. This project, called EMC^2 (Egemin Modular Control Concepts), envisions a radical redesign of the current architecture of the control software for E'GV installations. Traditionally, E'GVs in a factory are directly

Figure 1: An E'GV at work in a cheese factory.

controlled by a central server. E'GVs have no autonomy: the server plans the schedule for the system as a whole, dispatches commands to the E'GVs and continually polls their status. This system architecture has successfully been deployed in numerous practical installations. The centralized server architecture has two main benefits. The control software can be customized easily to the needs of a particular project, since the server is a central configuration point. This allows for specific per-project optimizations. A second benefit is that the system is deterministic and predictable.

However, the evolution of the market put forward new requirements for E'GV transportation systems. Customers request for flexibility of the transportation systems, E'GVs should adapt their behavior with changing circumstances. E'GVs should be able to exploit opportunities, anticipate possible difficulties, and cope with particular situations. The system should also be able to deal with E'GVs leaving the

system, e.g. for maintenance, or new E'GVs entering the system.

In the EMC^2 project, we are investigating the feasibility of a decentralized architecture, using concepts from *situated multi-agent systems* (situated MAS). In this approach, E'GVs are autonomous agents that cooperate to ensure the functionality of the system.

Decentralized control of automated warehouse transportation systems is an active area of research. In [3], 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 [4] that discusses a decentralized cognitive planning approach for collision-free movements of vehicles, and [1] 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 E'GV transportation system also becomes more flexible. Since each E'GV 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-E'GV 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 EMC^2 project is to support the current functionality, while aiming to improve flexibility, and keeping in mind the benefits of the centralized approach.

At this point in the project, we have built a detailed agent architecture, describing the various entities in our system. An interesting aspect of this architecture is the virtual environment: many techniques and mechanisms in situated MAS rely on a shared environment. However, in this project the E'GVs do not have access to such a shared environment. To solve this problem, we introduced a virtual environment, which the E'GV agents can use to exchange information and coordinate their behavior.

Eight months after the start of the project, we achieved our first milestone, allowing one real E'GV to drive around in an experimental setup with an agent at the wheel. Recently, we added collision avoidance, allowing two E'GVs to drive around in an industrial test setup, and allowing any number of E'GVs to drive around in a simulated test setup. Many challenges lie ahead, of which the most important are deadlock avoidance and order assignment. This paper presents the first promising results of our approach in this challenging real-world domain.

This paper is structured as follows. In section 2, we introduce the E'GV application. We give a brief overview of the traditional approach and point to new quality requirements. Section 3 introduces a new decentralized approach that models an E'GV transportation system as a situated MAS. In section 4, we zoom in on the software architecture of the E'GV transportation systems, and illustrate a concrete scenario in which two E'GVs coordinate their behavior to avoid collisions. Finally, in section 5 we draw conclusions.

2. E'GV TRANSPORTATION SYSTEM

In this section, we introduce the E'GV application and list the required functionalities. We give a high-level overview of the traditional centralized solution and discuss new quality requirements for E'GV transportation systems.

2.1 The E'GV Application

An E'GV transportation system uses unmanned vehicles that are custom made to be able to transport various kinds of *loads*, from basic or raw materials to completed products. Typical applications are repackaging and distributing incoming goods to various branches, or distributing manufactured products to storage locations. An E'GV uses a battery as its energy source. E'GVs can move through a warehouse guided by a laser navigation system, or following a physical path on the factory floor that is marked by magnets or cables that are fixed in the floor.

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 particular 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 E'GV. For example, picking up 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 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 E'GVs that can execute them.

2. Routing: E'GVs 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 E'GVs in general is dynamic, and depends on the current conditions in the system. Gathering traffic information concerns the monitoring of the current traffic status of the system to adapt the routing of the E'GVs to these dynamic conditions.

4. Collision avoidance: obviously, E'GVs may not collide. E'GVs can not cross the same intersection at the same moment, however, safety measures are also necessary when E'GVs pass each other on closely located paths.

5. Deadlock avoidance: since E'GVs are relatively constrained in their movement (they cannot divert from their path), the system must ensure that E'GVs do not find themselves in a deadlock situation.

When an E'GV is idle it can park at a free park location; however, when the E'GV runs out of energy, it has to charge its battery at one of the charging stations.

2.2 Traditional Centralistic Approach

Traditionally, vehicles are controlled by one central server, using wireless communication. The server receives transport requests from the client systems. According to the incoming transports, the server plans routes for E'GVs and instructs E'GVs to perform the jobs. The server continuously polls the E'GVs about their status. The low-level control of the E'GVs, in terms of sensors and actuators (staying on track, turning, determining the current position, reading out the battery level, etc.) is handled by the E'GV control software called E'nsor[1]. To this end, the layout of the factory is divided into logical elements: *segments* and *nodes*. Each segment and node is identified by a unique identifier. A logical segment typically corresponds to a physical part of a path of three to five meters. E'nsor is able to steer the E'GV per segment, and the E'GV can stop on every node, possibly to change direction. E'nsor understands five basic actions: (1) Move(segment): this instructs E'nsor to drive the E'GV over the given segment; (2) Pick(segment): instructs E'nsor to drive the E'GV over the given segment and pick up a load at the end of it; (3) Drop(segment): the same as pick, but drops a load the E'GV is carrying; (4) Park(segment): instructs E'nsor to drive the E'GV over the given segment and park at the park location at end of the segment; (5) Charge(segment): instructs E'nsor to drive the E'GV over a given segment to a battery charging station and start charging batteries there. When a transport is finished, the server reports the completion of the transport to the corresponding client system.

2.3 New quality requirements

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 E'GV transportation systems.

Customers request for flexibility of the transportation systems, E'GVs should adapt their behavior with changing circumstances. In the traditional approach, the assignment of transports, the routing of E'GVs and the control of traffic are all planned by the central server. Although this planning is efficient, is lacks flexibility. Most of the planning is based on rigid predefined schedules. Schedules are rules associated with E'GVs, segments or nodes, e.g. "if an E'GV a has dropped a load on node x, then that E'GV has to move to node y to wait for a transport assignment, however if node y is already occupied by another E'GV b, then E'GV a has to move to node z to park." A plan can be changed, however only under exceptional conditions, e.g. when an E'GV becomes defective on the way to a load, the transport can be reassigned to another E'GV. Customers have various requests with respect to flexibility. E'GVs should be able to exploit opportunities, e.g., when an E'GV is assigned a transport and moves toward the load, is should be possible for this E'GV to switch tasks along the way if a more interesting transport pops up. E'GVs should also be able to anticipate possible difficulties, e.g., when the battery level of an E'GV decreases, the E'GV should anticipate this and prefer a zone near to a charge station. Another desired property is that E'GVs 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 reorganized itself smoothly. Finally, customers expect that the system is able to deal with E'GVs leaving the system, or new E'GVs entering the system. One example is maintenance. Currently, maintenance of E'GVs is based on fixed worst-case rules. This leaves room for improvement by allowing E'GVs to decide themselves when service is necessary. E'GVs can then leave or enter the system arbitrary. Note that self-monitoring the physical state of the machine is out the scope of this work. As another example, customers expect to be able to intervene during the execution of the system. In particular situations, customers expect to be able to "use" an E'GV to perform a specific job. In summary, flexibility are openness are high-ranking quality requirements for today E'GV transportation systems.

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

3. A DECENTRALIZED APPROACH

The general idea of the decentralized approach is to put more autonomy in the E'GVs allowing for more flexibility. 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 the transports.

In this section we start with a brief introduction of situated MASs. Then we discuss the decentralized solution for E'GV transportation systems based on a situated MAS. We give a overview of the decentralized system architecture and discuss the responsibilities of the agents and the environment in the system. In the following section, we zoom in on the software architecture of the E'GV agents and the environment.

3.1 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 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.

In situated MASs, agents and the environment are first-order abstractions [11]. 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. Situated MASs have been applied with success in practical applications over a broad range of domains. Examples are manufacturing control [5], supply chains systems [6], and network management [2].

Cooperating agents situated in an environment is a natural concept to manage complexity in a decentralized man-

ner. Agents encapsulate their own behavior and are able to adapt to changes in their environment. Well known benefits of situated MAS are efficiency, robustness and flexibility [13]. These fundamental properties make situated MASs a suitable approach for building self-managing applications.

3.2 Decentralized Architecture

Since E'GVs are situated in an explicit environment and interaction is at the core of an E'GV transportation system (load manipulation, collision avoidance, etc.) an E'GV transportation system maps naturally to a situated MAS. Fig. 2 depicts the system architecture of the decentralized E'GV transportation system. The situated MAS consists

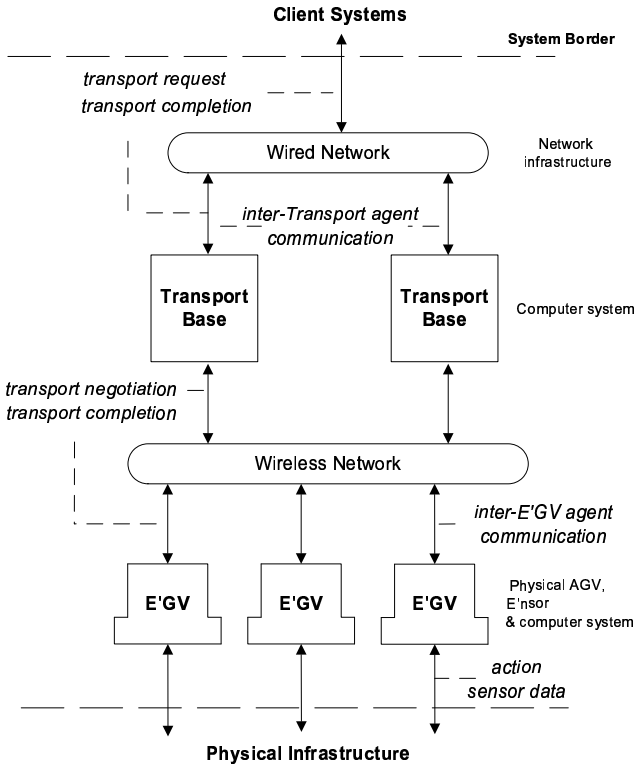

Figure 2: Decentralized E'GV system

of two kinds of agents, *transport agents* and *E'GV agents*. Transport agents are located at *transport bases*. A transport base is a computer system that is charge to manage the transports of a particular area in the warehouse. Together, the transport bases cover the whole layout of the warehouse. Distribution of the transport bases has several advantages, it improves robustness and reduce complexity of deployment and maintenance of the system. E'GV agents are located in E'GVs that are situated on the factory floor. A transport agent represents a transport that needs to be handled by an E'GV. E'GV agents are responsible for executing the assigned transports. We fully reused the E'nsor software that deals with the low-level control of the E'GVs. As such, the E'GV agents control the movement and actions of E'GVs 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 E'GVs to communicate with each other and with transport agents on transport bases.

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

3.3 Responsibilities of Agents and Environment

To describe how we apply a situated MAS to control an E'GV system, we revisit the five core functionalities of the application described in section 2.1. We describe the main responsibilities of the two types of agents in our solution, as well as the responsibilities of the virtual environment.

Transport Assignment. Transport bases receive transport requests from client systems. For each new transport, a new transport agent is created that is responsible to assign the transport to an E'GV 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. To assign the transport, we study two different tracks, one with a flexible 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 E'GVs to the pick location of the load, while each interested E'GV emits a gradient field that repels other competitor E'GVs. The gradient fields guide idle E'GVs toward the most appropriate transports, ensuring maximal flexibility (e.g., E'GVs take into account opportunities –new transports that pop up– when they drive toward a load). Once the transport is assigned, the awarded E'GV handles the transport. As soon as the transport is completed, the E'GV 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 E'GV is unable to complete the transport, the transport agent may negotiate with other E'GVs 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 are similar 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 E'GV to drive over the segment. The agent perceives the signs in their 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 E'GVs come in each others neighborhood, the information of the traffic maps is exchanged and merged to provide up-to-date information to the E'GV agents. Since pheromones evaporate over time, outdated information automatically vanishes over time. E'GV agents take the information on the traffic maps into account when they decide how to drive through the warehouse.

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

Deadlock Avoidance. The basic mechanisms for deadlock avoidance provided in the traditional approach could be adopted in the MAS approach. E.g., when an E'GV approaches a bidirectional path in the layout, the E'GV agent can lock that path via the hull reservation mechanism, or when an E'GV reaches an entry point of a critical area where only a limited number of E'GVs are allowed, the E'GV agent can instruct the EGV 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 E'GV movements and provides feedback to the E'GV agents, and another where E'GVs themselves monitor their neighborhood and exchange information regarding deadlock threats via the environment.

4. SOFTWARE ARCHITECTURE

This section zooms in on the software architecture of the E'GV transportation system. We focus on the software architecture deployed on E'GVs, the architecture of the transport bases is currently in development. First we give a broad overview of the software architecture deployed on E'GVs. Then we explain the decision making of E'GVs. Next we zoom in on the virtual environment (VE). We explain how the VE maintains its state in the distributed setting, and we describe how the VE handles perception, action and communication. Concluding with an example, we describe how the VE is exploited by the E'GV agents to avoid collisions.

4.1 Software Architecture Deployed on E'GVs

Fig. 3 depicts an overview of the software deployed on an E'GV. The E'GV agent is shown in the top layer of the model. The E'GV agent is situated in the virtual environment, shown as a layer below the top layer. The VE uses the middleware layer, that is composed of a Message Transfer

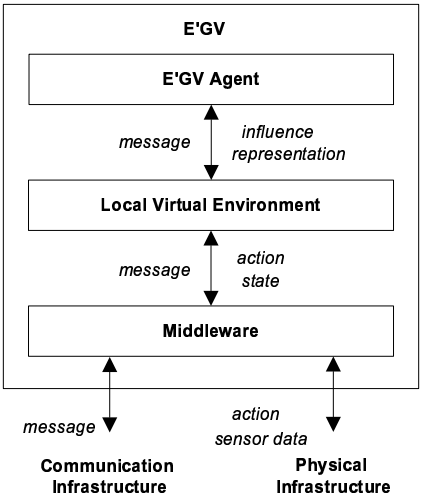

Figure 3: E'GV software architecture

System, the ObjectPlaces middleware [7] and E'nsor. The middleware layer is connected to the communication infrastructure and the actuators and sensors of the E'GV.

In the E'GV application, the only physical infrastructure available are the E'GVs and the wireless network to allow communication between the E'GVs. In other words, the VE is necessarily distributed over the E'GVs. In effect, each E'GV maintains a *local virtual environment*, which is a local manifestation of the VE. Local VEs are merged with other local VEs opportunistically, as the need arises. In other words, *the* virtual environment as a software entity does not exist; rather, there are as many local VEs as there are E'GVs. Some of these local VEs may recently be synchronized with each other, while others may not. From the agent perspective, the VE appears as one entity. To synchronize state, the VE make use of the ObjectPlaces middleware.

4.2 E'GV agents

We now zoom in on the E'GV agents. The architecture of the agents is based on the reference architecture for situated MASs described in [10] and [12]. Fig. 4 depicts an overview of architecture of the E'GV agents. The agent architecture

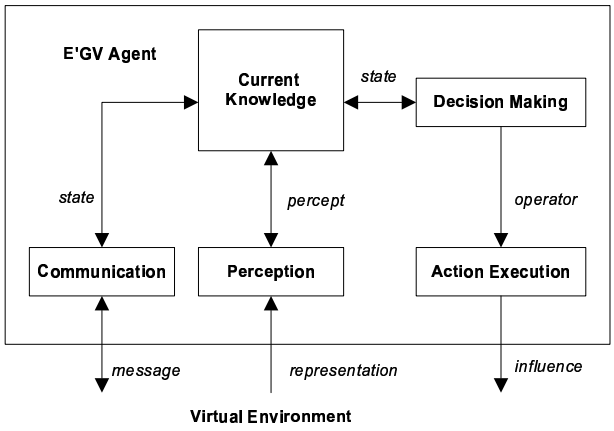

Figure 4: The E'GV agent architecture

is basically modelled as a data repository pattern. Differ-

ent concerns of the agent behavior, i.e., perception, communication, and decision making, are modelled as a separate modules of the architecture. The current knowledge module serves as data repository for the different modules.

Here, we limit the discussion to the agent's decision making module. Fig. 5 depicts the architecture of this module. The decision making module is responsible to select appropriate actions to perform the current transport. Selection of transports is a responsibility of the *communication module* (see Fig.4) of the agent. When the agent becomes idle, it will either park or charge its battery. The agent's current knowledge serves as a shared data repository for the sub-modules of decision making. The *action controller* coordinates the selection of an appropriate action. When a job is finished, the action selector instructs the *job selector* to select the next job. After job selection, the *action selection* module selects an action at a fairly high level (move, pick, park etc.). The action selection module is set up as a free-flow architecture [9]. For a detailed study of the action selection module, we refer to [8]. The *action generation* module transforms this action into a concrete preliminary action (e.g., move(segment x)). The *collision avoidance* module is responsible to lock the trajectory associated with the selected action. As soon as the trajectory is locked, the collision avoidance module passes the confirmed action to the virtual environment. If during the subsequent phases the selected action can not be executed (e.g., an obstacle is detected on the trajectory), feedback is sent to the action controller that will inform the appropriate module to revise its decision. This feedback loop enables flexible decision making.

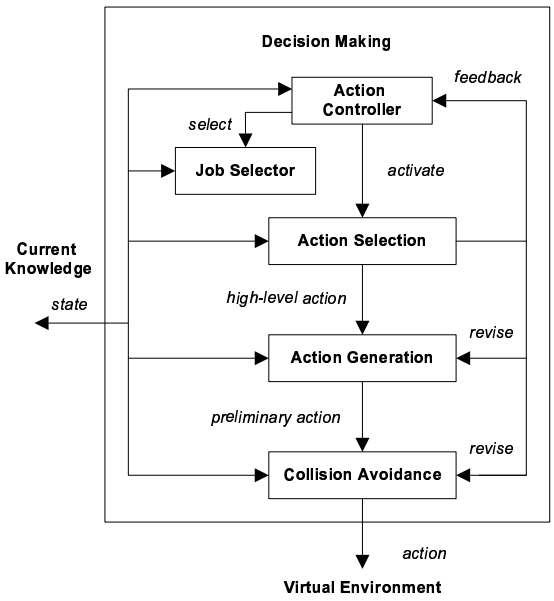

Figure 5: Decision making module

4.3 The Virtual Environment

We now zoom in on the VE. First we explain how the VE maintains its state across the E'GVs. Then we discuss how the VE offers abilities for perception, communication and actions to E'GV agents.

4.3.1 Managing State in the Virtual Environment

An important responsibility of the local VE is to keep its state synchronized with other local VEs. The state a local VE maintains is divided into three categories:

1. Static state: this is state that does not change in the system. A typical example is the layout of the factory floor, which is needed for the E'GV to navigate. Static state must never be exchanged between local VEs, since it is common knowledge and never changes.

2. Observable state: this is state that can be changed in one local VE, while other local VEs can only observe the state. An E'GV typically obtains this kind of state from its sensors directly. An example is an E'GV's position. Local VEs are able to observe another E'GV's position, but only the local VE on the E'GV itself is able to read it from its sensor, and change the representation of the position in the local VE. No conflict ever arises between two local VEs concerning observable state.

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

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

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

The second activity the VE performs is synchronizing the state of the local VE with other E'GVs. This is supported by the ObjectPlaces middleware [7]. ObjectPlaces offers high-level abstractions to deal with communication in mobile and ad hoc networks. The local VE uses the middleware by sharing objects in a tuplespace-like container, called an *objectplace*. Every E'GV has one objectplace locally available. Objects in objectplaces on remote E'GVs can be gathered using a *view*. The local VE can define a view by (1) specifying which E'GVs' objectplaces need to be included in the view (e.g. the objectplaces of all E'GVs 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 objectplaces. In other words, a local VE shares data with other E'GVs by putting objects in the local VE's objectplace. Local VEs gather data from other E'GVs by defining a view on the objectplaces of those E'GVs. For example, when an E'GV agent marks a hull in the environment, this hull is published in the local VE's objectplace. When the E'GV agent wants to perceive hulls in its vicinity, the local VE defines a view on all hull objects in objectplaces of E'GVs within a certain physical distance from the E'GV. The middleware then

gathers the hull objects from the objectplaces on the appropriate E'GVs. The local VE can then use this view to return the results to the agent.

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

In summary, the virtual environment deals with the management of state in the distributed system, hiding complex aspects of distribution from the E'GV agent.

4.3.2 Perceiving, Acting and Communicating

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

Perception in the VE 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 E'GVs, the perception manager queries the necessary information from the local VE and returns it to the agent.

Actions are handled by the *action manager*. A first kind of actions concerns the physical actions of the E'GV, for example moving over a segment or picking up 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 E'GV, but manipulates the VE. Marking hulls is one example of this, which is described in detail in section 4.4. In general, an action can be handled by passing it down to the Ensor layer, and/or by changing the local VE.

Communication is handled by the *communication manager*. Agents can communicate directly with other agents through the VE. A typical example is an E'GV agent that communicates with a transport agent. Another example is an E'GV agent that requests the E'GV agent of a waiting E'GV to move out of the way. The VE 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 VE offers high level primitives to the E'GV agent to act in the world, perceive the world, and communicate with other agents. The VE shields the agent from having to deal with lower level issues.

4.4 A 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 E'GV in the system, a series of *hulls* are calculated along the path each E'GV is going to drive. When two or more such *hull projections* overlap, E'GVs are on a collision course and all except one E'GV are commanded to wait.

In a decentralized architecture, a central arbitrator does not exist. However, since the virtual environment emulates a shared environment, the agents can act as if they are situated in a (real) shared environment, while the virtual environment takes on the burden of coordination.

Fig. 6 shows a series of screenshots of a simulation run. The code for the agents and virtual environment is the same whether the software runs in simulation or not; we have also tested on a real setup, but only with two E'GVs on a smaller layout. The system shown is part of a real layout used for one of the clients of Egemin.

In Fig. 6(a), two E'GVs can be seen that are approaching each other. From now on, we will call the E'GV approaching from the top E'GV A, and the other E'GV B. Both E'GVs are projecting hulls in the environment. At this point, no conflict is detected. In Fig. 6(b), the E'GV B has projected further ahead, and is now in conflict with the hull projection of the E'GV A. However, since E'GV A's hull projection was already confirmed to it, E'GV B must wait (its hull projection is shown in red). In 6(c), E'GV A is taking the curve, passing E'GV B. Finally, in 6(d), E'GV A has parked at the bottom, and E'GV B can start moving as well.

In a decentralized setting, the E'GVs execute a mutual exclusion protocol to make sure collision does not occur. The protocol is a variant of well-known distributed mutual exclusion algorithms based on voting. Informally, each E'GV wishing to project its hull further ahead, asks all E'GVs within a given area for permission. If it does not receive this permission from every E'GV in the area, it waits; otherwise, it proceeds. The size of the area is based on the position and the length of the hull projection of each E'GV; this information is exchanged periodically between all E'GVs in the system. Based on other E'GVs positions and length of hull projections, an E'GV knows with whom it might collide, and start negotiating.

This example shows the strength of the virtual environment abstraction: it relieves the agents from the complexity of distribution, and offers a high-level abstraction to lower-level infrastructure.

5. CONCLUSIONS

The evolution of the market of logistic services in warehouses and manufactories put forward new challenging requirements. Customers request for flexibility and openness of the transportation systems, vehicles must be able to adapt their behavior with changing circumstances. In this paper, we presented an innovative approach to the control of an E'GV transportation system that aims to cope with these new requirements, while keeping in mind the benefits of the traditionally centralized approach. In this approach, we exploit principles and mechanisms from situated multiagent systems for modelling and implementing a decentralized control system. One of the main concerns in this approach is careful consideration of bandwidth.

At this moment, real E'GVs are able to manipulate loads, drive around, and avoid collisions in an industrial test setup. The next challenges are order assignment and deadlock avoidance. Currently, we are developing architectural models to cope with these challenges. It would be unfair to state that the current status of the project is production software. Yet we will do field testing –not only lab testing– in the near future, and we aim to deliver a fully functional decentralized control system within the term of the project.

6. REFERENCES

[1] S. Berman, Y. Edan, and M. Jamshidi. Decentralized autonomous AGVs in material handling. *Transactions on Robotics and Automation*, 19(4), 2003.

[2] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in telecommunications networks with "smart" ant-like

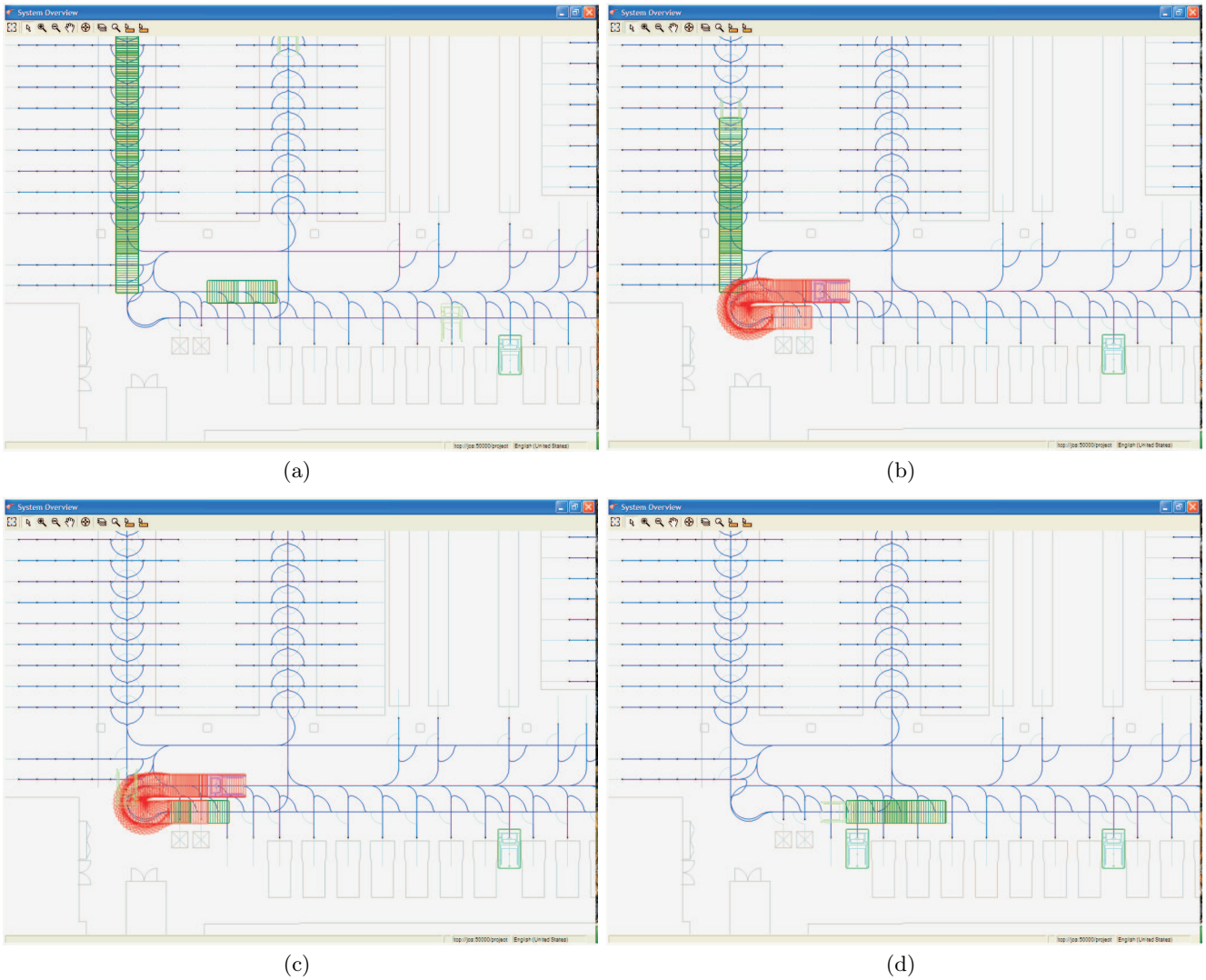

Figure 6: (a) Two E'GVs approaching, (b) A conflict is detected, (c) One E'GV passes safely, (d) The second E'GV can pass as well.

agents. In *Intelligent Agents for Telecommunications Applications*, 1998.

[3] L. Ong. An investigation of an agent-based scheduling in decentralised manufacturing control. *Ph.D Disseration, University of Cambridge*, 2003.

[4] L. Pallottino, V. G. Scordio, E. Frazzoli, and A. Bicchi. Decentralized cooperative conflict resolution for multiple nonholonomic vehicles. In *AIAA Ccnf. on Guidance, Navigation and Control*, 2005.

[5] V. Parunak, A. Baker, and S. Clark. The AARIA agent architecture: From manufacturing requirements to agent-based system design. In *Agent-Based Manufacturing*. Minneapolis, MN, 1998.

[6] J. Sauter and V. Parunak. Ants in the supply chain. In *Agent based Decision Support for Managing the Internet-Enabled Supply Chain*. Seattle, WA, 1999.

[7] K. Schelfthout and T. Holvoet. Views: Customizable abstractions for context-aware applications in MANETs. In *Software Engineering for Large-Scale Multi-Agent Systems, St.Louis, USA*, 2005.

[8] E. Steegmans, D. Weyns, T. Holvoet, and Y. Berbers. Designing roles for situated agents. In *Agent-Oriented Software Engineering*. New York, 2004.

[9] T. Tyrrell. *Computational Mechanisms for Action Selection. Ph.D Thesis*. University of Edinburgh, 1993.

[10] D. Weyns and T. Holvoet. A formal model for situated multi-agent systems. *Fundamenta Informaticae*, 63(2-3), 2004.

[11] D. Weyns, V. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for multiagent systems, state-of-the-art and research challenges, First International Workshop on Environment for Mutiagent Systems. *Lecture Notes in Computer Science, Springer Verlag*, 3374, 2005.

[12] D. Weyns, E. Steegmans, and T. Holvoet. Protocol based communication for situated multi-agent systems. In *3th Joint Conference on Autonomous Agents and Multi-Agent Systems*. New York, 2004.

[13] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, Ltd., England, 2002.