Architectural Design of a Situated Multiagent System for Controlling Automatic Guided Vehicles

Danny Weyns* and Tom Holvoet

Department of Computer Science,
Katholieke Universiteit Leuven, Belgium
E-mail: {danny.weyns,tom.holvoet}@cs.kuleuven.be
*Corresponding author

Abstract:

Automatic Guided Vehicles (AGVs) are fully automated vehicles that are able to transport goods in an industrial environment. To cope with new and future system requirements such as flexibility and openness, we have applied a situated multiagent system to developed a decentralized control architecture for AGV transportation systems. In this paper, we give an overview of the software architecture of the system and we zoom in on two specific concerns: transport assignment and collision avoidance. We discuss the evaluation of the software architecture and test results obtained from realistic simulations and a demonstrator system that we have developed.

The architectural design and development of this real-world application learns us that the primary use of a situated multiagent system comes from the way in which it *structures* the software. In particular, the set of adaptive agents that coordinate through the environment allows to shape the software architecture of the transport application to provide the required functionalities of the system and achieve the important quality goals flexibility and openness.

Keywords: situated multiagent system; software architecture; automated transportation system, AGV.

Reference to this paper should be made as follows: Weyns, D. and Holvoet, T. (2007) 'Architectural design of a situated multiagent system for controlling automatic guided vehicles ', Int. J. Agent-Oriented Software Engineering (IJAOSE).

Biographical notes: Danny Weyns is a post-doctoral researcher at the Katholieke Universiteit of Leuven, Belgium. He obtained his PhD in 2006 for research on the connection between multiagent systems and software architecture. Danny's research interests include the role of the environment in multiagent systems, architectural description languages for decentralized systems, and aspect-oriented software development.

Tom Holvoet is a full time professor in the Computer Science Department at the Katholieke Universiteit of Leuven, Belgium. He obtained his PhD in 1997 for work in open concurrent software development. Since then he is active in research on decentralized systems, aspect-oriented software development, and middleware. Tom leads a research group with seven PhD students within the DistriNet Lab.

1 INTRODUCTION

Automatic Guided Vehicles (AGVs) are fully automated, battery powered vehicles that are able to transport goods in a logistic or production environment. An AGV control system receives transport requests from a client system such as a warehouse managing system or machine operating software, and instructs AGVs to execute the transports. The stream of transports that enter the transportation system is typically irregular and unpredictable. AGVs are provided with low-level control software connected to sensors and actuators to move safely through the warehouse environment. While moving, the vehicles follow specific paths in the warehouse by means of a navigation system which uses stationary beacons in the work area (e.g., laser reflectors on walls or magnet strips in the floor). To enable the AGV software to communicate with software systems on other machines, the vehicles are equipped with infrastructure for wireless communication.

Traditionally, AGVs are directly controlled by a central server. AGVs have little autonomy: the server plans the schedule for the system as a whole, dispatches commands to the AGVs and continually polls their status. In a joint R&D project (EMC^2 [13]), our research group and Egemin, a producer of logistic service systems, have applied a situated multiagent system architecture. The goal of the project was to investigate the feasibility of a *decentralized* control system to cope with new and future system requirements such as flexibility and openness. In the new architecture the AGVs are provided with a considerable amount of autonomy. This opens perspectives to improve flexibility and openness of the system. The AGVs can adapt themselves to changing situations in their vicinity: transport assignment is dynamic, the system can deal autonomously with AGVs leaving and re-entering the system for maintenance, etc.

In previous work, we have studied a number of specific research issues of AGV transportation systems. [31] focusses on the decentralized control, [29] zooms in on transport assignment, and [7] reflects on the assessment of qualities. In this paper, we give an integrated overview of the software architecture of the decentralized AGV control system that we have developed. Starting from system requirements, we discuss the architectural design of the application. We elaborate on the evaluation of the software architecture of the system and we discuss test results collected from simulations and an implemented demonstration system. Finally, we conclude and reflect on our experiences with applying MAS in practice.

2 AGV TRANSPORTATION SYSTEM

In this section, we introduce the AGV application. We start with an overview of the functionalities of the system and we discuss the main quality requirements. System requirements are kept fairly general, independent of any particular AGV system. In section 7, we zoom in on specific functionalities and quality scenario's for a concrete AGV transportation application. Then we outline a number of important problem characteristics of industrial AGV transportation systems that have to be taken into account during architectural design.

2.1 Main Functionalities

The main functionality the system has to perform is *handling transports*, i.e. moving loads from one place to another. There should be enough AGVs available to execute the transports that enter the system, i.e. the AGVs should be able to handle the load of the system. In order to execute transports, the main functionalities the system has to perform are:

- Transport assignment: transports are generated by clients and have to be assigned to AGVs that can execute them.
- Routing: AGVs must route efficiently through the layout of the warehouse when executing their transports.
- Collision avoidance: safety measures are necessary when AGVs cross the same intersection at the same moment and when AGVs pass each other on closely located paths.
- Deadlock avoidance: since AGVs are relatively constrained in their movements (they cannot divert from their path), the system must ensure that AGVs do not find themselves in a deadlock situation.

To perform transport tasks, AGVs have to maintain their battery. AGVs can charge their battery at the available charging stations. Finally, when an AGV is idle it can park at a free park location.

2.2 Quality Requirements

Stakeholders of an AGV transportation system have various quality requirements. *Performance* is a major requirement; customers expect that transports are handled efficiently by the transportation system. *Configurability* is important, it allows installations to be easily tailored to client-specific demands. Obviously, an automated system is expected to be *robust*, intervention of service operators is time consuming and costly.

Besides these "traditional" qualities, evolution of the market puts forward new quality requirements. Customers request for self-adapting systems, i.e. systems that are able to adapt their behavior with changing circumstances autonomously. Self-adaptation with respect to system dynamics translates to two specific quality goals: flexibility and openness.

Flexibility refers to the system's ability to deal with dynamic operating conditions. The centralized planning algorithms for transport assignment and routing applied by Egemin are based on predefined rules that are associated with AGVs and locations in the layout. This approach lacks flexibility. A flexible control system allows an AGV that is assigned a transport and moves toward the load, to switch tasks along the way if a more interesting transport pops up. Another desired property is that the system can handle particular situations autonomously, e.g., when a truck with loads arrives, the system should adapt its behavior taking into account this new task.

Openness refers to the transportation system's ability to deal with AGVs leaving and (re-)entering the system autonomously. Examples are an AGV that temporarily leaves the system for maintenance, and an AGV that resumes work after its battery is recharged. In some cases, customers expect to be able to intervene manually during execution of the system, e.g., to force an AGV to perform a particular job.

In summary, flexibility and openness are high-ranking quality requirements for today AGV transportation systems. One possibility to tackle these new quality requirements would be to adapt the central planning approach aiming to improve the flexibility and openness of the system. In the EMC^2 project, we investigated the feasibility to apply a new decentralized architecture to cope with the new quality requirements.

2.3 Problem Characteristics

In addition to the required functionalities and the quality goals, a number of specific problem characteristics must be considered during architectural design.

- AGVs have to move toward loads before they can actually execute the transports. Moving toward a load may imply a considerable effort.
- AGVs are very constrained in their movements, they are confined to follow the paths of a predefined layout.
- The speed of AGVs is orders of magnitude lower than the speed of communication and the execution of the control software.
- A wireless LAN provides continual communication access to the distributed software system.

The architect(s) have to take into account these problem characteristics when selecting suitable architectural approaches for the software architecture.

3 HIGH LEVEL MODEL OF THE AGV SYSTEM

In this section, we give a high-level model of the agent-based AGV system. First, we introduce the agent types. Then, we explain the concept of virtual environment and we illustrate how the agents use this environment to coordinate their behavior.

3.1 AGV Agents and Transport Agents

We have introduced two types of agents: AGV agents and transport agents. The choice to let each AGV be controlled by an AGV agent is obvious. Transports have to be handled in negotiation with different AGVs, therefore we have introduced transport agents.

Each AGV in the system is controlled by an AGV agent. The agent is responsible for obtaining and handling transports, and ensuring that the AGV gets maintenance on time. As such, an AGV becomes an autonomous entity that can take advantage of opportunities that occur in its vicinity, and that can enter and leave the system without interrupting the rest of the system.

Each transport in the system is represented by a transport agent. A transport agent is responsible for assigning the transport to an AGV and reporting the status and completion of the transport to the client that has requested the transport. Transport agents are autonomous entities that interact with AGV agents to find suitable AGVs to execute the transports. The transport agents reside at a transport base, i.e. a dedicated computer located in the warehouse.

3.2 Virtual Environment

To achieve the system functionality, AGV agents and transport agents have to coordinate. Agents have to coordinate for routing, for transport assignment, for collision avoidance, etc. One approach is to provide an infrastructure for communication that enables the agents to exchange messages to coordinate their behavior. Such approach however, would put the full complexity of coordination in the agents and result in complex architectures of the agents, in particular for the AGV agents. We have chosen for a solution that enables the agents to exploit the environment to coordinate their behavior. This approach separates responsibilities in the system and helps to manage the complexity [30].

The physical environment in which AGVs are situated is very constrained: AGVs cannot manipulate the environment, except by picking up and dropping loads. This restricts how agents can exploit their environment. We introduced a *virtual environment* that offers a medium for AGV agents and transport agents to exchange information and to coordinate their behavior. Besides, the virtual environment serves as a suitable abstraction that shields the agents from low-level issues, such as the communication of messages and the physical control of an AGV vehicle. Fig. 1 shows a high-level model of an AGV transportation system. The virtual environment is necessarily distributed over the AGVs and the trans-

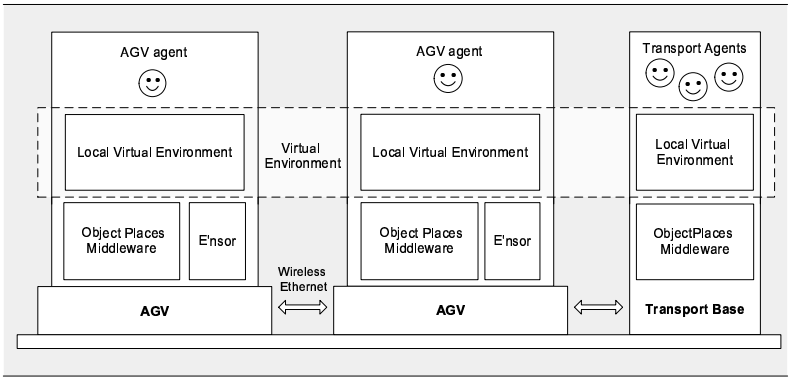

Figure 1 High-level model of an AGV transportation system

port base. In effect, each AGV and the transport base maintain a local virtual environment, which is a local manifestation of the virtual environment. The instances are tailored to the type of agents deployed on the nodes. For example, the local virtual environment on the AGVs enables the AGV agent to read out the status of the AGV and send commands to the vehicle. Obviously, this functionality is not available in the local virtual environment on the transport base.

The states of local virtual environments on neighboring nodes are synchronized with each other opportunistically, as the need arises. State synchronization is supported by the ObjectPlaces middleware [22]. ObjectPlaces provides services to gather data from neighboring network nodes and set up an interaction between neighboring nodes. We discuss an interaction protocol for collision avoidance of AGVs in section 6.

AGVs are equipped with low-level control software that is called E'nsor (Egemin Navigation System On Robot). E'nsor provides an interface to command the AGV and to monitor its state. We fully reused the control software in the project. The local virtual environment uses E'nsor to steer the vehicle based on the commands of the AGV agent,

and to regularly poll the vehicle's status and adjust its own state appropriately.

Coordination Medium. The local virtual environment offers high-level primitives to agents to act in the environment, perceive the environment, and communicate with other agents. This enables agents to share information and coordinate their behavior. As an illustration, we explain how agents exploit the virtual environment to avoid collisions by coordinating with other agents through the local virtual environment. AGV agents mark the path they are going to drive in their local virtual environment using *hulls*. A hull is the physical area an AGV occupies and a series of hulls describe the physical area the AGV occupies along a certain path. If an AGV's hulls do not intersect with hulls of other AGVs, the AGV can drive over the reserved path. In case of a conflict, the involved local virtual environments use the priorities of the transported loads to determine which AGV can move on. AGV agents monitor the local virtual environment and only instruct the AGV to move on when they are allowed. Afterwards, the AGV agents remove the markings in the virtual environment.

This example shows that the virtual environment serves as a flexible coordination medium: agents coordinate by putting marks in the local virtual environment, and observing marks from other agents.

4 SOFTWARE ARCHITECTURE

We now give an overview of the software architecture of the AGV transportation system. First, we explain the architecture design process we used. Next, we discuss the main views of the software architecture.

4.1 Architectural Design

The general motivation to apply a situated MAS to the AGV transportation system was the importance of the required qualities flexibility and openness. Situated agents are self-adapting entities that are able to efficiently respond to changing circumstances in the system. During architectural design, we have applied various mechanisms for adaptivity for situated MAS [28], including selective perception, behavior-based action selection with roles and situated commitments, and protocol-based communication. Selective perception enables an agent to adapt its perception according to its current tasks. An agent selects a set of *foci* that allows it to sense the environment for specific types of information. To enable situated agents to set up collaborations, we have extended behavior-based action selection mechanisms with the notions of *role* and *situated commitment*. A role represents a coherent part of an agent's functionality in the context of an organization. A situated commitment is an engagement of an agent to give preference to the actions of a particular role in the commitment. Behavior-based action selection with roles and situated commitments allow agents to adapt their behavior efficiently with changing circumstances in the environment. Protocol-based communication structures communicative interactions among agent according to well-defined sequences of messages. Exchanging messages enable situated agents to set up explicit collaborations and exploit opportunities that occur in their local context.

For the architectural design, we used the Attribute Driven Design method (ADD [9, 4]). ADD is a decomposition method that is based on understanding how to achieve quality

goals through proven architectural approaches. Roughly spoken, the design process consisted of the following steps. First, we have mapped the system functionality onto the basic decomposition of the system: AGV en transport agents and the local virtual environment. Then, we have iteratively decomposed the agents and the local virtual environment. In each decomposition step, we selected an architectural element of the software architecture and we determined the architectural drivers (i.e. the target functional and quality attribute requirements for that element). The order in which we have refined the architectural elements was essentially based on the incremental development of the application. We started with the functionality for one AGV to drive, then followed collision avoidance, next order assignment, deadlock avoidance, etc. For each decomposition, we have selected suitable architectural patterns to refine the architectural element. Where applicable, we have used the specification of the mechanisms for adaptivity to decompose architectural elements. The decomposition ended when a suitable level of detail was reached to allow the developers to build the software.

We now give an overview of the main views of the software architecture. Subsequently, we explain the deployment view, the high-level module decomposition view of the AGV software, and finally the collaboration component views of the AGV agent and the local virtual environment of AGVs. [6] provides the complete documentation of the software architecture of the AGV transportation.

4.2 Deployment View

Fig. 2 gives a general overview of the AGV transportation system and shows how the application software is allocated to computer hardware. The software consists of two types of subsystems: transport base system and AGV control system. The relationship of the deployment view is *allocated-to* [10]. Software elements are allocated to hardware elements, e.g., a transport base system is allocated to a transport base.

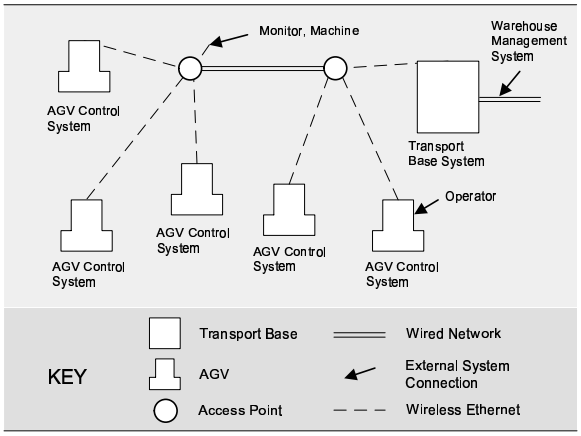

Figure 2 Deployment view of the AGV transportation system

4.2.1 Elements and their Properties

The **Transport Base System** provides the software to manage transports in the AGV system. The transport base system handles the communication with the warehouse man-

agement system. It receives transport requests and assigns the transports to suitable AGVs, and it reports the status and completion of the transports to the warehouse management system. The transport base system executes on the transport base, i.e. a stationary computer. The transport base system provides a public interface that allows an external monitor system to observe the status of the AGV transportation system.

The **AGV Control System** provides the control software to command an AGV machine to handle transports and to perform maintenance activities. Each AGV control system is deployed on a computer that is installed on a mobile AGV. AGV control systems provide a public interface that allows a monitor to observe the status of the AGVs, and let a service operator take over the control of the vehicle when necessary.

Communication Network. All the subsystems can communicate via a wireless network. The warehouse management system interacts with the AGV transportation system via the wired network. To debug and monitor the system, AGVs and the transport base can be accessed remotely via an external monitor system.

4.2.2 Rationale

The main motivation for the top level decomposition of the transportation system is the separation of functionality for transport assignment (ensuring that the work is done) from functionality for executing transports (doing the work). By providing each AGV vehicle with an AGV control system, AGVs become autonomous entities that can exploit opportunities that occur in their vicinity, and that can enter and leave the system without interrupting the rest of the system. Endowing AGVs with autonomy is a key property for flexibility and openness in the system.

The separation of functionality for transport assignment and executing transports also supports incremental development. In the initial stage of the project, we developed a basic version of the AGV control system that provided support for performing transports and avoiding collisions. For testing, we manually assigned transports to AGVs. In the next phase, when we extended the functionalities of AGVs and integrated automated transport assignment, the top level decomposition served as a means to assign the work to development teams.

4.3 Module Decomposition of the AGV Control System

Fig. 3 shows the primary presentation of the module uses view of the AGV control system.

The relation in the module uses view is *uses*. An element uses another element if the correctness of the first element depends on the correct implementation of the second element [10].

4.3.1 Elements and their Properties

AGV Agent. An AGV agent is responsible for controlling an AGV vehicle. The main functionalities of an AGV agent are: (1) obtaining transport tasks; (2) handling jobs; (3) avoiding collisions; (4) avoiding deadlock; (5) maintaining the AGV machine (charging battery, calibrating etc.); (6) parking when the AGV is idle.

Local Virtual Environment. The local virtual environment offers a medium that the AGV

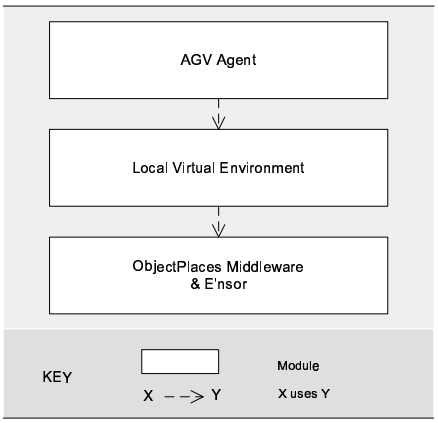

Figure 3 Module uses view of the AGV control system

agent can use coordinate its behavior with other agents. It also shields the AGV agent from low-level issues, such as the communication of messages to remote agents and the physical control of the AGV. Particular responsibilities of the local virtual environment are: (1) representing and maintaining relevant state of the physical environment and the AGV vehicle; (2) representing additional state for coordination purposes; (3) enabling the manipulation of state; (4) synchronization of state with neighboring local virtual environments; (5) providing support to signal state changes; (6) translating the actions of the AGV agent to actuator commands of the AGV vihicle; (7) translating and dispatching messages from and to other agents.

ObjectPlaces Middleware & E'nsor. The ObjectPlaces middleware enables communication with software systems on other nodes, providing a means to synchronize the state of the local virtual environment with the state of local virtual environments on neighboring nodes. E'nsor is the low-level control software of the AGV vehicle. The E'nsor software provides an interface to command the AGV vehicle and to read out its status. The E'nsor interface defines instructions to move the vehicle over a particular distance and possibly execute an action at the end of the trajectory. Example actions are `move(Segment x)` that instructs the AGV to drive over segment x and `pick(Segment y)` that instructs the vehicle to drive over segment y and pick a load at the end of the segment. The physical execution of the commands is managed by E'nsor. As such, the AGV agent can control the movement and actions of the AGV at a fairly high-level of abstraction.

4.3.2 Design Rationale

The layered decomposition of the AGV control system separates responsibilities. The AGV agent is a self-managing entity that is able to flexibly adjust its behavior with changing circumstances in the system. The local virtual environment provides an abstraction that allows agents to interact and coordinate their behavior in a way that is not possible in the physical environment. Separation of responsibilities helps to manage complexity. Since AGV agents only interact with other agents situated in their vicinity, state has only to be synchronized between neighboring local virtual environments. The ObjectPlaces middleware takes the burden of mobility. An alternative for indirect coordination through the

local virtual environment is an approach where the functionality to control an AGV vehicle is assigned to an AGV agent only, and where AGV agents coordinate through message exchange. Such a design however, would put the full complexity of coordination in the AGV agent, resulting in a more complex architecture.

4.4 Collaborating Components of the AGV Agent

We now zoom in on the software architecture of agents. We focus on the AGV agent. Fig. 4 shows a collaborating components view of the AGV agent. The collaborating components view shows the software system as a set of interacting runtime components that use a set of shared data repositories to realize the required system functionalities [28].

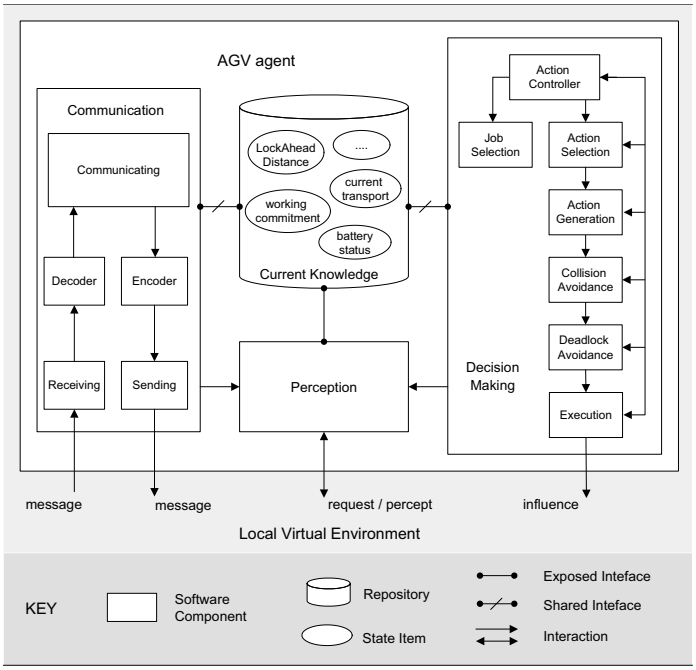

Figure 4 Collaborating components view of the AGV agent

4.4.1 Elements and their Properties

The **Current Knowledge** repository contains state that the agent uses for decision making and communication. Current knowledge consists of static state and dynamic state. An example of static state is the value of LockAheadDistance. This parameter determines the length of the path AGVs have to reserve when they move on to drive smoothly and avoid collisions; we elaborate on path locking in section 6. Examples of dynamic state are state collected from the observation of the environment such as the positions of neighboring AGVs, state of commitments related to collaborations with other agents, and runtime state related to the agent itself such as the battery status of the AGV. The current knowledge repository offers a shared interface to the communication and decision making components that can concurrently read and write state. The perception component is connected

to a separate interface to update the agents dynamic state according to the representations derived from observing the local virtual environment.

Perception enables the AGV agent to sense the local virtual environment according to the perception requests of communication and decision making, and to update the agent's current knowledge accordingly. AGV agents use different foci to sense the state of the local virtual environment that represents state in the physical environment (e.g., the positions of neighboring AGVs) and state that relates to virtual representations (e.g., fields that are used for transport assignment, see section 5).

The **Communication** component handles the agents communicative interactions with other agents. The main functionality of communication in the AGV transportation is handling messages to assign transports. The communicating component encapsulates the behavior of the communication component, including the protocols and the communication language used by the AGV agent. The protocols are specified as statecharts, we discuss an example in section 5.

The **Decision Making** component handles the agent's actions. Due to the complexity of decision making of the AGV agent, we have modeled the decision making component as a hybrid architecture that combines a blackboard pattern with sequential processing. This architecture combines complex interpretation of data with decision making at subsequent levels of abstraction. The current knowledge repository serves as blackboard, while the action controller coordinates the selection of a suitable action. After job selection (execute transport, charge battery, etc.), the action selection component selects an action at a fairly high-level (`move`, `pick`, etc.). We have designed the action selection component as a free-flow tree [21, 26] extended with the notions of role and situated commitment [28]. The main roles of the AGV agent are `work`, `charge`, and `park`. The main situated commitments are `working commitment` and `charging commitment`. Action generation transforms the selected high-level action into a concrete action (e.g., `move(Segment x)`). Collision avoidance and deadlock avoidance are responsible to lock the trajectory associated with the selected action. As soon as the trajectory is locked, the selected action is passed to the execution component that invokes the action in the local virtual environment. If during the subsequent phases of decision making the selected action can not be executed, feedback is sent to the action controller that will inform the appropriate component to revise its decision. For example, if for a selected action `move(segment x)` the collision avoidance module detects that there is a long waiting time for this segment, it informs the action controller that in turn instructs the action generation component to consider an alternative route.

4.4.2 Design Rationale

The current knowledge repository enables the data accessors to share state and to communicate indirectly. Communication and decision making act in parallel, each component in its own pace, supporting flexibility. Communication in the AGV application happens at a much higher pace than action selection. This difference in execution speed is exploited to continuously reconsider transport assignment in the period between an AGV starts moving towards a load and the moment when the AGV picks the load (a detailed explanation follows in section 5).

In the initial phase of the project, we used a free-flow tree for integral decision making. However, with the integration of collision avoidance and deadlock avoidance, it became

clear that the complexity of the tree was no longer manageable. Therefore we decided to apply an architecture that allows incremental decision making. At the top level, a free-flow tree is still used to select an action preserving the advantage of adaptive action selection with a free-flow tree. At the following levels, collision avoidance and deadlock avoidance are taken into account. Each component in the chain is able to send feedback to the action controller to revise the decision. This feedback loop further helps to improve flexible decision making.

4.5 Collaborating Components of the Local Virtual Environment

We now zoom in on the software architecture of the local virtual environment. We focus on the local virtual environment of the AGVs. Fig. 5 shows the collaborating components view of the local virtual environment.

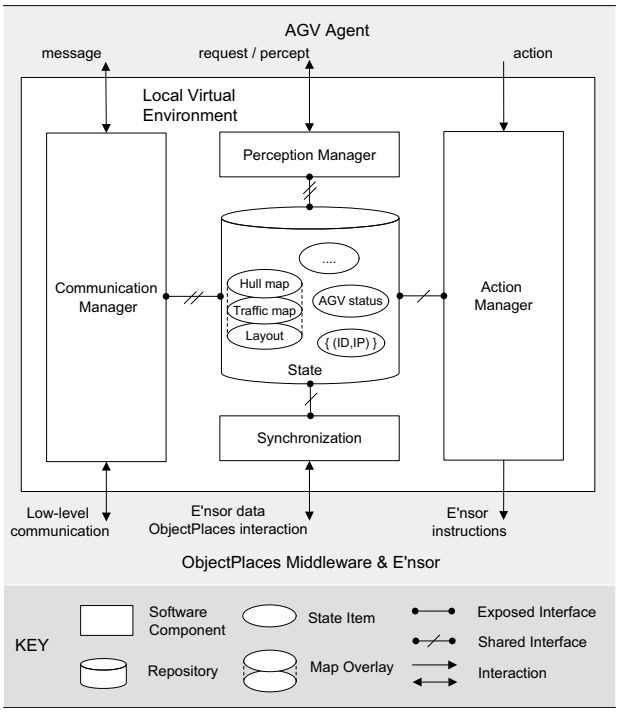

Figure 5 Collaborating components view of the local virtual environment of AGVs

4.5.1 Elements and their Properties

State. The state of the local virtual environment is divided into three categories:

1. Static state: this is state that does not change over time. Examples are the layout of the factory floor, which is needed for the AGV agent to navigate, and `(AGV id, IP number)` tuples used for communication. 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 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 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. An example is a hull map with marks that indicate where AGVs intend to drive—we explain the use of hull maps in detail when we discuss collision avoidance in section 6. When the local virtual environments on different machines synchronize, the local virtual environments must generate a consistent and up-to-date state in both local virtual environments.

Perception Manager handles perception in the local virtual environment. The perception manager's task is straightforward: when the agent requests a percept, e.g., the current positions of neighboring AGVs, the perception manager queries the necessary information from the state repository of the local virtual environment and returns the percept to the agent.

Action Manager handles agents' actions. AGV agents can perform two kinds of actions. One kind are AGV commands, for example moving over a segment and picking up a load. These actions are handled fairly easily by translating them and passing them to the E'nsor control software. A second kind of actions manipulate the state of the local virtual environment. Putting marks in the local virtual environment is an example. An action that changes the state of the local virtual environment may in turn trigger state changes of neighboring local virtual environments (see Synchronization below).

Communication Manager is responsible for exchanging messages between agents. Agents can communicate with other agents through the virtual environment. A typical example is an AGV agent that communicates with a transport agent to assign a transport. Another example is an AGV agent that requests the AGV agent of a waiting AGV to move out of the way. The communication manager translates the high-level messages to low-level communication instructions that can be sent through the network and vise versa (resolving agent names to IP numbers, etc.).

Synchronization has a dual responsibility. It periodically polls E'nsor and updates the state of the local virtual environment accordingly. An example is the maintenance of the actual position of the AGV in the local virtual environment. Furthermore, synchronization is responsible for synchronizing state between local virtual environments of neighboring machines. We give examples of state synchronization when we discuss task assignment and collision avoidance in the following sections.

4.5.2 Design Rationale

Different functionalities provided by the local virtual environment are assigned to different components. This helps architects and developers to focus on specific aspects of the functionality of the local virtual environment.

Changes in the system (e.g., AGVs that enter/leave the system) are reflected in the state of the local virtual environment, releasing agents from the burden to handle such dynam-

ics. As such, the local virtual environment—supported by the ObjectPlaces middleware—supports openness.

Since an AGV agent continuously needs up-to-date data about the system (position of the vehicles, battery status, etc.), we decided to keep the relevant state of the vehicle and it context in the local virtual environment synchronized with the actual state. Therefore, E'nsor and the ObjectPlaces middleware are periodically polled to update the status of the system. As such, the state repository maintains an accurate representation of the relevant local state of the system to the AGV agent.

5 TRANSPORT ASSIGNMENT

We now explain transport assignment in the AGV transportation system. We have developed two different approaches for adaptive transport assignment: FiTA (field-based transport assignment) employs computational fields in the virtual environment to guide AGVs to loads, and DynCNET which is an extention of the well-known contract net protocol (CNET [24]), with "Dyn" referring to support for dynamic task assignment. In this section, we give an overview of FiTA and DynCNET and we compare both approached based on: (1) the performance (throughput and bandwidth usage), (2) a number of important quality attributes (flexibility—adapt to dynamics that happen during transport assignment, openness—take into account agents that enter/leave the system in the process of transport assignment, and robustness to message loss), and (3) the complexity and support to engineer the approaches. In the experiments, CNET is used as a reference protocol.

5.1 FiTA

The basic idea of field-based transport assignment is to let each idle AGV follow the gradient of a field that guides it toward a load that has to be transported. The AGV agents continuously reconsider the situation of the environment and transport assignment is delayed until the load is picked, which benefits the flexibility of the system. To explain FiTA, we use the scenario shown in Fig. 6.

Fields. Transport assignment is achieved by the interaction between AGV agents and transport agents. Physically, transport agents execute at the transport base, but *conceptually* transport agents are situated in the local virtual environment of the transport base system and occupy the position of the load of its associated transport in the local virtual environment. Both AGV and transport agents emit fields in the local virtual environment, see Fig. 6. Transport fields attract idle AGVs. However, to avoid multiple AGVs driving toward the same transport, AGVs emit repulsive fields. AGV agents combine received fields and follow the gradient of the combined fields, that guide them toward pick locations of transports. Fields have a certain range and contain information about the source agent. AGV fields have a fixed range, while the range of transport fields is variable. Fields are refreshed at regular times, according to a predefined refresh rate.

AGV agents store received fields. When an AGV agent perceives fields, it stores the data contained in these fields in a *field-cache*. The field-cache consists of a number of cache-entries. Each cache entry contains the identity of the received field, the most recent data contained in that field and a *freshness*. The freshness is a measure of how up-to-date the cached data is. For example, in Fig. 6 the field-cache of AGV A will consists of three

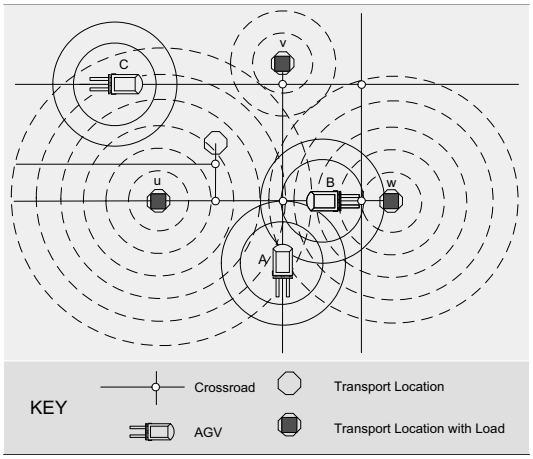

Figure 6 Example scenario to illustrate FiTA

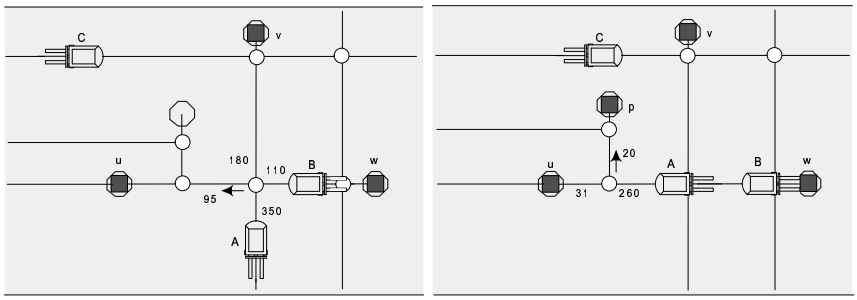

Figure 7 Two successive scenarios in which AGV A follows the gradient of the combined fields. For clarity, we have not drawn the fields.

entries, one for transport u, one for transport w, and one for AGV B.

AGV agents construct calculated-fields to decide their movement. An AGV agent constructs a *calculated-field* to decide in which direction to drive from a node. A calculated-field is a combination of the received fields, which are stored in the field-cache. The lower the freshness of a cache-entry, the lower the influence of the associated field on the calculated-field. The calculated-field is constructed from the last selected node on the AGV's path and contains values for each outgoing segment. An AGV agent follows the calculated-field in the direction of the smallest value. This can be considered as following the *gradient* of the calculated-field downhill.

In the top part of Fig. 7, AGV A constructs a calculated-field on the node in front. Although transport *w* is closer, the calculated-field will guide AGV A toward transport *u*. This is the result of the repulsive effect of AGV B. It would have been ineffective for AGV A to drive toward transport *w*, since AGV B is closer and is maneuvering toward this transport.

Adaptive task assignment. Final transport assignment is delayed until an AGV actually reaches a pick location and picks up the load. This allows agents to adapt the assignment of

transports while the AGVs drive toward loads. By delaying transport assignment, FiTA can cope with changing circumstances that arise. An example is shown in the bottom part of Fig. 7 where a new transport suddenly pops up. While AGV A is driving toward transport *u*, a new transport *p* appears close to AGV A. Since no transport has been assigned to AGV A yet, it can drive toward transport *p*.

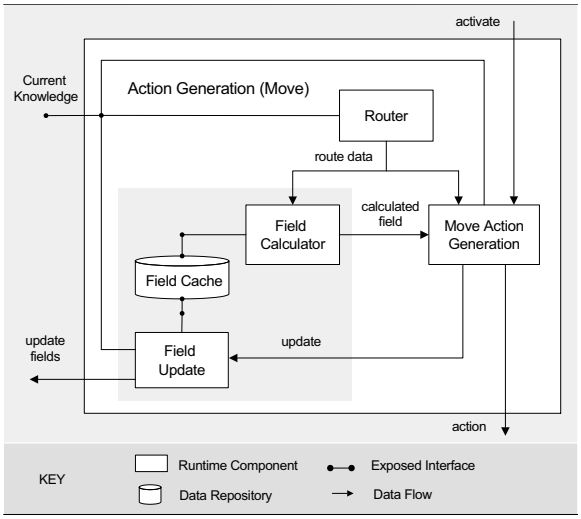

Figure 8 Generation of move actions of the AGV agent in FiTA. The elements in the shaded area deal with field management.

5.1.1 Software Architecture

We limit the discussion to the components of the AVG agent that are involved in the generation of move actions in FiTA. Fig. 8 shows the collaborating components view. We discuss the various elements in turn.

Router. The router uses a map of the warehouse layout with nodes and segments to calculate paths and distances from one node to another. For testing, we have used a static router that uses the A* algorithm [14]. However, the approach is compatible with a dynamic router that would take into account dynamic runtime information such as traffic distribution.

Field Cache: This repository stores the information of fields of other AGV agents and transport agents in cache-entries.

Field Calculator. The field calculator constructs a calculated-field from the last selected target node by combining the received fields from the field-cache. The field calculator makes use of the router to calculate the values of the calculated-field on different positions. The gradient of the calculated-field is used as driving direction on the target node.

Field Update. The field update component is responsible to update the fields for the AGV agent. During task assignment, the move action generator periodically invokes requests for field updates.

Move Action Generation. The move action generation component is activated by the action selection component (see Fig. 4) and generates concrete move actions. During task

assignment, move action generator uses the calculated field of the field calculator to guide the AGV to a load. When the AGV has picked a load, it will inform the transport agent and execute the transport. The generated move action is passed to the Collision and Deadlock Avoidance components that lock the required path to execute the selected action (see Fig. 4). As soon as the path is locked, the action is invoked in the local virtual environment.

Spreading fields. The local virtual environment is responsible for spreading the fields. Field management requires synchronization among local virtual environments. The spreading of fields takes into account the status of the agents such as the positions of AGVs and the priorities of transports.

5.2 DynCNET Protocol

We now explain DynCNET, a protocol-based approach for task assignment. We start by explaining a number of general properties of the protocol. Then we give an overview of the default sequence of the protocol. Next we explain how the agents dynamically can switch the assignment of tasks. We use the AGV transportation scenario depicted in Fig. 9 to illustrate the steps of the protocol. The DynCNET protocol describes communicative interactions among AGV agents and transport agents and is part of the agents' communication module, compare Fig. 4.

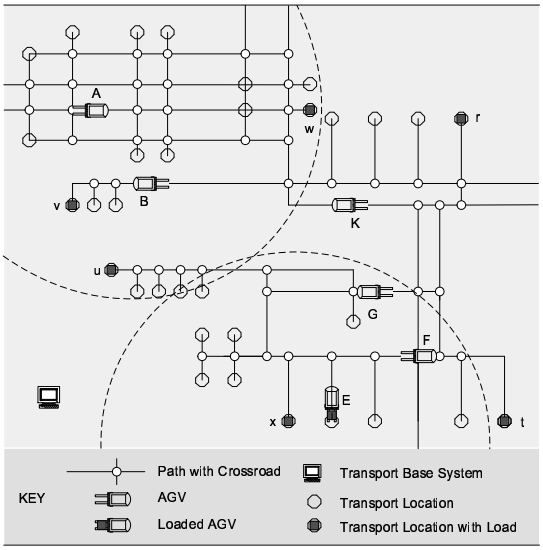

Figure 9 Scenario to illustrate DynCNET.

General Properties. DynCNET is an $m \times n$ protocol. An initiator that offers a task can interact with m participants, i.e. the candidate agents that can execute the task. On the other hand, each participant can interact with n initiators that offer tasks. In the AGV transportation system, an initiator corresponds with a transport agent that represents a task (i.e., a transport) in the system and the participant corresponds with an AGV agent that can

execute tasks. We denote the area where an initiator (or participant) searches for participants (or initiators) the *area of interest* of the initiator (or participant). The dotted circles in Fig. 9 show the current areas of interest of AGV A (top) and transport x (bottom). For transport x, there are currently two candidate AGVs to execute the transports: F and G (AGV E is delivering a load). For AGV A on the other hand, there are three possible transports to execute: u, v, and w. Due to the dynamics in the system, the set of candidate tasks (initiators) and agents that can execute a task (participants) can change over time. E.g., in the right part of Fig. 11, AGV E has just dropped its load and becomes a candidate to execute transport x.

Default Sequence. The left part of Fig. 10 shows an UML interaction diagram with the default message sequence of DynCNET. The default protocol consists of four steps: (1) the initiator sends a call for proposals; (2) the participants respond with proposals; (3) the initiator notifies the provisional winner; and finally, (4) the selected participant informs the initiator that the task is started. These four steps are basically the same as in the standard CNET protocol. The flexibility of DynCNET is based on the possible revision of the provisional task assignment between the third and fourth step of the protocol.

Switching Initiators and Participants. To explain how agents can switch tasks when the conditions in the environment change, we use the UML state diagram shown of Fig. 10. This state diagram shows a compact representation of the behavior of the initiator and participant agents in the protocol. First we look at the protocol from the perspective of the participant, then we look from the point of view of the initiator.

Switching Initiators. Consider the situation in Fig. 9 where AGV A has a provisional agreement to execute transport w. While AGV A drives toward the pick location of w, a new transport p enters the system, see the left part of Fig. 11. This new transport is an opportunity for AGV A. DynCNET enables participants to switch initiators and exploit such opportunities. When a participant is ready to execute a task, it enters the `Voting` state where it answers `cfp`'s with `proposals`. When the participant receives a `provisional–accept` message (step 3 in the interaction diagram of Fig. 10), it enters the `Intentional` state, see the right part of Fig. 10. As soon as the participant starts the task, it sends a `bound` message to the initiator to inform this latter that the execution of the task is started. The participant is then committed to execute the task. However, if a new opportunity occurs before the task is started, i.e. the participant receives a `better offer`, the participant changes to the `Switch Initiator` state and `retracts` from the provisional task assignment to switch to the more suitable task (`SwitchTask()`).

Switching Participants. Consider the situation in Fig. 9 where the transport agent x has a provisional agreement with AGV agent G. While AGV G drives toward the pick location of transport x, AGV E becomes available, see the right part of Fig. 11. This new AGV is an opportunity for transport x. DynCNET enables initiators to switch participants and exploit such opportunities. When an initiator has sent a `cfp` and received the `proposals` from the participants, it sends a `provisional-accept` message (step 3 in Fig. 10) and enters the `Assigned` state, see the right part of Fig. 10. As soon as the initiator receives a `bound` message from the selected participant it enters the state `Executing` in which the task is effectively started. However, if a new opportunity occurs before the task is started, i.e. the initiator receives a `better offer` from a participant, the initiator changes to the `Switch Participant` state. In this state the initiator sends an `abort` to the provisionally assigned participant and switches to the more suitable participant.

`TaskInScope()` and `TaskOutScope()` are functions that notify the participant when

new tasks enter and leave its area of interest. Such functionality can be provided by the perception module of the participant that monitors the area of interest of the agent in the environment. Similarly, `ParticipantInScope()` and `ParticipantOutOfScope()` notify the initiator when new participants enter and leave its area of interest. [7] elaborates on these functions in the AGV application.

Convergence. A potential risk of DynCNET is that the assignment of tasks oscillates between participants and no tasks are executed. In the AGV application, oscillations were avoided by limiting and differentiating the areas of interest for initiators (transport agents) and participants (AGV agents). In particular, the area of interests of AGV agents covered up to 1/10th of the total area of the map and the area of transport agents was 4 times smaller as that of AGV agents.

Synchronization messages. To handle synchronization, confirmation messages are used. For example, when an initiator switches participants it first sends an abort to the participant that has provisionally accepted, see the state diagram of Fig. 10. This latter then sends a message to confirm the abort.

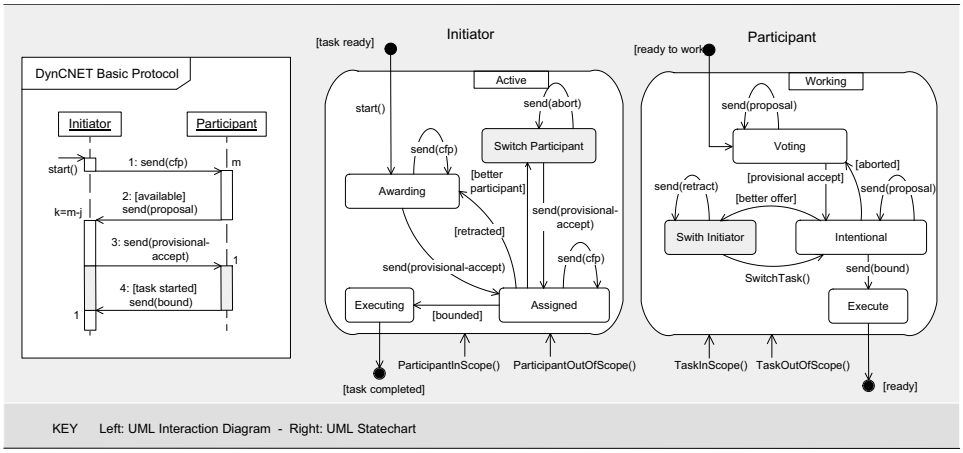

Figure 10 Left: High-level interaction diagram of DynCNET. Shaded areas in the activation boxes represent periods when agents can switch the provisional agreement. Right: Statechart diagram of DynCNET. Agents' switch the provisional agreement in the shaded states.

However, if this participant has already started the task (transition `Intentional` to `Execute`) but the initiator has not yet received the bound message, it refuses the abort. The switch will then be canceled. Due to space limitations, we made abstraction of these synchronization messages in our explanation. [23] discusses synchronization in detail.

5.3 FiTA and DynCNET Test Results

We have tested DynCNET and FiTA on the map of an AGV transportation system that is implemented by Egemin at EuroBaltic. The size of the physical layout is 134 m x 134 m. The map has 56 pick and 50 drop locations. We used a standard transport profile that Egemin uses for testing purposes. This profile generates 140 transports with a random pick location and a random drop location per hour real time. Each transport is assigned a

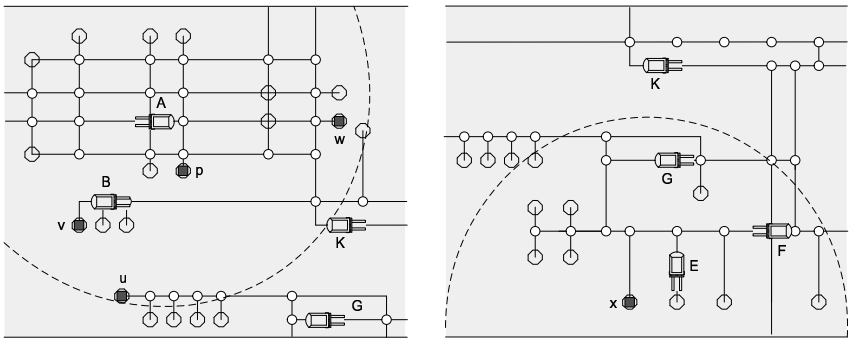

Figure 11 Left: task *p* provides an opportunity for AGV A. Right: AGV E becomes available for task *x*.

random priority that increases over time. The system uses 14 AGVs. The average driving speed of AGVs is 0.7 m/s, while load manipulations take an average time of 5 s.

For the tests, we used an AGV simulator[a]. Tests where executed on a cluster of 40 machines: P4 2Ghz, 512MB RAM, Debian Stable 3.0. Every simulation was run for 200.000 timesteps, corresponding to approximately 4 hours real time, i.e. one timestep represents 20 ms in real time. All displayed test results are average values over 20 to 40 simulation runs.

5.3.1 Test Results

We have measured communication load (number of messages sent per transport), reaction time (average waiting time per transport as a function of simulated timesteps), and throughput (number of finished transports as a function of simulated timesteps). Since taks are generated randomly and priorities are assigned randomly, we have verified the statistic significance we have calculated 95 % confidence intervals [27] denoted with error bars in the graphs. The small intervals indicate that the test results are sufficiently reliable.

Communication Load. To compare the communication load, we have measured the average number of messages sent per finished transport. The left part of Fig. 12 shows the results of the test. The number of messages of DynCNET and FiTA are approximately the same, while the communication load of CNET is about half of the load of the dynamic mechanisms.

Average Waiting Time and Throughput. The right part of Fig. 12 shows the test results for average waiting time for transports. Average waiting time is expressed as the number of timesteps a transport has to wait before an AGV picks up the load. After a transition period (around 20.000 timesteps), DynCNET and FiTA outperform CNET. The difference increases when time elapses. FiTA is slightly better than DynCNET over the full test range. A possible explanation is that idle AGVs in FiTA immediately start moving when they sense a field of a task, while in DynCNET AGVs only start moving after they are provisionally committed to execute a task.

After four hours in real-time, the throughput of CNET was 380 transports, DynCNET

[a]AGV Simulator, 2007, AgentWise, DistriNet Labs, Katholieke Universiteit Leuven, www.cs.kuleuven.ac.be/∼distrinet/taskforces/agentwise/agvsimulator/

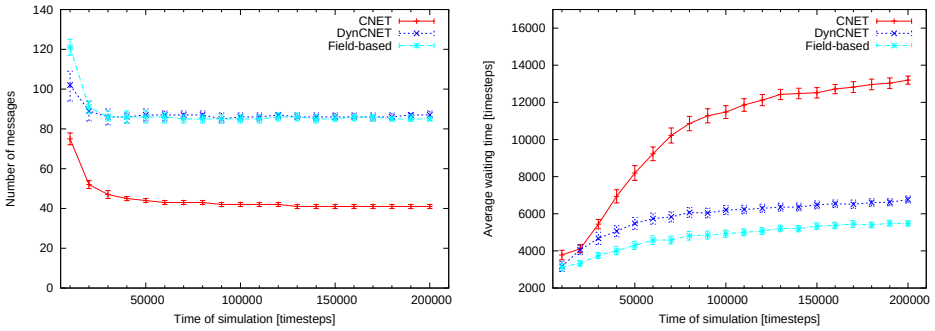

Figure 12 Left: amount of messages being sent per finished transport. Right: average waiting time

has handled 467 transports, and FiTA 515 transports. For the 467 executed transports of DynCNET, we measured an average of 414 switches of transport assignments performed by transport agents and AGV agents.

5.3.2 Discussion

DynCNET and FiTA have similar performance characteristics. Both outperform CNET on all performance measures, the cost is a doubling of required bandwidth. Both DynCNET and FiTA support flexible assigning of tasks. In FiTA, the choices of the AGV agents are implicitly determined by the combination of the sensed fields. DynCNET provides explicit points of choice for transport and AGV agents. We used the priorities of transports and the distance between AGVs and loads to switch transports. More advanced approaches can be considered, e.g., AGV agents may (to some extent) favour transports that are located near to one another increasing the change to find a closely located transport when the original assignment for some reason switches. Both DynCNET and FiTA support openness, i.e. both mechanisms allow initiators to take into account new participants that become available and vice versa. Whereas FiTA inherently supports openness (the combination of fields adapts when fields disappear or new fields appear), the DynCNET protocol includes explicit functions (ParticipantInScope, etc.) that notify initiators and participants when other agents enter or leave their area of interest. Neither flexibility, nor openness is supported by CNET.

Robustness to Message Loss. Robustness to message loss is the ability of a task assignment approach to withstand message loss (i.e., graceful degrade). In FiTA, the freshness of the received fields is taken into account to determine the attraction and repulsion of fields. When an agent misses an update of a field due to the loss of a message, the previous values (with less importance) is used to calculate the combined field. As such, FiTA is (to some degree) robust to message loss. DynCNET (as CNET) on the other hand fails when a message gets lost and the prescribed sequence of messages is disrupted. As such, DynCNET require additional support for robustness to message loss. Exception handling in protocol design is a non-trivial problem [18].

Mechanism Engineering. No common engineering approaches are currently available for designing and developing FiTA. On the other hand, DynCNET allows to specify the behav-

ior of the agents by means of common engineering diagrams such as interaction diagrams and statecharts, allowing engineers to reason on the assignment of tasks. Parameter tuning of DynCNET and FiTA requires similar efforts.

The tradeoff between robustness to message loss and engineering comfort is an important criteria for selecting a task assignment approach in practice.

6 COLLISION AVOIDANCE

We now explain how AGVs avoid collisions. In the centralized approach, collision avoidance is realized as follows: for each AGV in the system, a series of hulls are calculated that represent the physical area the AGV occupies along the path it is going to drive. A *hull projection* projects a hull over a part of the path the AGV intends to drive on. When two or more hull projections overlap, AGVs are in *collision range* and all except one AGV are commanded to wait.

6.1 Decentralized Mechanism for Collision Avoidance

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 local virtual environments take the burden of coordination. Fig. 13 shows a series of screenshots of a simulation run in a realistic map. In Fig. 13(a), two AGVs, A and B, are approaching one another. Both AGVs are projecting hulls in the environment. At this point, no conflict is detected (we explain the circles below). In Fig. 13(b), AGV B has projected further ahead, and is now in conflict with the hull projection of AGV A. If we assume that AGV A has already reserved the trajectory occupied by its hull, AGV A is given priority to AGV B that must wait. In 13(c), AGV A is taking the curve, passing AGV B. Finally, in 13(d), AGV A has parked at the bottom, and AGV B is 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 selects the path it intends to follow over the layout and determines how much of this path it wants to lock. This is determined by LockAheadDistance parameter[b].
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 local virtual 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

[b] Besides the LockAheadDistance, the AGV also applies basic rules for deadlock avoidance such as locking a bi-directional path until the end. Yet, we do not further elaborate on deadlock avoidance here.

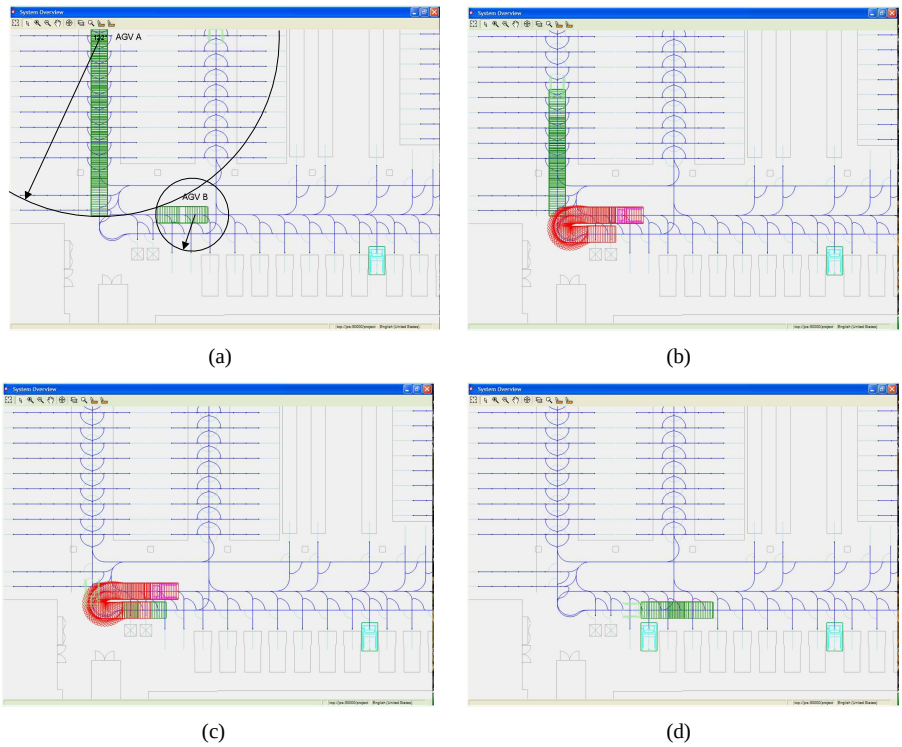

Figure 13 (a) Two AGVs approaching, (b) A conflict is detected, (c) One AGV passes safely, (d) The second AGV can pass as well. The hull projection circles in (a) are used to determine the AGVs in collision range.

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 local 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. A solution to this problem is shown in Fig. 13(a). 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 determined by the furthest point of the AGV's hull projection. If two such circles overlap, this indicates 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 virtual environments of all requested AGVs.

2. The local virtual environments of requested AGVs check whether the projection overlaps with their hull projection. There are three possibilities for each of the requested AGVs. (a) If 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) If 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) If the requesting AGV's hull projection overlaps with the requested AGV's hull projection, and the requested AGV's hull is not locked, ties are broken by the priorities of the hulls, i.e. the local virtual environment of the requested AGV replies "allowed" if the priority of its hull is lower than the hull of the requesting AGV and 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 state of 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. The approach used for collision avoidance shows how the virtual environment serves as a flexible coordination medium for the agents.

6.2 Software Architecture: Communicating Processes for Collision Avoidance

We now illustrate how collision avoidance is dealt with in the software architecture. Fig. 14 shows the communication processes view for collision avoidance.

The communicating processes view presents the basic modules of the AGV control system and overlay them with the main processes and repositories involved in collision avoidance. We discuss the main architectural elements involved in collision avoidance in turn.

The **Perception Process** is part of the agent's perception component (see Fig. 4). If the perception process receives a request for perception, it requests the up-to-date data from the local virtual environment and updates the agent's current knowledge.

The **Perception Generator Process** is part of the perception manger (see Fig. 5). This process is responsible for handling perception requests, it derives the requested data from the state repository of the local virtual environment according to the given foci. Current state of the AGV vehicle and state from other nodes that is needed by the AGV agent is maintained by dedicated synchronization processes.

Collision Avoidance Process is part of the AGV agent's decision making component. The collision avoidance process calculates the required hull projection for collision avoidance, and projects the hull in local virtual environment. Once the hull is locked, the collision avoidance process invokes a move command in the local virtual environment.

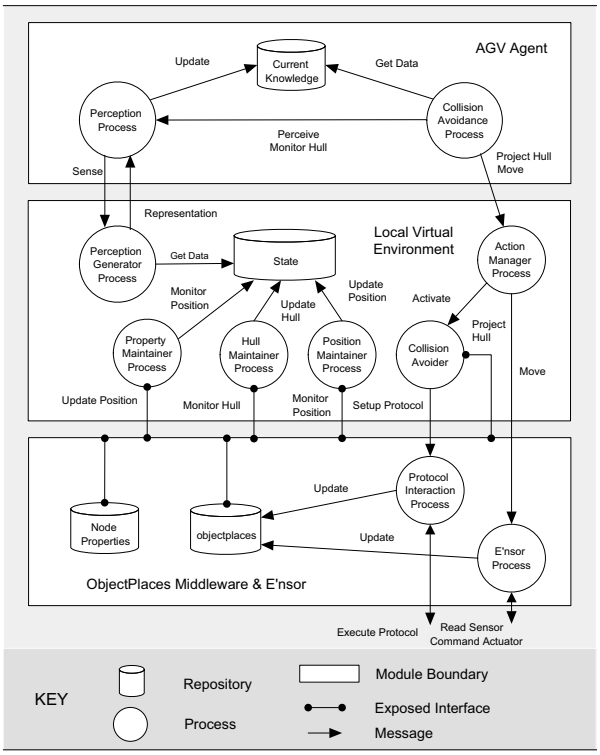

Figure 14 Communicating processes for collision avoidance

The **Action Manager Process** is part of the action manager component. The action manager process collects the actions invoked in the local virtual environment and dispatches them to the applicable processes. For a hull projection, the action manager process passes the actions to the collision avoider process of the local virtual environment. A move action is passed to the E'nsor process.

Objectplaces repository is a repository of data objects in the ObjectPlaces middleware (see Fig. 3) that contains the hulls the AGV agent has requested.

NodeProperties is a data repository in the middleware in which relevant properties of the node are maintained, an example is the AGV's current position. Maintenance of node properties in the repository is handled by the **Property Maintainer Process**. This process is a synchronization process of the local virtual environment that is part of the synchronization component (see Fig. 5). The data objects of the NodeProperties repository are used by the middleware to synchronize the state among local virtual environment on neighboring nodes. For example, the current position in the node properties repository is used by the ObjectPlaces middleware to determine whether the AGV is within collision range of other AGVs.

The **Collision Avoider** is a helper process of the action manager process that projects the requested hull in the objectplaces repository and initiates the collision avoidance protocol in the middleware.

The **Protocol Interaction Process** is part of ObjectPlaces and is responsible for executing the mutual exclusion protocol for collision avoidance. This process maintains the state of the agent's hull in the objectplaces repository.

The **Hull Maintainer and Position Maintainer Processes** are part of the synchronization component. The hull maintainer process monitors the hull object in the objectplaces repository and keeps the state of the hull in the state repository of the local virtual environment consistent. The position maintainer process maintains in a similar way the actual position of the vehicle.

Finally, the **E'nsor Process** is part of E'nsor (see Fig. 3). The E'nsor process (1) periodically provides updates of the vehicles physical state (such as position and battery status), and (2) translates the high-level actions from the action manager process into low-level commands for the vehicle actuators.

7 ARCHITECTURE EVALUATION

For the evaluation of the software architecture of the AGV transportation system we used the Architecture Tradeoff Analysis Method (ATAM) [7, 6]. The main goal of the ATAM is to examine whether the software architecture satisfies system requirements, in particular the quality requirements. We applied the ATAM for one concrete application, in casu a tobacco warehouse transportation system that was used as a test case in the project. We start this section with a brief overview of the ATAM evaluation and an introduction of the tobacco warehouse application. Then, we discuss the analysis of architectural approaches for two important quality scenarios of the system.

7.1 ATAM Workshop

In preparation to the ATAM evaluation, three stakeholders together with one of the evaluators held a four-days Quality Attribute Workshop (QAW [3]). A QAW is a facilitated method that engages stakeholders to discover the driving quality attributes of a software-intensive system. During the QAW, the participants developed a utility tree [11]. A utility tree characterizes the important quality requirements in a four-level tree structure where each level provides more specific information about important quality goals, with leaves specifying measurable quality attribute scenarios. Each scenario is assigned a ranking that expresses its priority relatively to the other scenarios.

The ATAM itself was conducted by a team of three evaluators and nine stakeholders, including a project manager, two architects, a project engineer, two developers, a service and a simulation engineer, and a representative for the customer. The workshop took one day and followed the standard ATAM phases, i.e., presentations of ATAM, business drivers, architecture and architectural approaches. Next the quality attribute utility tree was discussed with the stakeholders and two quality scenarios were analyzed in detail (see below). The workshop initiated a number of additional activities. A number of tests were conducted to investigate the main risks that were identified during the workshop. An extra analysis of risks and tradeoffs of the software architecture was performed with a reduced number of stakeholders. Finally, the architects finished the architecture documentation and the evaluators presented the main workshop results. [6] presents the integral report.

7.2 Tobacco Warehouse Transportation System

In the tobacco application, AGVs have to bring bins with tobacco to different processing machines and storage locations. The warehouse measures 75 x 55 meters with a layout of approximately 6000 nodes. The installation provides 12 AGVs that use navigation with magnets in the fbor. AGVs use opportunity charging and a 11 Mbps wireless ethernet is available for communication. Transports are generated by a warehouse management system at an average load of 120 transports/hour.

The system is subject to a number of technical constraints, including the use of .NET as development framework, backwards compatibility with E'pia the general purpose framework developed by Egemin that provides basic support for persistency, security, logging, etc., and compatibility with E'nsor the low-level control software deployed on AGVs. Finally, the load of the wireless network is restricted to 60 % of its full capacity.

7.3 Analysis of Architectural Approaches

During the ATAM, the architectural approaches that address the high-priority quality scenarios were elicited and analyzed. A number of architectural risks (i.e. problematic architectural decisions), sensitivity points (i.e. architectural decision that involve architectural elements that are critical for achieving the quality attributes), and tradeoff points (i.e. architectural decisions that affect more than one attribute) of the software architecture were identified. We give an overview of two important quality attribute scenarios that were analyzed: one scenario concerning flexibility (transport assignment) and another scenario concerning performance (bandwidth usage).

7.3.1 Architectural Analysis of Flexibility

Fig. 15 shows an overview of the analysis of architectural decisions for the main quality attribute scenario of flexibility (F2.1).

Analysis of Architectural Approach			
Scenario #: F2.1	As long as a transport has not been picked up, the system dynamically changes that transport's assignment to the most suitable AGV.		
Attributes	Flexibility		
Environment	Normal operation		
Stimulus	A transport has not been picked up and the transport's assignment can be improved.		
Response	The system dynamically changes the assignment of the transport to the most suitable AGV.		
Architectural decisions	**Sensitivity**	**Tradeoff**	**Risk**
AD 1 Negotiating agents		T1	
AD 2 Locality	S1		
AD 3 DynCNET protocol for transport assignment			R1

Figure 15 Analysis architectural approaches for a flexibility scenario

The table shows the main architectural decisions (AD) that achieve the quality attribute scenario, and specifies sensitivity points, tradeoffs, and risks associated with the architectural decisions. We briefly explain the various architectural decisions:

AD 1 To assign transports, multiple AGV agents negotiate with multiple transport agents. Agents continuously reconsider the changing situation, until a load is picked. The continuous reconsideration of transport assignments improves the flexibility of the system. However, it also implies a significant increase of communication. This was registered as tradeoff T1.

AD 2 For decision making, agents take only into account local information in the environment. The most suitable range varies per type of information, and can vary over time for particular types of information, e.g., candidate transports, vehicles to avoid collisions, etc. The determination of this range for various functionalities is a sensitivity point. This sensitivity point was denoted as S1.

AD 3 The DynCNET protocol is documented at a high-level of abstraction. At the time of the ATAM, several important decisions were not taken yet. The difficulty of parameter tuning to ensure convergence and optimal behavior was unclear. This lack of clearness was registered as risk R1.

7.3.2 Architectural Analysis of Bandwidth Usage

Fig. 16 shows an overview of the analysis of architectural decisions for the main quality attribute scenario of bandwidth usage (P2.1). We give a brief explanation of the various architectural decisions:

Analysis of Architectural Approach			
Scenario #: P2.1	The amount of communication, with maximal 12 AGVs and a maximal load of 140 transporta per hour, does not exceed 60 % of the bandwidth of the 11 Mbps communication channel.		
Attributes	Performance		
Environment	All operation modes with maximal 12 AGVs and a maximal load of 140 transports per hour.		
Stimulus	Communication among subsystems.		
Response	Communication load should not exceed 60 % of the bandwidth of the 11 Mbps communication channel.		
Architectural decisions	**Sensitivity**	**Tradeoff**	**Risk**
AD 1 Choice for .NET remoting	S2		
AD 2 Agents located on machine controls AGV		T2	R2
AD 3 DynCNET protocol for transport assignment		T3	
AD 4 Two step deadlock prevention mechanism			R3
AD 5 Unicast communication in the middleware	S3		

Figure 16 Analysis architectural approaches for a bandwidth usage scenario

AD 1 The AGV transportation system software is built on top of the .NET framework. This

choice was a business constraint but also an evident choice since the E'pia library that is used for logging, persistence, security, etc., also uses .NET. The overhead induced by the choice for the point-to-point communication approach of .NET remoting was registered as a sensitivity point S2.

AD 2 Each AGV vehicle is controlled by an agent that is physically deployed on the machine. This decentralized approach induces a risk with respect to the required bandwidth for inter-agent communication. This was recorded as risk R2. An AGV agent can flexibly adapt its behavior to dynamics in the environment. AGVs controlled by autonomous agents can enter/leave the system without interrupting the rest of the system. However, flexibility and openness comes with a communication cost. This tradeoff was noted as T2.

AD 3 The DynCNET protocol for transport assignment enables flexible assignment of transports among AGVs. Yet, the continuous reconsideration of transport assignment implies a communication cost. This tradeoff was denoted as T3.

AD 4 AGV agents use a two phase deadlock prevention mechanism. AGV agents first apply static rules to avoid deadlock, e.g. agents lock unidirectional paths over their full length. These rules however, do not exclude possible deadlock situations completely. If an agent detects a deadlock, it contacts the other involved agents to resolve the problem. Yet, the implications of the deadlock mechanism on the communication overhead are at the time of the ATAM not fully understood. This lack of insight was denoted as risk R3.

AD 5 The ObjectPlaces middleware uses unicast communication. However, some messages have to be transmitted to several agents, causing overhead. Support for multicast is possible, yet, this implies that the basic support of .NET remoting would no longer be usable. This potential problem was registered as sensitivity point S3 (see also S2).

7.3.3 Testing Communication Load

The analysis of the architectural approaches improved the understanding of the tradeoff between flexibility and communication load. To further investigate this tradeoff, we conducted a number of tests after the ATAM workshop. Besides the simulation tests of the two approaches for transport assignment (see section 5), we tested the efficiency of the middleware in the AGV application by measuring bandwidth usage of a system in a real factory layout.

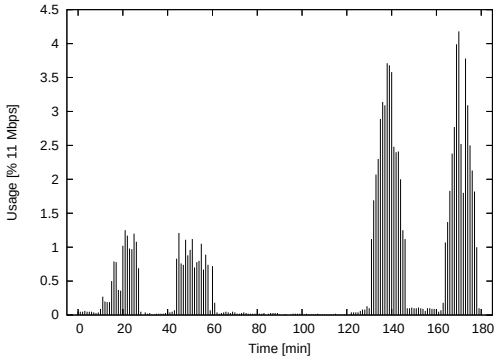

Figure 17 Bandwidth usage in a test setting

Fig. 17 shows the results of four consecutive test runs. We measured the amount of data sent over the network by each AGV, and averaged this per minute to obtain the bandwidth

usage relative to the bandwidth of a 11 Mbps IEEE 802.11 network. The first test (Time: 10–30 min.) has three AGVs, of which two were artificially put in deadlock (a situation which is avoided in normal operation), because then the collision avoidance protocol is continually restarted, and never succeeds. This is a peak load of the system. The second test (40–60 min.) has three AGVs driving around freely. The third test (130–150 min.) has five AGVs driving around freely. The fourth test (160–180 min.) has five AGVs, all artificially put in deadlock. During the time in between test runs, AGVs were repositioned manually. On average, the bandwidth usage doubles when going from three to five AGVs. This is because the AGVs need to interact relatively more to avoid collisions. Based on these test results, Egemin experts consider the bandwidth usage acceptable for an extrapolation to 12 AGVs, taking into account a maximal bandwidth of 60 % of 11 Mbps, and given that bandwidth optimizations were not applied yet.

7.4 Demonstrator of AGV Transportation System

As a proof of concept, we have developed a demonstrator of the decentralized AGV transportation system. The demonstrator with two AGVs is developed in .Net and supports the basic functionality for executing transport orders. The core of the action selection module of the AGVs is set up as a free-fbw tree. A monitor enables remote access of the AGVs and generates a fusion view that represents the status of the local virtual environments of both AGVs. Fig. 18 shows a snapshot of the AGVs in action with the fusion view.

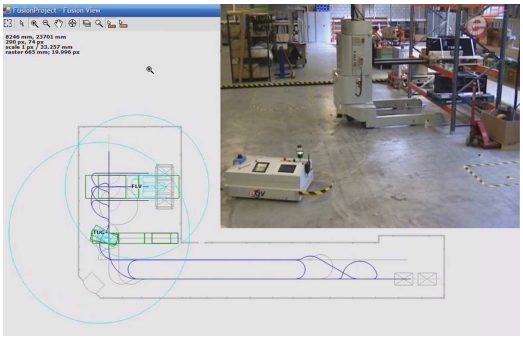

Figure 18 Demonstrator with AGVs in action

More information and demonstration movies of the prototype implementation can be found at the project website [13].

7.5 Applicability of the Software Architecture to Other Domains

The software architecture presented in this paper is developed for AGV transportation systems. In this section, we explain how parts of this particular software architecture can be transferred to other contexts. On the other hand, we also identify important issues that constrain the practicality of the architecture to other domains.

Two important properties of AGV transportation systems are: (1) there is an inherent distribution of resources and activity in the system, and (2) the control software has to

operate in a highly dynamic environment. Domains that share these properties are peer-to-peer applications (such as file sharing systems), manufacturing control, traffic monitoring and control, and large-scale wireless sensor networks.

We discuss the applicability of parts of the software architecture to other domains from two perspectives. First we explain how the notion of virtual environment can provide a means to tackle complexity. Next, we discuss how a number of architectural approaches to achieve flexibility and openness can help to deal with the dynamics in the system.

Managing Complexity. The most crucial architectural decision to manage complexity in the AGV transportation system is the introduction of the virtual environment. The virtual environment allows the separation of two concerns: controlling the AGVs by means of selecting proper actions, and managing the coordination among the vehicles. The concept of a virtual environment as an active entity in the design of a decentralized system is an architectural approach that can be translated to other related domains.

Flexibility and Openness. The choice for a situated MAS is central to the way flexibility and openness are achieved in the software architecture of the AGV transportation system. A number of important architectural approaches to achieve these qualities are: separating decision making from communication in the agent architecture, task assignment with DynCNET and FiTA, and the ObjectPlaces middleware.

Separating decision making from communication in the software architecture of the agents allows both functions to act in parallel and at a different pace. This architectural approach supports flexible coordination among agents and can be useful in domains where the speed of action selection in the environment is orders of magnitude lower than the speed of communication.

Task assignment in a decentralized architecture is a complex coordination problem. We presented DynCNET and FiTA as two alternative solutions. Both these approaches are suitable for domains that are characterized by delayed commencement of tasks, i.e. an agent that has to execute a task has to perform a significant effort before it can effectively execute that task. Two additional assumptions are: the environment needs to provide continual communication access, and there should be enough agents to execute the tasks that enter the system, i.e. the agents can handle the load of the system. We believe that DynCNET is applicable in other domains that share these properties. For FiTA, there is one important additional constraint. FiTA requires that the strength of the fields can be expressed proportional to the real distance between tasks and agents rather than to the Euclidian distance. This constraint ensures that agents that use FiTA do not get stuck in local minima. As such, the approach is less suitable for domains where agents are less restricted in their movements in space.

Finally, the ObjectPlaces middleware provides support for gathering and maintaining context information and setting up protocol-based communication in mobile and ad-hoc networks. An important part of this middleware was developed in the context of the EMC^2 project, however, the middleware is independent of the AGV transportation system and as such can be useful in other related domains as well.

8 RELATED WORK

AGV control is subject of active research since the mid 1980s. Most research has been conducted in the domain of AI and robotics. Recently, a number of researchers have ap-

plied MAS, yet, most of this work is applied in small-scale projects.

AI and Robotics Approaches. The problems of routing and scheduling of AGVs is different from conventional path finding and scheduling problems. Scheduling and routing of AGVs is a time-critical problem, while a graph problem usually is not. Besides, the physical dimensions of the AGVs and the layout of the map must be taken into account.

Roughly spoken, three kinds of methods are applied to solve the routing and scheduling problem. Static methods use a shortest path algorithm to calculate routes for AGVs, see e.g. [12]. In case there exists an overlap between paths of AGVs, only one AGV is allowed to proceed. The other AGVs have to wait until the first AGV has reached its destination. Such algorithms are simple, but not efficient. Time-window-based methods, maintain for each node in the layout a list of time-windows reserved by scheduled AGVs. An algorithm routes vehicles through the layout taking into account the reservation times of nodes, see e.g. [16]. Dynamic methods apply incremental routing. An example algorithm is given in [25]. This algorithm selects the next node for the AGV to visit (towards its destination) based on the status of the neighboring nodes (reserved or not) and the shortest travel time. This is repeated until the vehicle reaches its destination. Measurements show that the algorithm is significant faster than non-dynamic algorithms, yet, the calculated routes are less efficient.

Contrary to the decentralized approach we have applied in the EMC^2 project, traditional scheduling and routing algorithms usually run on a central traffic control system from where commands are dispatched to the vehicles [20]. Moreover, most approaches are intended to find an optimal schedule for a particular setting. Such approaches are very efficient when the tasks are known in advance as for example the loading and unloading of a ship in a container terminal. In our work, scheduling and routing are going concerns, with AGVs operating in a highly dynamic environment.

Multiagent System Approaches. [19] presents a decentralized approach for collision-free movements of vehicles. In this approach, agents use cognitive planning to steer the AGVs through the warehouse layout. [5] discusses a behavior-based approach for decentralized control of automatic guided vehicles. In this work, conflict resolution with respect to collision and deadlock avoidance is managed by the agents based on local information. In [17], Lindijer applies another agent-based approach to determine conflict-free routes for AGVs. The author motivates his approach by considering quality requirements, including safety, flexibility, and scalability. Central to the approach is the concept of semaphore that is used as a traffic control component that guards shared infrastructure resources in the system such as an intersection. The system is validated with simplified scale models of real AGVs.

Arora and his colleagues have published a number of papers that describe the control of AGV systems with an agent-based decentralized architecture [1, 2]. Vehicles select their own routes and resolve the conflicts that arise during their motion. Control laws are applied to find save conditions for AGVs to move. [8] discusses a variation on the field-based approach where agents construct a field in their direct neighborhood to achieve routing and deadlock avoidance in a simplified AGV system. Hoshino et al. [15] study a transportation system in which cranes unload a container ship and pass the loads to AGVs that bring them to a storage depot. Each AGV and crane is represented by an agent. The authors investigate various mechanisms for AGV agents to select a suitable crane agent. Off-line simulations allow to determine the optimal vehicle combination for a particular throughput. Such approach is restricted to domains where no disturbances are expected.

Contrary to our research, the discussed agent-based approaches are only validated in

simulations and under a number of simplifying assumptions. Applying decentralized control in a real industrial setting involves numerous complicating factors that deeply affect the scheduling and routing of AGVs. Most of the related work focusses on isolated concerns in AGV control. For a practical application however, different concerns have to be integrated, which is not a trivial problem. One lesson we learned is that communication is a major bottleneck in a decentralized AGV control system. Most related work only considers simple layouts with a small number of AGVs, and abstracts from communication costs. An important difference between our research and the discussed approaches is that we have applied an architecture-centric design for the AGV application in the EMC^2 project. Scheduling and routing are integrated in the software architecture with other concerns such as deadlock avoidance and maintenance of the AGVs. Most related work does not consider software architecture explicitly. As a consequence, little attention is payed to the tradeoffs between qualities. In the EMC^2 project the tradeoffs between quality goals were the drivers for the system design.

9 CONCLUSIONS

In this paper, we gave an overview of the architectural design of situated MAS for AGV control. The AGV and transport agents that coordinate through a virtual environment allowed us to shape the software architecture of the transport application to provide the required functionalities of the system and achieve the important quality goals flexibility and openness. We conclude this paper with some lessons we learned from applying MAS in this complex real-world application.

Qualities and tradeoffs. A main motivation for applying a MAS to the AGV transportation system was to investigate whether the decentralized architecture could improve flexibility and openness. Obviously, an industrial AGV transportation system is subject to various quality requirements and business constraints. The decentralized architecture introduces new tradeoffs between the various system requirements. Important lessons we learned are (1) the motivation to apply a MAS should be driven by quality goals; (2) considering MAS from a software architecture point of view compels stakeholders to deal explicitly with tradeoffs between quality requirements from the early start of a project.

Integration with legacy systems. Most industrial software systems require an integration with legacy systems, and this was the case for the AGV transportation system as well. We reused various parts of the existing AGV control software in the decentralized architecture, examples are the low-level control software for AGVs and the layout of maps. This saved a lot of work. Lessons we learned are: (1) the integration with legacy software is a matter of fact when agent technology is applied in an industrial setting; (2) software architecture provides the means to reason about, and deal with the integration of legacy software in an agent-based system.

Stepwise integration. Switching from a centralized architecture toward an agent-based decentralized architecture is a big step with far reaching effects for the company, not only for the software but for the whole organization. A lesson we learned is: integration of an agent-based approach should be done in a controlled way, step-by-step. At the time of writing this paper, Egemin is re-factoring the basic AGV control architecture and as a fist step plans to integrate one of the adaptive transport assignment approaches in the architecture.

Evaluation. The evaluation of the software architecture contributed to a better un-

derstanding of the strengths and weaknesses of the decentralized architecture. Besides qualities, the functional behavior of the system must be evaluated. It is well-known that giving guarantees about the global behavior of a decentralized system is hard. Lessons we learned are: (1) a disciplined evaluation of the software architecture of the agent-based system is invaluable; (2) debugging a decentralized system is hard; (3) simulations are the main vehicle to give (to a certain extent) guarantees about global system properties.

The connection between software architecture and MAS provides a promising venue for future research. In this paper, we have put forward flexibility and openness as important qualities to apply situated MAS. However, MAS are generally considered to be useful for other qualities as well, such as robustness and scalability. It would be interesting to investigate how these qualities translate to architectural approaches and how the qualities tradeoff with other qualities in the system.

ACKNOWLEDGMENTS

Danny Weyns is supported by the Research Funding of the Katholieke Universiteit Leuven and the Funding for Scientific Research in Flanders (FWO). We are grateful to Kurt Schelfthout, Jan Wielemans, Tom Lefever, Nelis Boucké, Alexander Helleboogh, and the anonymous reviewers, for the valuable feedback on earlier versions of this paper.

References and Notes

1 S Arora, A. Raina, and A. Mittal. Collision Avoidance Among AGVs at Junctions. In *IEEE Intelligent Vehicles Symposium*, 2000.

2 S Arora, A. Raina, and A. Mittal. Hybrid Control in Automated Guided Vehicle Systems. In *IEEE Conference on Intelligent Transportation Systems*, 2001.

3 M. Barbacci, R. Ellison, A. Lattanze, J. Stafford, C. Weinstock, and W. Wood. uality Attribute Workshops. Technical Report CMU/SEI-2003-TR-016, Software Engineering Institute, Carnegie Mellon University, PA, USA, 2003.

4 L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, 2003.

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

6 N. Boucké, T. Holvoet, T. Lefever, R. Sempels, K. Schelfthout, D. Weyns, and J. Wielemans. Applying the ATAM to an Industrial Multiagent System Application. In *CW Report 431*. Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 2005.

7 N. Boucké, D. Weyns, K. Schelfthout, and T. Holvoet. Applying the ATAM to an Architecture for Decentralized Control of a AGV Transportation System. In *2nd International Conference on Quality of Software Architecture, QoSA, Lecture Notes in Computer Science, Vol. 4214*, Vasteras, Sweden, 2006. Springer.

8 L. Breton, S. Maza, and P. Castagna. Simulation multi-agent de systèmes d'AGVs: comparaison avec une approche prédictive. 5^e *Conférence Francophone de Modélisation et Simulation*, 2004.

9 F. Buchmann and L. Bass. Introduction to the Attribute Driven Design Method. In *23rd International Conference on Software Engineering*, Toronto, 2001. IEEE Computer Society.

10 P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison Wesley, 2002.

11 P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison Wesley Publishing Comp., 2002.

12 D. Daniels. Real-time Conflict Resolution in Automated Guided Vehicle Scheduling. *PhD Dissertation: Dept. of Industrial Eng., Penn. State University, USA*, 1988.

13 EMC^2: Egemin Modular Controls Concept, Project Supported by the Institute for the Promotion of Innovation Through Science and Technology in Flanders, (8/2006) *http://emc2.egemin.com/*

14 P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):28–29, 1968.

15 S. Hoshino, J. Ota, A. Shinozaki, and H. Hashimoto. Design of an AGV Transportation System by Considering Management Model in an ACT. *Intelligent Autonomous Systems*, 9:505–514, 2006.

16 C. Kim and J. Tanchoco. Operational Control of a Bi-directional Automated Guided Vehicle Systems. *Production Research*, 31(9):2123–2138, 2002.

17 D. Lindeijer. Controlling Automated Traffic Agents. *PhD Dissertation: University of Delft, The Netherlands*, 2003.

18 A. Mallya and M. Singh. Modeling Exceptions via Commitment Protocols. In *Fourth International Joint Conference on Autonomous agents and multiagent systems*, New York, NY, USA, 2005. ACM Press.

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

20 L. Qiu, W. Hsu, S. Huang, and H. Wang. Scheduling and Routing Algorithms for AGVs: A Survey. *Production Research*, 40(3), 2002.

21 K. Rosenblatt and D. Payton. *A Fine Grained Alternative to the Subsumption Architecture for Mobile Robot Control*. International Joint Conference on Neural Networks, 1989.

22 K. Schelfthout. Supporting Coordination in Mobile Networks: A Middleware Approach. In *Ph.D Dissertation, Katholieke Universiteit Leuven, Belgium, 2006.*

23 W. Schols, N. Boucké, D. Weyns, and T. Holvoet. *DynCNET: A Protocol for Flexible Transport Assignment in AGV Transportation Systems*. CW Report 478, Katholieke Universiteit Leuven, Belgium, 2007.

24 R. Smith. The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. In *IEEE Transactions on Computers, C-29(12)*, 1980.

25 F. Taghaboni and J. Tanchoco. Comparison of Dynamic Routing Techniques for Automated Guided Vehicle Systems. *Production Research*, 33(10):2653–2669, 1995.

26 T. Tyrrell. *Computational Mechanisms for Action Selection*. PhD Dissertation, University of Edinburgh, 1993.

27 E. Weisstein. Confidence Intervals, MathWorld (2/2007). *http://mathworld.wolfram.com/Confi denceInterval.html*.

28 D. Weyns. An Architecture-Centric Approach for Software Engineering with Situated Multiagent Systems. In *Ph.D Dissertation, Katholieke Universiteit Leuven, Belgium, 2006.*

29 D. Weyns, N. Boucké, and T. Holvoet. Gradient Field Based Transport Assignment in AGV Systems. In *5th International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS, Hakodate, Japan*, 2006.

30 D. Weyns, A. Omicini, and J. Odell. Environment as a First-Class Abstraction in Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.

31 D. Weyns, K. Schelfthout, and T. Holvoet Decentralized Control of E'GV Transportation Systems. In *4th Joint Conference on Autonomous Agents and Multiagent Systems, Industry Track*, Utrecht, The Netherlands, 2005. ACM Press.