Decentralized Control of Automatic Guided Vehicles

Applying Multi-Agent Systems in Practice

Danny Weyns Tom Holvoet

DistriNet Labs, Katholieke Universiteit Leuven, Belgium
http://distrinet.cs.kuleuven.be/
{danny.weyns,tom.holvoet}@cs.kuleuven.be

Kurt Schelfthout Jan Wielemans

Egemin International nv
http://www.egemin.com/
{kurt.schelfthout,jan.wielemans}@egemin.be

Abstract

An automatic guided vehicle (AGV) transportation system is a fully automated system that provides logistic services in an industrial environment such as a warehouse or a factory. Traditionally, the AGVs that execute the transportation tasks are controlled by a central server via wireless communication. In a joint effort between Egemin, an industrial manufacturer of AGV transportation systems, and DistriNet Labs research at the Katholieke Universiteit Leuven, we developed an innovative decentralized architecture for controlling AGVs. The driving motivations behind decentralizing the control of AGVs were new and future quality requirements such as flexibility and openness. At the software architectural level, the AGV control system is structured as a multi-agent system; the detailed design and implementation is object-oriented. In this paper, we report our experiences with developing the agent-based control system for AGVs. Starting from system requirements, we give an overview of the software architecture and we zoom in on a number of concrete functionalities. We reflect on our experiences and report lessons learned from applying multi-agent systems for real-world AGV control.

Categories and Subject Descriptors D.2.11 [*Software Engineering*]: Software Architectures; I.2.11 [*Distributed Artificial Intelligence*]: Multiagent systems

General Terms design

Keywords decentralized control, software architecture, multi-agent system, automatic guided vehicle, AGV

OOPSLA'08, October 19–23, 2008, Nashville, Tennessee, USA.
ACM 978-1-60558-220-7/08/10.

1. Introduction

An AGV transportation system consists of a number of unmanned vehicles that need to work together to transport loads in an industrial environment. Transports are generated by client systems, typically an enterprise resource planning system and possibly operators. The main functionalities that an AGV transport system has to fulfill is assigning incoming transport tasks to appropriate AGVs, routing the AGVs through the warehouse efficiently while avoiding collisions and deadlocks, and maintaining the AGVs' batteries.

An AGV transportation system has to deal with dynamic and changing operating conditions. The stream of transports that enter the system is typically irregular and unpredictable, AGVs can leave and re-enter the system for maintenance, production machines may have variable waiting times, etc. All kinds of disturbances can occur, supply of goods can be delayed, certain areas in the warehouse may temporarily be closed for maintenance services, loads can block paths, AGVs can fail, etc. Despite these challenging operating conditions, the system is expected to operate efficiently and robustly.

Egemin has successfully been providing full life cycle support for AGV systems used to automate internal logistics in production, distribution and warehousing environments since the seventies. Traditionally, the AGVs systems deployed by Egemin are directly controlled by a central server. The server plans the schedule for the system as a whole, dispatches commands to the AGVs and continually polls their status. This results in reliable and predicable solutions. The central point of control also enables easy diagnosis of errors. Driven by a shift in user requirements towards increased flexibility and openness of the system, Egemin and DistriNet Labs set up an R&D project to develop a radically new architecture based on multi-agent systems. Applying a multi-agent system opens perspectives to improve flexibility and openness of the system: the AGVs can adapt themselves to the current situation in their vicinity, order assignment is dynamic, the system can deal autonomously with AGVs leaving and re-entering the system. At the same time, the project

aimed to investigate the tradeoffs implied by a decentralized architecture in this industrial application. In particular, we were interested in the tradeoff between the performance of the system and its flexibility to deal with disturbances, and the impact of local decision making and the overall efficiency of the system (such as bandwidth requirements).

Although multi-agent systems have been applied in manufacturing control, to the best of our knowledge, the application reported in this paper is the first account of the use of multi-agent systems for real world AGV control. The decentralized AGV control system was designed and developed between 2004 and 2007 by a team of engineers and developers of Egemin and researchers of DistriNet Labs. The software system was implemented on a setup with real AGVs, and tested in larger, industrially used simulations. The delivered code base for the control software consists of about 100K lines of C# code. This system interfaces with a lower-level AGV steering system that for its real time properties is written in C. The effort demonstrates the industrial applicability and benefits of multi-agent systems for a medium size real world application.

In this paper, we report our experiences with developing the agent-based control system for AGVs. (Weyns et al. 2005) provides an initial report of the project. For in dept discussions of technical aspects of the system we refer to (Weyns and Holvoet 2008) that discusses architectural design, (Boucké et al. 2006) discusses the evaluation of the software architecture, (Schelfthout et al. 2006) elaborates on the middleware of the system, and (Weyns et al. 2008) discusses two approaches for dynamic task assignment that were developed in the context of the project.

Overview. This paper is structured as follows. Section 2 gives a brief overview of an AGV transportation system and explains the main system requirements. Before we zoom in the agent-based approach for decentralized control of AGVs, we first explain our perspective on software engineering and multi-agent systems in section 3. In section 4, we give an overview of the software architecture of the AGV transportation system and we illustrate a number of specific functionalities. We conclude the paper with a reflection on our experience and report lessons learned.

2. AGV Transportation Systems

AGVs are fully automated, custom made vehicles that are able to transport goods in a logistic or production environment, see Fig. 1. AGV transportation systems can be used for distributing manufactured products to storage locations or as an inter-process system between various production machines. 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, typically laser reflectors on walls or magnet strips in the floor. To enable the AGV control software to communicate with software systems on other machines, the vehicles are equipped with infrastructure for wireless communication.

Figure 1. One of the AGVs used in the project

We now give an overview of the functionalities of the system and the main quality requirements.

2.1 Main Functionalities

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

1. Transport assignment: transports are generated by client systems and have to be assigned to AGVs that can execute them. The stream of transports that enter the system is typically irregular and unpredictable.
2. Routing: AGVs must route efficiently through the layout of the warehouse when executing their transports.
3. Gathering traffic information: although the layout of the system is static, the best route for the AGVs in general is dynamic, and depends on the actual traffic conditions and forecasts in the system. Taking into account traffic dynamics enables the system to route AGVs efficiently through the warehouse.
4. Collision avoidance: obviously, AGVs must not collide. AGVs can not cross the same intersection at the same moment, however, safety measures are also necessary when AGVs pass each other on closely located paths.
5. Deadlock 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 are equipped with a battery as energy source. AGVs have to charge their battery at the available charging stations. Depending on the applica-

tion characteristics, a vehicle recharges when its available energy goes down a certain level, or the vehicle follows a pre-defined battery charge plan, or the vehicle can perform opportunity charging, i.e. the vehicle charges when it has no work to do. Finally, when an AGV is idle it can park at a free park location.

2.2 Quality Requirements

Various stakeholders have an interest in an AGV transportation system, such as customers, project managers, architects, simulation engineers, developers, deployment engineers, system testers, and service engineers. Evidently, these stakeholders have different, sometimes conflicting quality requirements. *Performance* is a major quality 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 systems that are able to adapt their behavior with changing circumstances autonomously. Dealing with system dynamics autonomously translates to two specific quality goals: flexibility and openness.

2.2.1 Flexibility

Flexibiltiy refers to the system's ability to exploit opportunities and anticipate possible difficulties. In the traditional centralized approach, the assignment of transports, the routing of AGVs and the control of traffic are planned by the central server. The current planning algorithm applied by Egemin is based on predefined schedules. Schedules are rules associated with AGVs and particular locations in the layout, e.g. "if an AGV has dropped a load on location x, then that AGV has to move to node y to wait for a transport assignment". A plan can be changed, however only under exceptional conditions. E.g., when an AGV becomes defective on the way to a load, the transport can be re-assigned to another AGV. 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. Flexibility also enables AGVs to anticipate possible difficulties. For example, when the amount of traffic is high in a certain area, AGVs should avoid that area; or when the energy level of an AGV decreases, the AGV should anticipate this and prefer a zone near to a charge station. Another desired property is that AGVs should be able to cope with particular situations, e.g., when a truck with loads arrives at the factory, the system should be able to reorganize itself autonomously according to the changing situation.

2.2.2 Openness

Openness of an AGV transportation system refers to the system's ability to deal autonomously with AGVs leaving and (re-)entering the system. Examples are an AGV that temporarily leaves the system for maintenance, and an AGV that re-enters the system 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. Openness is also important in the start up phase when the system is gradually deployed and tested.

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

3. Multi-Agent Systems and Software Engineering

Before we elaborate on how we have applied a multi-agent system for decentralized control of AGVs, we first explain our perspective on software engineering and multi-agent systems.

Multi-agent system research is an active research field that spawned from distributed AI in the 1980s. A multi-agent system consists of a collection of autonomous agents that interact with each other and their environment (Sycara 1998). Durfee and Lesser define a multi-agent system as "a loosely coupled network of problem solvers (agents) that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver" (Durfee and Lesser 1989). Characteristics of multi-agent systems are: (1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (2) there is no system global control; (3) data are distributed; and (4) computation is asynchronous.

Since the mid-1990s the idea that multi-agent systems are a radically new way of engineering software has dominated research in agent-oriented software engineering. Wooldridge, Jennings and Kinny state: *There is a fundamental mismatch between the concepts used by object-oriented developers and other mainstream software engineering paradigms, and the agent-oriented view. [...] Existing software development techniques are unsuitable to realize the potential of agents as a software engineering paradigm* (Wooldridge et al. 2000). This vision has led to the development of numerous multi-agent system methodologies (Henderson-Sellers and Giorgini 2005). Studying literature, however, reveals that very limited results have been obtained in the application of these methodologies to real world problems. A notable exception is the DACS methodology (Designing Agent-based Control Systems) introduced by Bussmann et al. that was applied in the design of a multi-agent system for manufacturing control at DaimlerChrysler (Bussmann et al. 2004).

Our perspective on engineering multi-agent systems starts from the viewpoint that multi-agent system engineer-

ing fits well within mainstream software engineering. This vision is based on the observation that multi-agent systems are essentially a way to structure a software system, making software architecture (Bass et al. 2003) of paramount importance in engineering multi-agent systems. Software architecture is a corner stone for ensuring that systems achieve their quality and functional goals and ultimately serve an organization's business and mission goals. It provides the required level of abstraction and generality to deal with the increasing challenges of adaptation required in distributed software applications (Kramer and Magee 2007).

Multi-agent systems are characterized by specific intra-agent and inter-agent structures yielding new architectural patterns with particular quality attributes and tradeoffs. At the level of individual agents, many different architectures have been developed, ranging from simple reactive agents to complex reasoning agents. At the system level, the multi-agent system can be structured as an organization of selfish agents that play different roles in the organization pursuing their own interests. Other multi-agent systems consist of co-operative agents that aim to achieve a common goal. Agents can interact in different ways: via a high-level communication language and specific interaction protocols, or via manipulating marks in a shared agent environment. Since specific multi-agent system structures imbues the software systems with certain qualities, while making certain trade-offs, a primary focus of multi-agent system engineering is on the software architecture of the system. Multi-agent systems are known for quality attributes such as adaptability, openness, robustness, and scalability, making multi-agent systems particularly interesting to deal with the demanding challenges of complex distributed software applications.

During architectural design, architects apply proven architectural approaches to transfer the system requirements into appropriate software structures. For the architectural design of the AGV transportation system, the architects have used a set of architectural patterns for multi-agent systems (Weyns 2006). These patterns have proven their value for systems with similar characteristics and requirements as the AGV transportation system.

4. A Multi-Agent System for Decentralized AGV Control

In this section, we give an overview of the software architecture of the AGV transportation system. We motivate why we have used a multi-agent system for decentralized control of AGVs, and we illustrate a number of aspects of the system.

4.1 Overview of the AGV Transportation System

Fig. 2 shows a general overview of the software of the AGV transportation system. The software consists of three layers. Each layer represents a virtual machine with a public interface that provides a cohesive set of services that other software can utilize without knowing how those services are

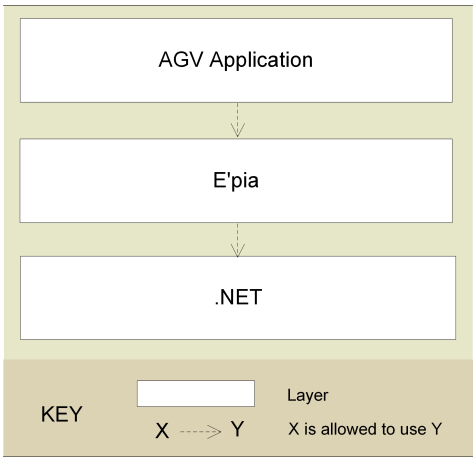

Figure 2. Software layers of the AGV transportation system

implemented. The layers are allowed to interact with each other according to a strict ordering relation. In particular, a layer A (such as AGV Application layer in Fig. 2) is allowed to use[1] any of the public facilities of the virtual machine provided by the *nearest lower layer* B (the E'pia layer in Fig. 2). Layers contribute to the modifiability and portability of a software system.

The AGV application layer is the application-specific software that accepts transport requests and instructs AGVs to handle the transports. In the traditional systems deployed by Egemin, the AGV application software consists of a central server that instructs AGVs to perform the transport requests. In the decentralized architecture, the AGV application software is structured as a multi-agent system that handles the transport requests of the clients.

The AGV application layer makes use of E'pia[2]. E'pia is a component framework developed by Egemin that provides common middleware services for logistic systems. E'pia provides general support for system configuration, communication, persistency, security, logging, visualization and diagnosis. E'pia also handles the interfacing with the low-level control software of AGVs called E'nsor[3]. We fully reused this control software in the project. E'pia translates high-level actuator commands to a low-level digital format of the actuator control software of E'nsor, and in the opposite direction, it parses the digital information derived from the sensors to provide a high-level representation of the actual status of the AGV. Internally, E'nsor is equipped with a map of the factory floor that divides the physical layout of the warehouse into logical segments of a number of meters. E'nsor is able to steer the AGV per segment. To control the AGV, E'pia provides basic actions such as `Move(segment)`

[1] The uses relation is defined by Parnas as: a unit of software A is said to use unit B if A's correctness depends upon a correct implementation of B being present (Parnas 1979).

[2] E'pia® is an acronym for Egemin Platform for Integrated Automation.

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

that instructs E'nsor to drive the AGV over the given segment, `Pick(segment)` and `Drop(segment)` that instructs E'nsor to drive the AGV over the given segment and to pick/drop the load at the end of it.

The E'pia layer make use of the Microsoft .NET framework (Richter 2002). The .NET framework provides a large body of pre-coded solutions to common program requirements, including support for user interfacing, database connectivity, network communication, and threading. .NET includes the Common Language Runtime environment (CLR) that serves as an application virtual machine shielding programmers from underlying platform details. The CLR also provides services such as security mechanisms, memory management, and exception handling.

4.2 A Multi-Agent System for the AGV Transportation System

We now zoom in on the AGV Application layer in Fig. 2 and explain how this layer is set up as a multi-agent system. The primary building blocks of the multi-agent system are agents and a virtual environment. First we introduce the two types of agents that are used in the AGV transportation system. Then, we explain the structure of the virtual environment and we show how agents use local instances of the virtual environment to coordinate their behavior. We briefly introduce ObjectPlaces, a middleware for mobile applications, that we have developed to support mobile application development. Fig. 3 shows the main building blocks of the AGV transportation system.

4.2.1 AGV Agents and Transport Agents

We have defined two types of agents: AGV agent and transport agent. 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 defined transport agents. Both types of agents share a common architectural structure, yet, they have different internal structures that provide the agents with different capabilities.

Transport Agent. 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. Transport agents reside at the Transport Base, i.e. a dedicated computer located in the warehouse, see Fig. 3.

AGV Agent. Each AGV in the system is controlled by an AGV agent. The AGV 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. AGV agents are deployed on their associated AGVs.

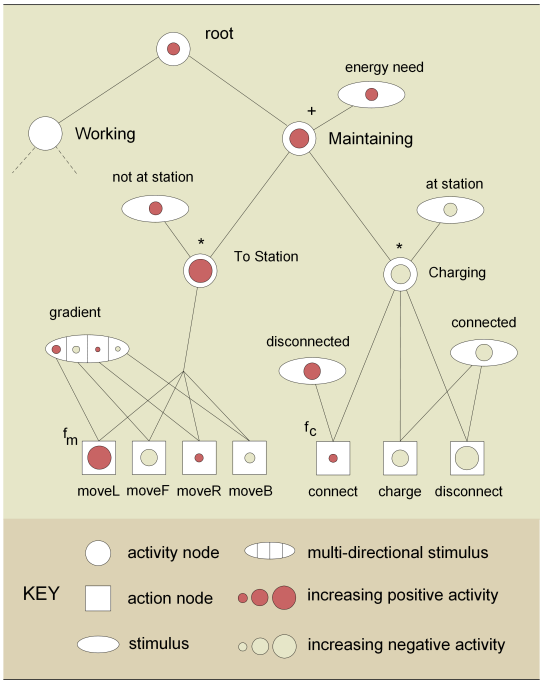

Figure 4. Example of a free-flow tree

To select actions, an AGV agent employs a free-flow tree (Tyrrell 1993). To explain action selection with a free-flow tree, we use the sample tree shown in Fig. 4. The tree is composed of a set of linked nodes. The left part of the tree represents the functionality for the agent to search a charge station and charge its battery. To select an action, activity is injected via the *root* node of the tree. The root node feeds its activity to the Working node and the Maintaining node. The Maintaining node combines the received activity with the activity from the *energy need* stimulus. The "+" symbol indicates that the activities are summed up. The Maintaining node feeds the combined activity down through the hierarchy until the activity arrives at the action nodes, i.e. the leaf nodes of the tree. While the activity flows through the tree, some of the nodes receive additional activity from stimuli such as *at station*, *connected*, and *gradient*. This latter is a multi-directional stimulus that provides a value of the stimulus for four moving direction. These values are based on the sensed values of a computational field that is transmitted by the charge station (we elaborate on the use of fields in section 4.2.2). In a similar way, the Working node feeds its activity through the hierarchy. Action nodes that receive activity from different nodes combine that activity according to a specific function (f_m and f_c in the sample tree of Fig. 4) to calculate the final activity levels. When all action nodes have collected their activity the node with the highest activity level is selected for execution. In the example, the activity collected by the *moveL* node (move left) is clearly dominant so the agent would select this action.

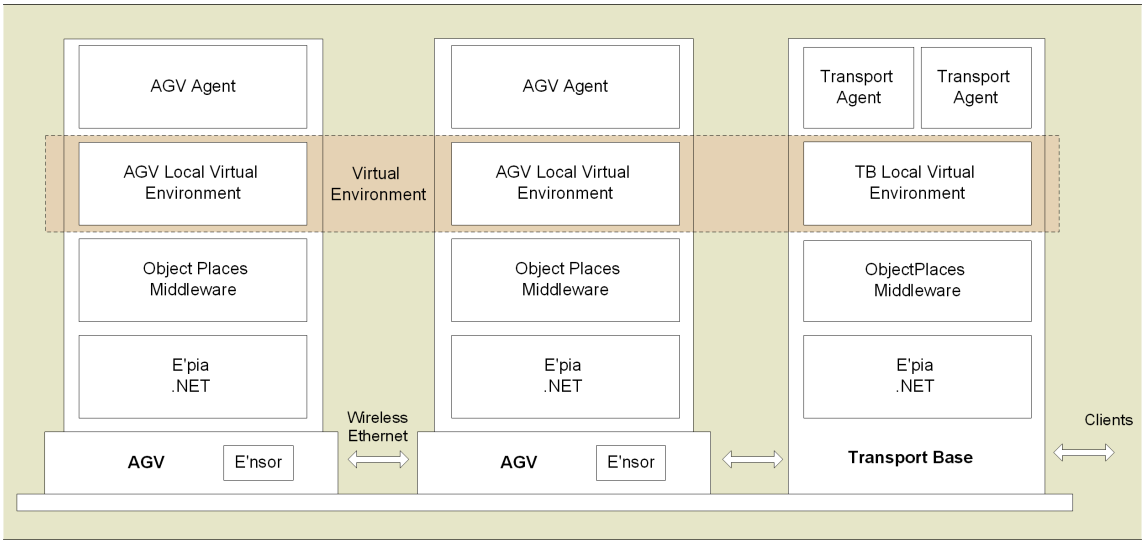

Figure 3. High-level model of an AGV transportation system

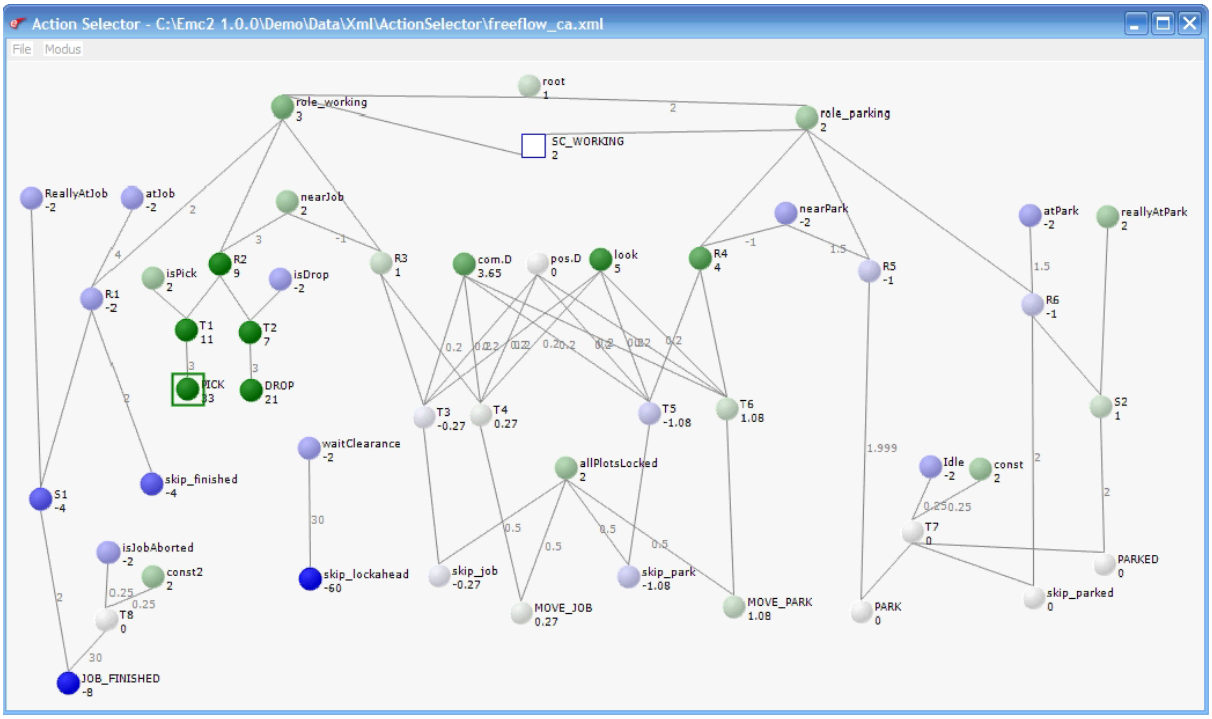

Figure 5. Runtime snapshot of the decision tree of an AGV agent

Fig. 5 shows a runtime snapshot of the decision making tree of the AGV agent.

To enable agents to set up collaborations, we extended free-flow trees with the notions of role and commitment to allow agents to set up collaborations (Steegmans et al. 2006). Roles correspond to particular subtrees; see e.g. the *role_working* and *role_parking* in Fig. 5. Commitments are represented by directed connections between top nodes of roles, see the *SC_WORKING* commitment in Fig. 5. When a commitment is activated, additional activity is injected in the role of the commitment. As a result, the behavior of the agent will give preference to actions in this role. In the snapshot, the AGV is manoeuvering towards a pick location and the *PICK* action is selected.

In the initial phase of the project, the complete decision making mechanism of the AGV agents was based on a free-flow tree. 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 at a high-level of abstraction; this preserves the advantage of adaptive action selection with a free-flow tree. At the following levels, the selected action is further refined taking into account collision avoidance and deadlock avoidance. Each decision-making component in the chain is able to send feedback to the action controller to revise the decision. This feedback loop helps to improve flexible decision making.

Agents contribute to the quality attributes flexibility and openness. Particular motivations are: (1) agents act locally; this enables agents to exploit opportunities and adjust their behavior with changing circumstances in the system and its environment—this is an important property for flexibility; (2) agents are autonomous entities that interact with one another in their vicinity; agents can enter and exit each others area of interaction at any time—this is an important property for openness.

4.2.2 Virtual Environment

To achieve the system requirements, AGV agents and transport agents have to coordinate. Agents have to coordinate for routing, for transport assignment, for collision avoidance, etc. A typical approach would provide an infrastructure for communication that enables the agents to exchange messages to coordinate their behavior. This however, would put the full complexity of coordination in the agents resulting in complex architectures of the agents, in particular for the AGV agents. We have chosen for a solution that enables the agents to exploit a *virtual environment* to exchange information and to coordinate their behavior, see Fig. 3. Besides, the virtual environment serves as a suitable abstraction that shields the agents from low-level issues, such as the transmission of messages and the low-level interfacing with sensors and actuators of AGVs. This approach separates responsibilities in the system and helps to manage the complexity.

Since AGV agents and transport agents are deployed on different nodes, the virtual environment is necessarily distributed over the AGVs and the transport base. AGVs and the transport base maintain a local virtual environment, which is a local manifestation of the virtual environment. The states of the local virtual environments are synchronized opportunistically, as the need arises. We explain state synchronization of local environments below when we elaborate on the ObjectPlaces middleware. The instances of the local virtual environment deployed on the nodes in the system are tailored to the type of agents deployed on the nodes. For example, the local virtual environment on the AGVs provides a high-level interface that 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.

Coordination through the local virtual environment. 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. We illustrate how agents exploit the local virtual environment to assign tasks and to avoid collisions.

Transport assignment. We have developed two approaches for adaptive task assignment and used it the AGV transportation system. FiTA (field-based task assignment) is an approach in which agents emit fields in the local virtual environment that guide idle AGV agents to loads. DynCNET is a protocol-based approach that extends standard contract net (CNET (Smith 1980)). In DynCNET, the agents use explicit negotiation to assign tasks. Here we illustrate FiTA in which agents coordinate through the local virtual environment.

The basic idea of FiTA is to let each idle agent follow the gradient of a field that guides it toward a task that has to be executed. In FiTA, two types of fields are used: transport fields which are emitted by transports and AGV fields are emitted by AGVs. Transport fields attract idle AGVs. However, to avoid multiple AGVs driving toward the same transport, AGVs emit repulsive fields. AGV agents combine perceived 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. The fields of the AGV agents have a fixed range and contain the identity of the AGV and its current location. The range of transport fields is variable and depends on the priority of the tasks. The spreading of the fields is a responsibility of the local virtual environments. With FiTA, the agents continuously reconsider the situation and task assignment is delayed until the execution of the task starts which benefits the flexibility of the system. When a task or AGV enters or leaves the system the perceived fields of local agents will be adapted supporting openness of the system. Both AGV agents and transport agents emit fields in the local virtual environment. Fig. 6 shows a snapshot of a simulation of an implemented system where a number of agents emit fields.

Fig. 7 compares the performance of FiTA with a static approach for transport assignment in an industrial test case with 14 AGVs (Weyns et al. 2008). At the beginning of the test, 45 transports are simultaneously created at a particular number of locations in the warehouse. These transports have to be dropped at a specific set of destinations. The test represents for example the arrival of a truck with loads that have to be unloaded and stored in a warehouse as quickly as possible. The slope of the curve of FiTA is much steeper than the curve of the static approach for transport assignment. The results demonstrate that static transport assignment requires about 2.5 times more time to complete the 45 transports than FiTA.

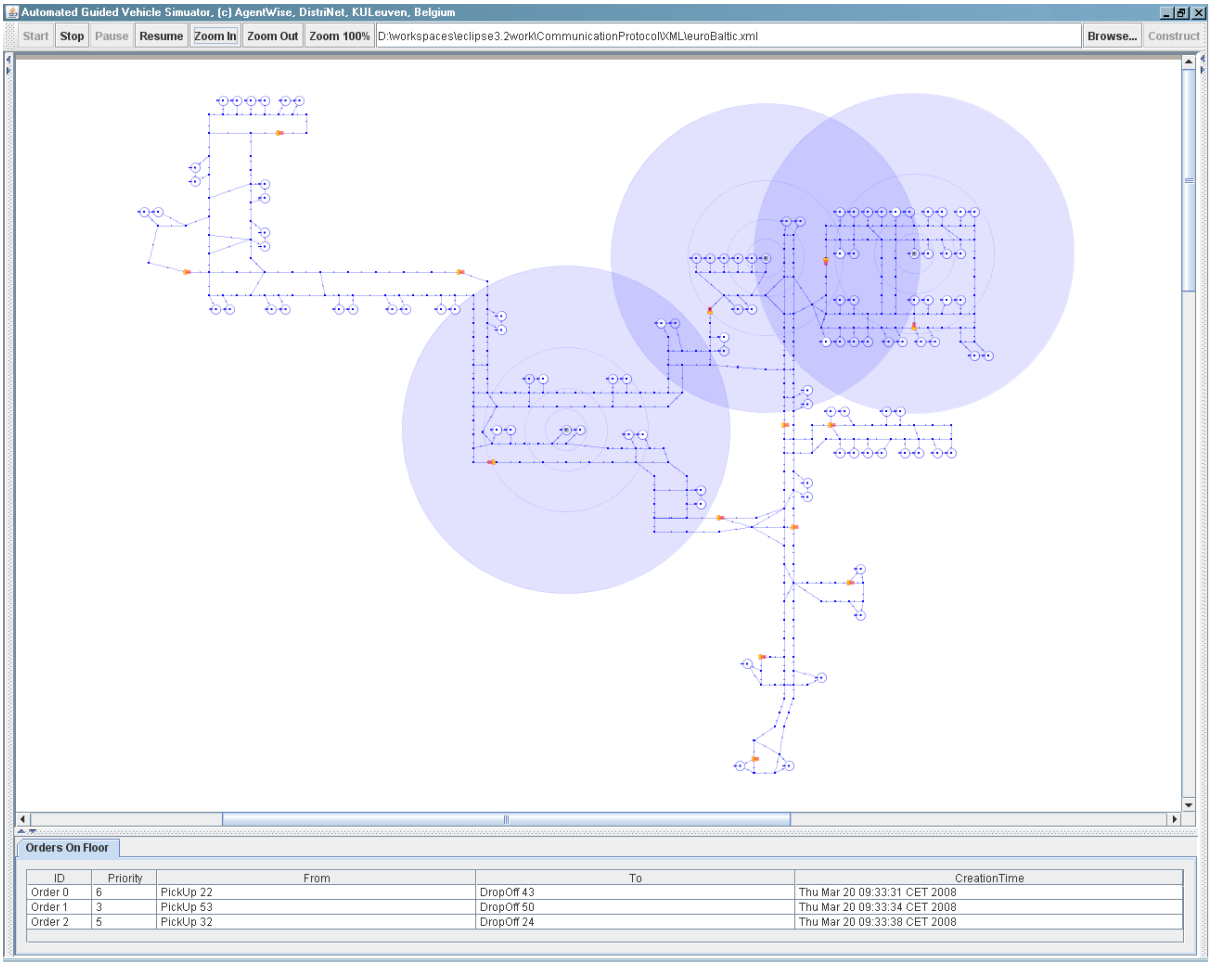

Figure 6. Transports emit fields in the virtual environment to attract idle AGVs

Collision avoidance. AGV agents mark the path they are going to drive in their local virtual environment using *hulls*. The hull of an AGV is the physical area the AGV occupies. A series of hulls describe the physical area an AGV occupies along a certain path. If the area is not marked by other hulls (the AGV's own hulls do not intersect with others), the AGV can move along and actually drive over the reserved path. In case of a conflict, the involved local virtual environments use the priorities of the transported loads and the vehicles 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 local virtual environment.

The left parts of Fig. 8 and Fig. 9 show two AGVs coordinate to avoid a collision (we explain the right part of the figures below). The snapshots show a fusion view of the two local virtual environments that is collected via remote access of the two AGVs.

In Fig. 8 two AGVs are coordinating. Both AGVs are projecting hulls in the virtual environment. At this point there is an overlap of hulls: the AGV on top (box marked with X) is ready to move to the left, the smaller AGV at the bottom is manoeuvring to turn right. The AGV at the bottom has already reserved the trajectory occupied by its hull, and thus it has priority to the AGV at the top that must wait. In Fig. 9, the AGV at the bottom has passed the curve. There is no longer an overlap of hulls and the AGV at the top has started driving to the left.

4.2.3 ObjectPlaces: Middleware for Mobile Applications

The mobility of the AGVs impose highly dynamic operation conditions and inherent distribution of resources. A typical approach in mainstream software engineering is to support distribution with a suitable middleware. We have developed a middleware for mobile applications called ObjectPlaces (Schelfthout 2006). Mobile applications such as an AGV transportation system are characterized by (1) their need to take into account their physical environment (usually called context) explicitly; and (2) their need to deal with dynamics and unexpected events originating from their con-

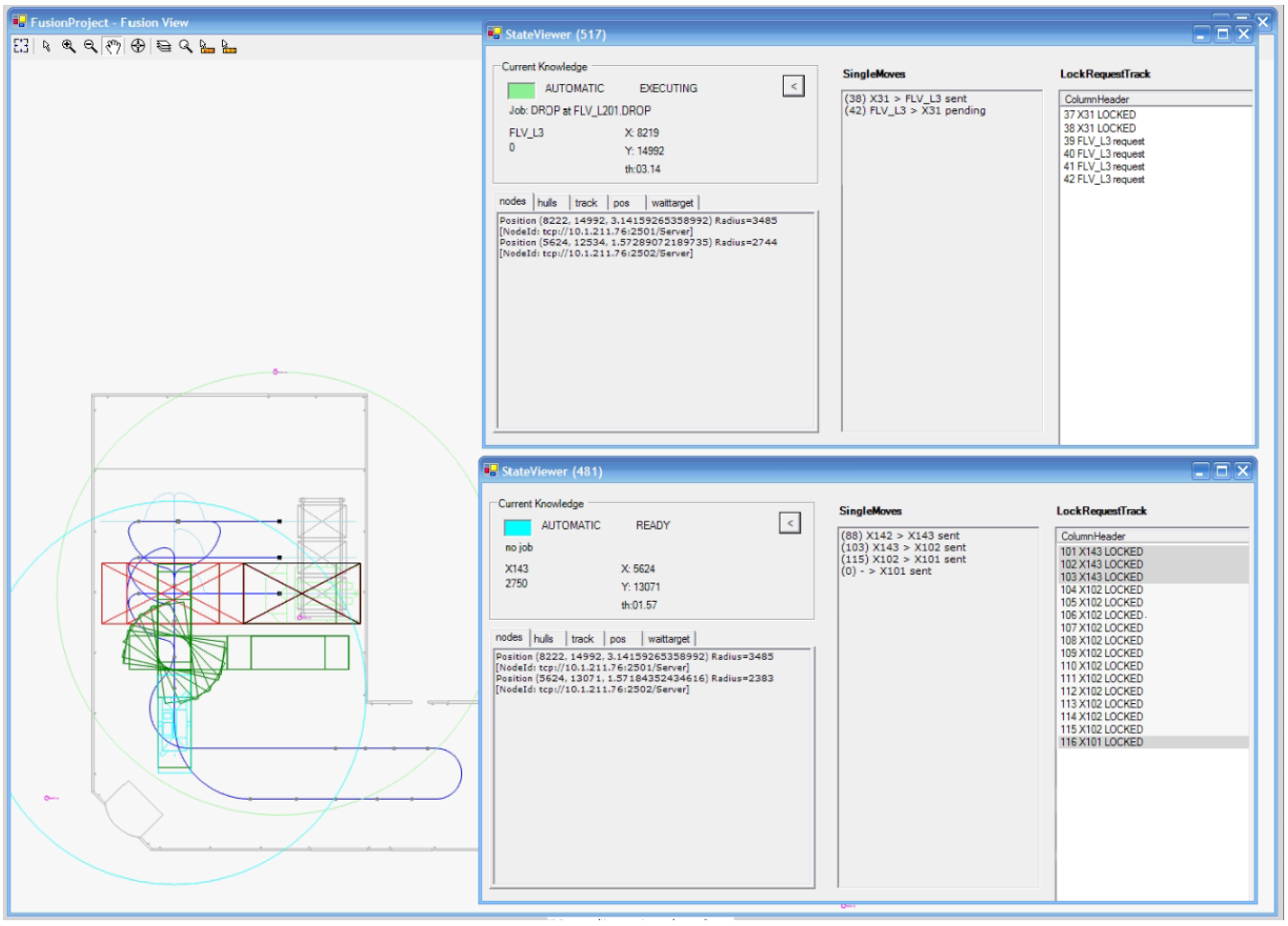

Figure 8. Hulls are overlapping; the AGV on top has to wait

text. ObjectPlaces proposes two programming abstractions, views and roles, to support mobile application development with respect to those two needs.

The first abstraction, a view, is an automatically up-to-date collection of data objects that are copies or representations of data objects available on a set of nodes in the network. The middleware automates gathering the data objects from a set of nodes and maintains the view in the face of dynamically changing availability of the data objects.

The second abstraction, a role, encapsulates the behavior of a component of the application engaging in a protocol. The middleware automates the setup and maintenance of an interaction session between a number of participating components in the mobile network in the face of a continuously changing number of participants.

ObjectPlaces has significantly simplified the development of the application components of the AGV transportation system. The middleware encapsulates the tedious management tasks associated with distribution in mobile systems. This significantly reduced the complexity of tackling distributed coordination problems such as collision avoidance, deadlock detection, and transport assignment in the AGV transportation system.

We illustrate the use of the middleware abstractions for collision avoidance. When an AGV moves on, a view is maintained that keeps track of AGVs within a certain distance, i.e. the AGVs in collision range. When an AGV approaches and enters the collision range, the middleware will include that AGV in the view. Similarly, when an AGV leaves the collision range, that AGV will be removed from the view. As such, the application components have an up to date view of the AGVs in collision range that they can use to coordinate the vehicles avoiding collisions. In order to avoid collisions AGVs need to execute a mutual exclusion protocol with the group of all vehicles in collision range. The protocol will determine which AGV can move first. It is important to note that this group is dynamic, since AGVs may enter and leave the collision range continuously. The middleware automates the process of discovering the group of AGVs that are in collision range, and maintaining the group of interacting vehicles as they arrive and leave. The state viewer in Fig. 8 on top (right hand side, section LockRequestTrack) shows

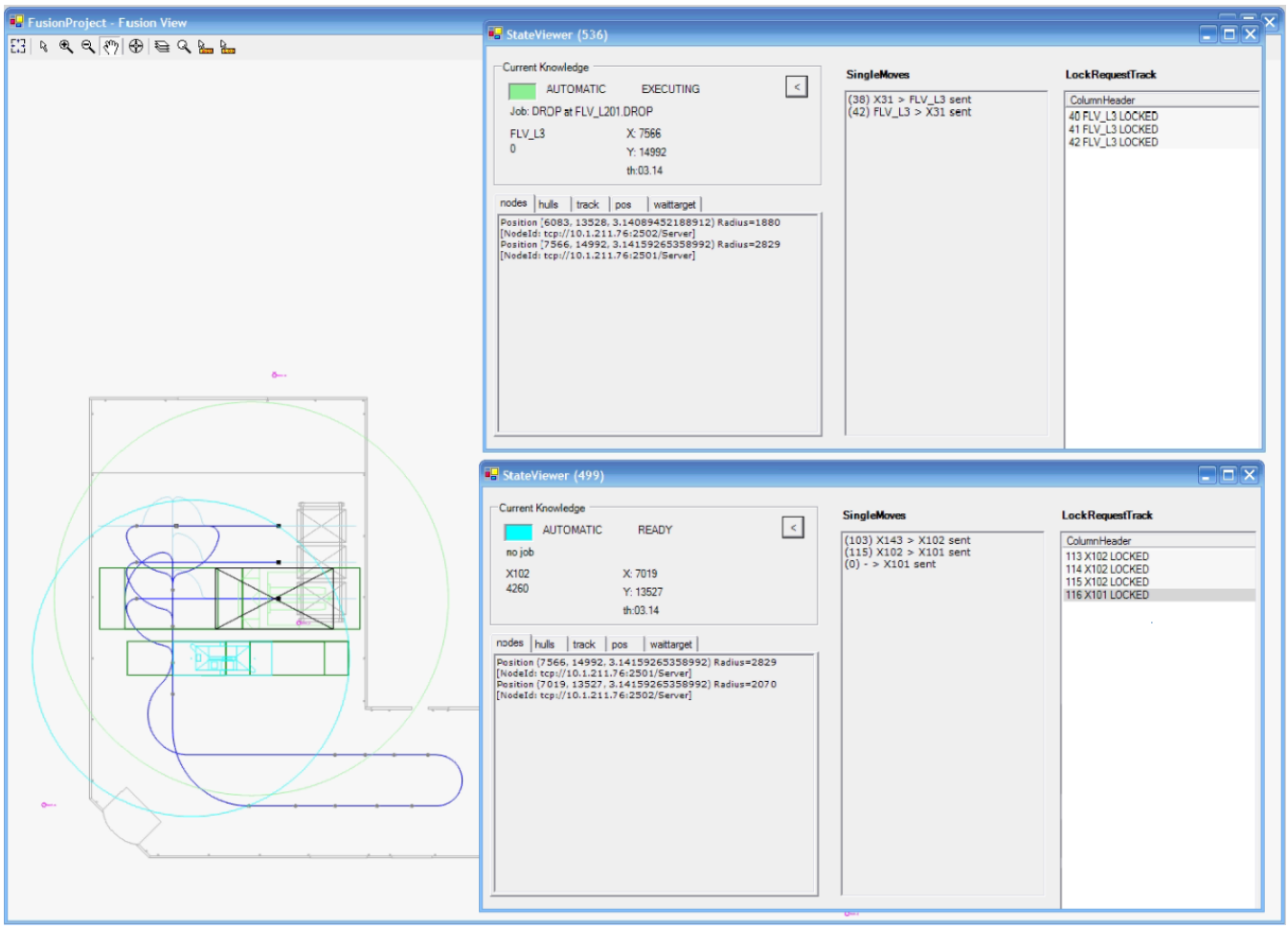

Figure 9. The AGV on top can start driving now

the state of the segments reserved by the big AGV (i.e. the AGV on top in Fig. 8 on the left hand side). Currently, two of the reserved segments are in the state *LOCKED*, four other segments are in the state *request*. On the other hand, the state viewer at the bottom shows that all the segments reserved by the small AGV (i.e. the AGV at the bottom in Fig. 8 on the left hand side) are in the state *LOCKED*. As a result the small AGV at the bottom is allowed to move on while the big AGV at the top must wait. The state viewers in Fig. 9 show that all the reserved segments of both AGVs are *LOCKED* and thus the AGVs can safely move on.

5. Reflection and Lessons Learned

We conclude the paper with a reflection on our experiences and lessons learned.

Dealing with stakeholders' requirements. The general motivation to apply a multi-agent system in the AGV control system were new and future quality requirements, in particular flexibility (deal autonomously with dynamic operating conditions) and openness (deal autonomously with AGVs entering and leaving the system). However, as we have explained, for a complex system such as the AGV control system the stakeholders have various, often conflicting requirements. During a four day Quality Attribute Workshop (Barbacci et al. 2003) with the key stakeholders of the system, we identified and prioritized important quality attributes in terms of concrete scenarios. The highest ranked quality scenarios were the main drivers for architectural design. The workshop enabled us (1) to precisely specify the qualities addressed by adopting a multi-agent system, and (2) to determine their importance relative to other qualities. This was important for preventing the industrial partner from overestimating or underestimating agent technology.

Managing complexity. AGV control systems are very complex software systems. The design and implementation of the agent-based AGV transportation system needed 8+ man-years of effort. The delivered code base consists of about 100K lines of C# code. Such complexity can only be managed through abstraction. Software architecture is centered on the idea of reducing complexity through abstraction and separation of concerns. In the AGV control system, software architecture allowed us to manage the complexity

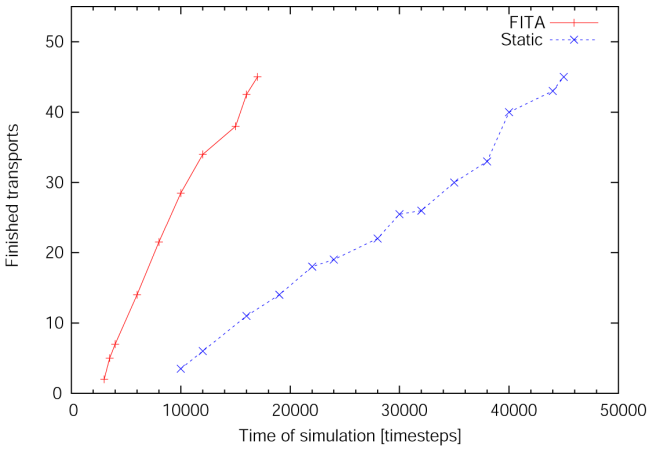

Figure 7. Number of finished transports in a stress test. With *Static*, each transport that enters the system is immediately assigned to the best AGV at that moment. With *FiTA*—the new approach for transport assignment—the AGV agents follow fields in the local virtual environment that guide them to the loads. The agents continuously reconsider the situation and transport assignment is delayed until the execution of a transportation task starts.

of the multi-agent system at different levels of abstraction (intra-agent and inter-agent structures, behavior, and hardware/software allocation).

Integrating a multi-agent system with its software environment. In an industrial setting, systems are not built in isolation. When introducing a multi-agent system, it must be integrated with its environment (common frameworks, legacy systems, etc.). In Egemin, .NET is the standard environment and the company uses the in-house developed E'pia framework that provides common middleware services. Examples of legacy systems with which the multi-agent system needed to be integrated are the enterprise resource planning system that generates the transport tasks and the low-level control software of the AGVs. Software architecture was the key to accommodate the integration of the multi-agent system with its environment. We integrated E'pia as a basic layer that provides the required services to deal with various crucial requirements. With respect to legacy systems, we were able to develop proper mediator components/agents to integrate legacy systems with the multi-agent system.

Architectural design and evaluation. Preceding experiences with developing multi-agent system applications with characteristics and requirements similar as the AGV control system yielded a set of architectural patterns for multi-agent systems and a supporting middleware for mobile applications. Initially, we faced the problem of how we could exploit these reusable assets and integrate them in the design of the AGV control system. The solution was the Attribute-Driven Design method (ADD (Buchmann and Bass 2001)). ADD is a well-established method for architectural design that is based on understanding how to achieve quality goals through proven architectural approaches. During the architectural design we employed the patterns for multi-agent systems together with a number of common architectural patterns to decompose and structure the system and realize the required functionalities and qualities. To pinpoint the qualities and tradeoffs implied by the decentralized multi-agent system architecture, a disciplined evaluation of the software architecture was necessary. Therefore, we organized a one day ATAM (Architectural Tradeoff Analysis Method (Clements et al. 2002)). During the ATAM an external evaluation team and the main stakeholders determined the trade-offs and risks with respect to satisfying important quality attribute scenarios, particularly scenarios related to flexibility, openness, performance, and robustness. One important outcome of the ATAM was an improved insight on the tradeoff between flexibility and communication load.

Agents and objects. By considering multi-agents systems from a software architecture perspective, agents and the virtual environment provide a specific architectural style that imposes particular constraints on the system while yielding particular qualities and tradeoffs. The agent-based software architecture served as a blueprint for system development. The software architecture defines constraints on detailed design and implementation, it describes how the implementation must be divided into elements and how these elements must interact with one another to fulfill the system goals. On the other hand, the software architecture did not *define* the implementation, many fine-grained design decisions were left open by the architects and must be resolved by designers and developers. Examples are internal data structures of modules, specific algorithms, the use of specific object-oriented design patterns, detailed exception handling, etc. Whereas the multi-agent system determined the course-grained structure of the software, the final implementation of the system was realized in C#. This project demonstrates that agents and objects can be complementary paradigms.

Impact of a multi-agent system on the company's organization. From our experience, a crucial issue with respect to industrial adoption of multi-agent systems is the impact of introducing a multi-agent system on the organization of the developing company. At Egemin, the existing AGV control system has a centralized server-oriented architecture. The agent-based approach on the other hand has a decentralized architecture. Switching from a centralized toward a decentralized agent-based architecture is a big step with far reaching effects for a company, not only for the software but for the whole organization. To give one example: in the centralized architecture transport assignment to AVGs is based on application-specific rules that are associated with particular locations in the environment. A team of specialized layout engineers is responsible for defining these rules. In the decentralized architecture, however, one approach for transport assignment is

using a dynamic protocol between AGV agents and transport agents (Weyns et al. 2008). This protocol must be tuned per project, but this requires completely different skills. Our experience indicates that the introduction of an agent-based approach should be done in a controlled way, step-by-step. Software architecture is the indispensable vehicle for stepwise integration of a multi-agent system. It provides the required level of abstraction to reason about and deal with gradual integration of multi-agent systems.

6. Conclusion

In this paper, we reported our experiences with applying a decentralized architecture for AGV control. We gave an overview of the agent-based architecture of the system and reflected on lessons learned. By linking multi-agent systems to software architecture, we were able to convince the industrial partner of the benefits of multi-agent system in the AGV control system.

Self-adaptability, scalability, and local autonomy are generally considered as key properties to tackle the growing complexity of software. These are exactly properties that characterize multi-agent systems. The project we reported in this paper demonstrates that by putting multi-agent systems it in a broader setting of mainstream software engineering, especially software architecture, multi-agent systems can contribute to tackle the challenges of future software systems.

Acknowledgments

Danny Weyns is supported by the Funding for Scientific Research in Flanders (FWO). We are grateful to the employees of Egemin, in particular Jan Vercammen, Wim Van Betsbrugge, Rudi Vanhoutte, Walter De Feyter, and the former employee Tom Lefever. Thanks to Alexander Helleboogh, Nelis Boucké, Wannes Schols and Bart Demarsin for the collaboration in the EMC^2 project. We recognize the anonymous reviewers for the valuable feedback on earlier versions of this paper. Finally, we thank Harald Wesenberg for his support with the finalization of the paper.

References

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

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

Nelis Boucké, Danny Weyns, Kurt Schelfthout, and Tom Holvoet. Applying the ATAM to an architecture for decentralized control of a transportation system. In *Second International Conference on Quality of Software Architectures*, volume 4214 of *Lecture Notes in Computer Science*, Springer, 2006.

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

S. Bussmann, N. Jennings, and M. Wooldridge. *Multiagent Systems for Manufactoring Control: A Design Methodology*. Springer Series on Agent Trchnology, 2004.

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

E. Durfee and V. Lesser. Negotiating Task Decomposition and Allocation Using Partial Global Planning. *Distributed Artificial Intelligence*, 2:229–244, 1989.

B. Henderson-Sellers and P. Giorgini. *Agent-oriented Methodologies*. Idea Group Inc., 2005.

J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *FOSE '07: 2007 Future of Software Engineering*, Washington, DC, USA, 2007. IEEE Computer Society.

D. Parnas. Designing Software for Ease of Extension and Contraction. *IEEE Transactions on Software Engineering*, 5(2):128–137, 1979.

J. Richter. *Applied Microsoft .NET Framework Programming*. Microsoft Press, Redmond, USA, 2002.

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

K. Schelfthout, D. Weyns, and T. Holvoet. Middleware that Enables Protocol-Based Coordination Applied in Automatic Guided Vehicle Control. *IEEE Distributed Systems Online*, 7(8), 2006.

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.

E. Steegmans, D. Weyns, T. Holvoet, and Y. Berbers. A Design Process for Adaptive Behavior of Situated Agents. In Agent-Oriented Software Engineering V, volume 3382 of *Lecture Notes in Computer Science*, Springer, 2005.

K. Sycara. Multiagent Systems. *Artificial Intelligence*, 10(2):79–93, 1998.

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

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

D. Weyns and T. Holvoet. Architectural design of a situated multiagent system for controlling automatic guided vehicles. *International Journal on Agent Oriented Software Engineering*, 2(1): 90–128, 2008.

D. Weyns, K. Schelfthout, T. Holvoet, and T. Lefever. Decentralized control of E'GV transportation systems. In *4th Joint Conference on Autonomous Agents and Multiagent Systems, Industry Track*, Utrecht, The Netherlands, 2005. ACM, New York, USA.

D. Weyns, N. Boucke, and T. Holvoet. A field-based versus a protocol-based approach for adaptive task assignment. *Autonomous Agents and Multi-Agent Systems*, 2008. in press.

M. Wooldridge, N. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.