A Field-Based Versus a Protocol-Based Approach for Adaptive Task Assignment

Danny Weyns (danny.weyns@cs.kuleuven.be)
Katholieke Universiteit Leuven, Belgium

Nelis Boucké (nelis.boucke@cs.kuleuven.be)
Katholieke Universiteit Leuven, Belgium

Tom Holvoet (tom.holvoet@cs.kuleuven.be)
Katholieke Universiteit Leuven, Belgium

Abstract.

Task assignment in multi-agent systems is a complex coordination problem, in particular in systems that are subject to dynamic and changing operating conditions. To enable agents to deal with dynamism and change, adaptive task assignment approaches are needed. In this paper, we study two approaches for adaptive task assignment that are characteristic for two classical families of task assignment approaches. FiTA is a field-based approach in which tasks emit fields in the environment that guide idle agents to tasks. DynCNET is a protocol-based approach that extends standard contract net (CNET). In DynCNET, agents use explicit negotiation to assign tasks.

We compare both approaches in a simulation of an industrial automated transportation system. Our experiences show that: (1) the performance of DynCNET and FiTA are similar, while both outperform CNET; (2) the complexity to engineer DynCNET is similar to FiTA but much more complex than CNET; (3) whereas task assignment with FiTA is an emergent solution, DynCNET specifies the interaction among agents explicitly allowing engineers to reason on the assignment of tasks, (4) FiTA is inherently robust to message loss while DynCNET requires substantial additional support. The tradeoff between (3) and (4) is an important criteria for the selection of an adaptive task assignment approach in practice.

1. Introduction

The advances in computing and communication technology pave the way to large-scale distributed software systems such as automated transportation systems and online manufacturing control. At the same time, these systems introduce increasing levels of complexity to software engineers. Two important factors for the growing complexity are: the highly dynamic operating conditions under which systems have to operate, such as altering workloads and variations in availability of resources, and the inherent distribution of resources which makes central control practically infeasible.

In our research, we study situated multi-agent systems (situated MAS) for engineering such systems. A situated MAS structures the software as a number of interacting autonomous entities (agents) that have an explicit position in an environment. Situated agents use computationally efficient action

selection mechanisms (Tyrrell, 1993, Maes, 1994, Ferber and Muller, 1996, Bandini et al., 2002, Weyns and Holvoet, 2006) to respond rapidly to dynamic and changing circumstances. Situated MAS are collaborative systems in which agents work together locally to solve a complex overall problem. Great emphasis is put on a suitable coordination mechanism. Situated agents typically coordinate indirectly through a shared coordination medium. Examples are agents that coordinate their behavior via computational fields or digital pheromones.

Task Assignment in Situated MAS. One particularly challenging coordination problem in situated MAS is task assignment. Tasks in situated MAS are often characterized by delayed commencement, i.e. the execution of a task requires a preceding effort of an agent before the task can actually be executed. We illustrate the problems of assigning tasks with delayed commencement in dynamic systems with an example application from the domain of manufacturing control that is described in (Bussmann and Schild, 2000). In this system, workpieces move on a conveyor belt along a set of machines. Workpieces have to pass through a series of operations. Some of the machines can perform the next operation on the workpieces. The authors propose an approach where workpieces select the next machine based on a first-price single-round auction. However, several kinds of dynamics may arise while a workpiece moves toward the assigned machine. A machine may finish an operation on another workpiece, making the machine available for the workpiece that is on its way to its assigned machine. Machines can be added to the system, or can be temporarily unavailable due to a breakdown or maintenance. Task assignment with a first-price single round auction is not able to deal with such kinds of dynamics. The assignment of tasks with delayed commencement requires adaptive task assignment approaches that enable agents to deal with dynamism and change in the system and its environment.

Two Approaches for Adaptive Task Assignment. In this paper, we study two approaches for adaptive task assignment that are characteristic for two classical families of task assignment approaches. In particular, we study and compare a field-based approach for task assignment (FiTA) with a protocol-based approach (DynCNET). In FiTA, tasks emit fields in the environment that attract idle agents. Agents follow the gradient of the combined field that guide them toward tasks. DynCNET is an extension of the well-know contract net protocol (CNET (Smith, 1980)), with "Dyn" referring to support for dynamic task assignment. Both FiTA and DynCNET enable task assignment in the system based on local interaction among agents and allow for adaptation of tasks assignment during delayed commencement. Yet, the approaches differ in the manner agents realize task assignment. In FiTA, agents use simple rules that guide them toward tasks, providing an emergent solution for task

assignment. Contrary, in DynCNET agents use explicit selection mechanisms and can negotiate about task assignment.

Our focus is on systems with homogeneous tasks that can be executed by individual agents. In this paper, we do not consider complex tasks, for instance composite tasks that have to be divided among agents, or a combination of related tasks that have to be executed by a single agent. The motivation for this restriction is twofold. First, the adaptive task assignment approaches were developed and tested in the context of an R&D project in which we developed a decentralized control architecture for an automated transportation system. The basic task of the automatic guided vehicles (AGVs) in such system is to transport loads in an industrial environment. Second, it allowed us to focus on the basic challenges of task assignment in systems that are subject to dynamic and changing operation conditions.

In our study, we apply DynCNET and FiTA to a simulation of an industrial transportation system. We make a tradeoff analysis and compare: (1) the performance of both approaches (throughput and bandwidth usage), (2) a number of important quality attributes, including flexibility (adapt to dynamics that happen during task assignment), openness (take into account agents that enter/leave the system in the process of task assignment), and robustness to message loss (degrade gracefully with increasing loss of messages), and (3) the complexity and support to engineer the approaches. In the experiments, CNET is used as a reference protocol. CNET is equivalent to DynCNET without reallocation of tasks.

Overview. In the next section, we introduce the AGV transportation system. Section 3 discusses the two approaches for dynamic task assignment: FiTA and DynCNET. In section 4, we present test results obtained from applying both approaches in a simulated AGV transportation system, and we make a tradeof analysis. Section 5 discusses related work. Finally, in section 6 we draw conclusions.

2. The AGV Transportation System

FiTA and DynCNET were developed in the context of a joint R&D project between DistriNet Labs at the Katholieke Universiteit Leuven and Egemin[1], a manufacturer of AGVs and control software for automating logistics services in warehouses and manufactories. In this project, we have applied a situated MAS to develop a decentralized control architecture for an automated transportation system that uses multiple AGVs (Weyns et al., 2005). The goal of the project was to improve flexibility and openness. To deal with the dynamic operating conditions, we have tested both FiTA and DynCNET for task assignment in a simulated AGV transportation system. It is common practice

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

in AGV system development to perform extensive simulation tests before the system is deployed on site. This avoids high costs of physical test settings with many AGVs. We use the AGV transportation system in this paper as a case to explain the two approaches for adaptive task assignment, and we make a tradeoff analysis of the approaches for a simulated AGV transportation system.

In this section, we give an overview of the AGV transportation system, providing the necessary background for the remainder of the paper. First, we briefly explain the main architectural views of the real system that was tested on a setup with AGVs at the Egmin test site (Weyns and Holvoet, 2007). Then, we give a general overview of the architecture of the simulated system that we have used for the tests presented in this paper, and we explain the assumptions of the simulation.

2.1. AGVs and Transports

An AGV is a unmanned, battery powered vehicle, capable of picking up a load, driving around and dropping it. AGVs can communicate by means of a wireless LAN. The AGVs are restricted to follow a predefined layout in the warehouse environment. The layout is defined in a map that consists of a network of stations and segments that are accessible by the AGVs. AGVs can leave and re-enter the system to charge their battery or for maintenance. AGVs are equipped with low-level control software that uses sensors and actuators to stay on track, turn, pick and drop a load, and determine the current position.

A transport represents a task to move a load from a pick location to a drop location. Transports are generated by an external system, typically a warehouse management system. The stream of transports that enter the system is typically irregular and unpredictable. Each transport has a priority that depends on the importance of the task and that increases over time to avoid starvation. Transports in an AGV transportation system are characterized by delayed commencement, i.e., an AGV first has to drive to a load before it can pick the load and transport it to the destination. While driving toward the load all kinds of changes in the system may occur. New tasks may enter the system that are more suitable for the AGV to execute, new AGVs may become available that are more suitable to perform the task, etc.

2.2. Main Views of the Software Architecture

Fig. 1 shows the deployment view of the decentralized AGV transportation system that describes how the software is deployed on hardware elements. The software system consists of two subsystems: the AGV Control System and the Transport Base System.

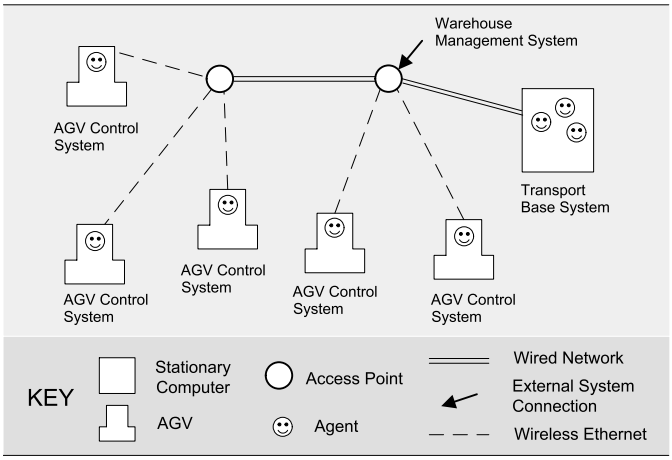

Figure 1. Deployment view of the AGV transportation system.

AGV Control System. Each AGV is equipped with a computer containing an AGV control system. The AGV control system contains a single *AGV agent* that is responsible for obtaining and executing transports, and ensuring that the AGV gets maintenance on time (such as charging the AGV's battery). As such, an AGV becomes an autonomous entity that can take advantage of opportunities that occur in its vicinity, and that can enter/leave the system without interrupting the rest of the system.

Transport Base System. Each transport in the system is represented by a *transport agent*. Transport agents are part of the transport base system that is deployed on a stationary computer system located in the warehouse. A transport agent is responsible for assigning the transport to an AGV and reporting the status and completion of the transport to the warehouse management system. Transport agents are autonomous entities that interact with AGV agents to find suitable AGVs to execute the transports.

All subsystems can communicate via a wireless network. The warehouse management system interacts with the AGV transportation system via a wired network. In the next section, we zoom in on the internal structure of the AGV control system. The transport base system has a similar architecture.

2.3. AGV CONTROL SYSTEM

The left part of Fig. 2 shows the architecture of the AGV control system. The right part zooms in on the AGV agent architecture.

AGV Agent. AGV agent is responsible for controlling the AGV and executing transports. The main functionalities of an AGV agent are: (1) obtaining transport tasks; (2) handling jobs and reporting the completion of jobs; (3)

avoiding collisions; (4) avoiding deadlock; (5) maintaining the AGV machine (charging battery, calibrating etc.); (6) parking when the AGV is idle. The AGV agent consists of three submodules: Decision Making is responsible for selecting actions, Communication handles communicative interactions, and Perception perceives the local virtual environment based on requests of Decision Making and Communication. The AGV agent's knowledge is maintained in the Current Knowledge repository.

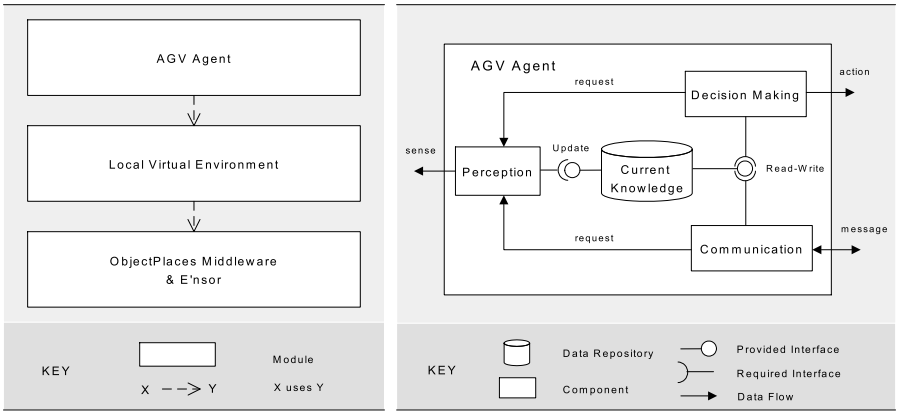

Figure 2. Architecture of the AGV control system on the right, with a detail of the AGV agent on the left

Local Virtual Environment. Since the physical environment of AGVs is very restricted, it offers little opportunities for agents to use the environment for coordination purposes. The local virtual environment is a software entity that represents and maintains relevant state of the physical environment and state that is used by the AGV agent to exchange information with other agents and to coordinate their behavior (e.g., by means of fields in FiTA). The local virtual environment 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 vehicle; (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 (Schelfthout, 2006). ObjectPlaces proposes two programming abstractions, views and roles, to support mobile application development. 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 (Schelfthout et al., 2006). A role encapsulates the behavior of a component of the application engaging in a protocol (Schelfthout and Holvoet, 2005). The middleware automates the setup and maintenance of an interaction session between a number of participating roles in the mobile network. ObjectPlaces encapsulates the tedious management tasks associated with distribution in mobile systems. This significantly reduced the complexity of tackling distributed coordination problems in the AGV transportation system, such as collision avoidance, deadlock detection, and task assignment.

E'nsor is the low-level control software of the AGV that provides an interface to command the 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. E'nsor understands basic actions such as `Move(Segment s)` that instructs E'nsor to drive the AGV over the given segment, and `Pick(Segment s)` and `Drop(Segment s)` that instructs E'nsor to drive the AGV over the given segment and to pick/drop the load at the end of it. 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.

2.4. Architecture of Simulated AGV Transportation System

Fig. 3 shows the architecture of the simulated AGV transportation system.

For each AGV in the simulated system, an instance of an AGV agent with a local virtual environment is provided. Transports are represented by transport agents that share a common local virtual environment. As such, each node of the physical system is represented by a separate set of software components, matching the decentralized control architecture of the real system. The simulated environment provides a software representation of the physical elements of the AGV transportation system, including a map of the warehouse layout, a representation of the vehicles with the low-level control software (vehicles are located on the map), and a simulation of the wireless communication network. Besides maintaining the state of the simulated elements of the physical system, the simulated environment collects and delivers messages, it provide the means for monitoring the state of elements in the system, and it determines the outcome of the actuator actions of the agents taking into account the state of the system.

Assumptions. Simulations are performed on a layout of a real AGV system that is implemented by Egemin. We also use the standard transport profiles

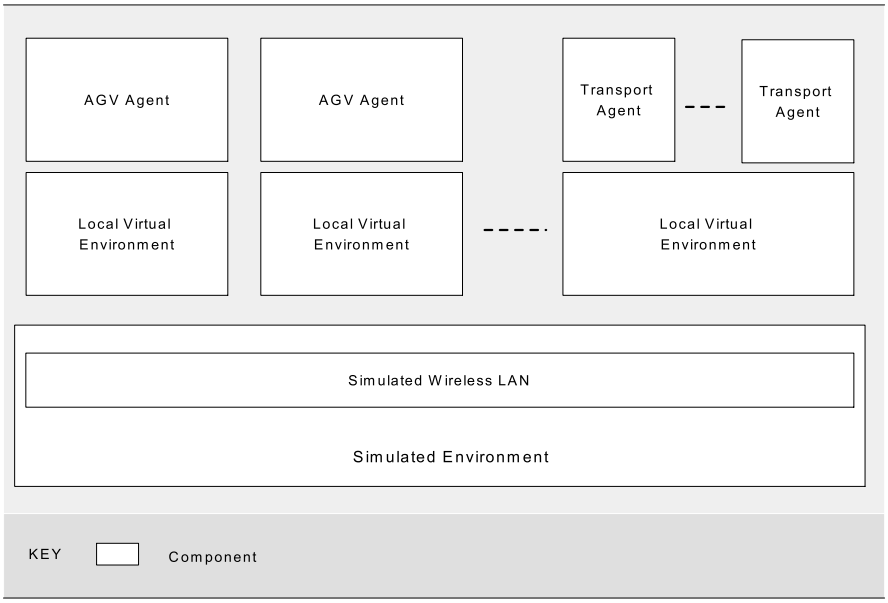

Figure 3. Architecture of the simulated AGV transportation system

that Egemin uses for testing purposes. However, the simulation makes also a number of simplifications. First, physical distribution is simulated. AGV movements, load manipulations, and message transport via the wireless network are simulated. Next, we do not consider physical errors in the system such as AGVs that fail, paths that are obstructed, etc. Finally, a number of non-functional concerns are not considered, such as charging of the batteries of AGVs, calibration of the vehicles, and persistency of data to recover from failures. It is common practice when testing specific properties of AGV transportation to make such abstractions. It allows to focus on the concern under test—in our case task assignment.

3. FiTA and DynCNET

We now introduce the two approaches for task assignment that enable agents to deal with dynamics during delayed commencement of tasks. We start with FiTA. First, we explain how fields are spread in the environment and how agents are guided by the fields toward tasks. Then, we give an overview of the software architecture that shows the main components of the AGV agent and the local virtual environment involved in FiTA. Next, we discuss DynCNET. We start by explaining the default sequence of protocol steps with an AUML interaction diagram. Then, we use a state chart to zoom in on the

adaptive properties of DynCNET. We use scenarios of the AGV application to illustrate the explanation of both approaches.

3.1. FiTA

Coordination by means of computational fields is a well-studied domain in MAS. The basic idea of field-based task assignment is to let each idle agent follow the gradient of a field that guides it toward a task that has to be executed. The agents continuously reconsider the situation in the environment and task assignment is delayed until the execution of the task starts, which benefits the flexibility of the system. To explain FiTA, we use the scenario shown in Fig. 4.

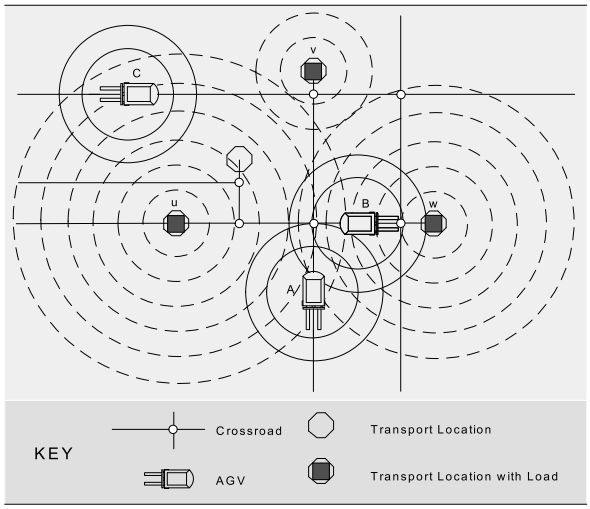

Figure 4. Example scenario in the AGV transportation system to illustrate FiTA.

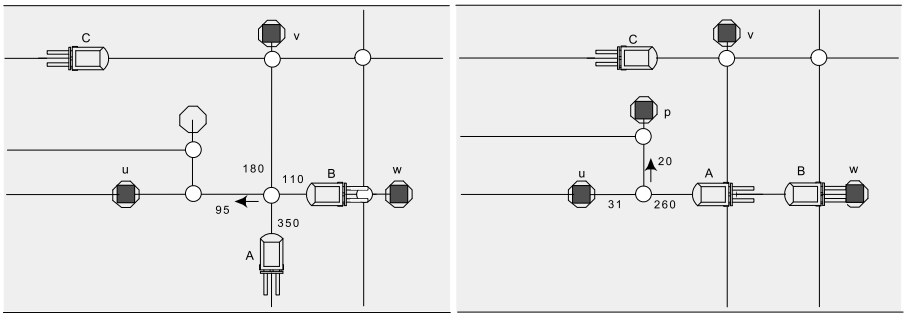

Figure 5. Two successive scenarios in which AGV A follows the gradient of the combined fields. For clarity, we have not drawn the fields. We have used the same key as in Fig. 4.

Fields. Task assignment is achieved by the interaction between AGV agents and transport agents. An AGV has a position in the warehouse that is represented in the local virtual environment. In a real AGV transportation system transport agents execute on the transport base. Conceptually, however, transport agents are situated in the local virtual environment and occupy the position of the load of its associated transport in the local virtual environment. The positions of AGVs and transports is shared among the local virtual environments. In the simulation, the actual positions of AGVs and transports is represented in the simulated environment. Both AGV and transport agents emit fields in the local virtual environment, see Fig. 4. 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. AGV fields have a fixed range, while the range of transport fields is variable and depends on the actual priority of the tasks. Fields are refreshed at regular times, according to a predefined refresh rate.

AGV agents store perceived 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 perceived 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. 4 the field-cache of AGV A will consists of three entries, one for transport *u*, one for transport *w*, and one for AGV B.

AGV agents calculate the coordination-fields. Each AGV agent calculates a *coordination-field* to decide in which direction to drive. A coordination-field is a combination of the perceived 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 coordination-field. The coordination-field is constructed from the next node on the AGV's path. An AGV agent follows the coordination-field in the direction of the smallest value. This can be considered as following the *gradient* of the coordination-field downhill. The coordination-field is computed as follows:

$$F_{calc} = min|_{j \in out\_nodes} \left(\delta \sum_{i=1}^{n_T} F_{i,j}(1+\phi_i) + (1-\delta) \sum_{k=1}^{n_A} F_{k,j}(1+\phi_k) \right)$$

The formula calculates the minimum of a set of combined fields from a particular node on the warehouse layout. For each possible direction the AGV can move from this node, the formula computes the sum of the fields (the first term sums the transport fields sensed by the AGV, the second term sums the sensed AGV fields) and then selects the minimum. The formula allows to determine the influence of various parameters such as the freshness of the

fields, and the balance between attracting and repelling fields. Concretely, F_{calc} is the selected coordination-field from the next node on the AGV's path. out_nodes is the set of outgoing nodes from the next node. n_T is the current number of entries of transport fields in the field cache, and n_A the number entries of AGV fields. δ is a weight coefficient that determines the contribution of transport fields relative to AGV fields. $F_{i,j}$ is the field strength of transport *i* of the field cache on the next node via node *j*. ϕ_i is the freshness coefficient of the sensed field of transport *i*. $F_{k,j}$ is the field strength of AGV *k* of the field cache on the next node via node *j*, and ϕ_k is the freshness coefficient of the sensed field of AGV *k*.

$F_{i,j}$ is computed as follows:

$$F_{i,j} = \frac{Router(l_i, j)}{p_i}$$

$Router(l_i, j)$ calculates the shortest path distance from l_i, the location of transport i, to the next node on the AGV's path via node j. p_i is the actual priority of transport i.

$F_{k,j}$ is computed as follows:

$$F_{k,j} = Router(l_k, j)$$

$Router(l_k, j)$ determines the shortest path distance from l_k, the location of AGV k, to the next node of the AGV via node j.

As an illustration, in the left part of Fig. 5, AGV A calculates the coordination-field on the node in front. Although transport *w* is closer, the coordination-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 task assignment is delayed until an AGV actually reaches a pick location and picks up the load. This allows agents to adapt the assignment of tasks while the AGVs drive toward loads. By delaying task assignment, FiTA can cope with changing circumstances that arise. An example is shown in the right part of Fig. 5 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*.

Software Architecture. Now, we discuss the main components of the AGV agent and the local virtual environment in FiTA. Fig. 2 shows an architectural view with collaborating components. Transport agents have a similar but more simple decision making component as AGV agents since these

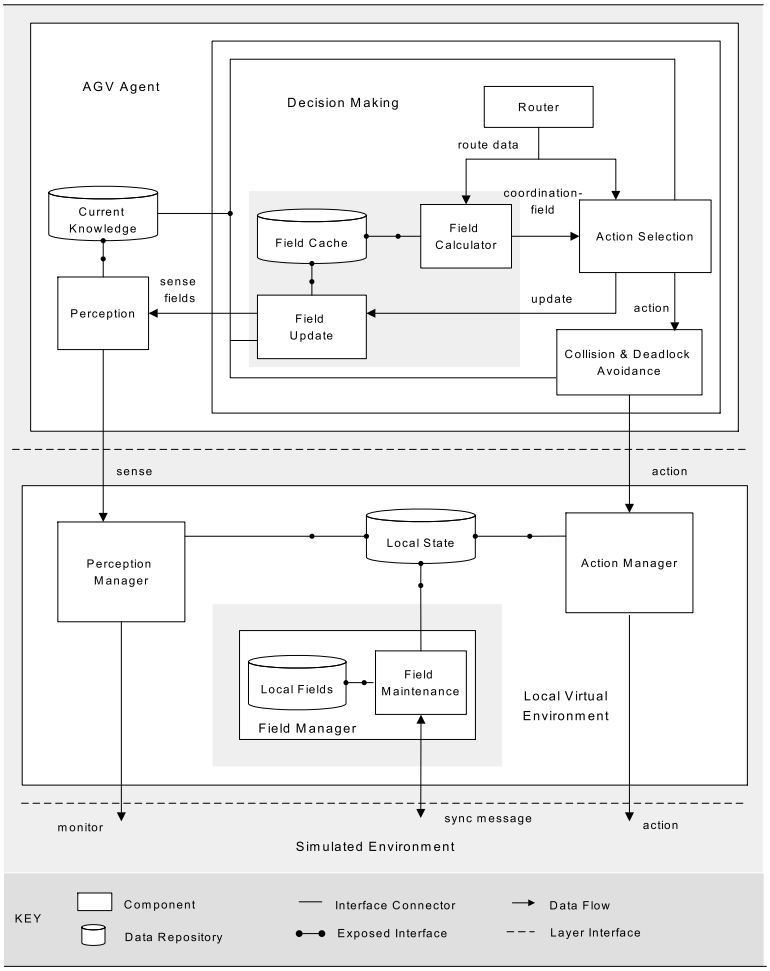

Figure 6. Software components of AGV agent and local virtual environment involved in FiTA. The elements in the shaded area of the local virtual environment deal with field management and the elements of the AGV agent deal with field calculation.

agents only have to deal with emitting fields. First, we discuss the various components of the AGV agent that deal with field calculation. Then, we zoom in on the components of the local virtual environment that deal with field management. In (Weyns et al., 2006), we elaborate on the architecture of the AGV transportation system.

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

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 (Hart et al., 1968).

However, the approach is compatible with a dynamic router that would take into account dynamic runtime information such as traffic distribution.

Field calculator. The field calculator computes the coordination-field from the last selected target node by combining the perceived fields from the field-cache. The higher the freshness of a cache-entry, the more influence the field associated with the cache-entry will have on the construction of the coordination-field. Thus, although still used, less importance is given to outdated information. The field calculator makes use of the router to calculate the values of the coordination-field in different directions. The AGV follows the gradient of the coordination-field downhill as driving direction.

Field update. The field update component requests perception updates (via the Perception component) to update the field cache of the AGV agent. Field update requests are periodically invoked by the action selection component.

Action selection. The action selection component continuously reconsiders the dynamic conditions in the environment and selects appropriate actions to perform the agent's tasks. We illustrate action selection of the AGV agent with a number of example rules:[2]

```
{Action selection rules of AGV agent}
 R1: (ready-to-pick) -> {action = pick}
 R2: (reserved-path < LookAheadDistance)
       -> { compute coordination-field;
            action = reserve-node }
 R3: (ready-to-move) -> {action = move}
```

Rule R1 states that the AGV agent selects a pick action when the AGV is ready to pick a load. Rule R2 states that the AGV agent reserves a next node on its way to a load if the current length of its reserved path is less than the predefined path length LookAheadDistance. Locking the path in advance according to the LockAheadDistance parameter ensures that the AGV moves smoothly and stops safely. The third rule states that the AGV agent selects a move action if it is ready to move on.

Action selection passes the selected action to the Collision & Deadlock Avoidance component. If applicable, this component locks the required path to execute the selected action. As soon as the path is locked, the action is invoked in the local virtual environment. When the AGV has picked up a load, it will inform the transport agent and execute the transport. The following two high-level descriptions summarize the behavior of the agents during task assignment:

```
{Action selection AGV agent}
while idle
```

[2] The format of the rules is defined as:
(condition) → {optional computation; selected action}

```
 do repeat with constant frequency {
  1. Sense fields and update the field-cache
  2. Select action
  3. Perform action in local virtual environment
}

{Action selection transport agent}
while not assigned
 do repeat with constant frequency {
  1. Calculate priority
  2. Update status in the local virtual environment
}
```

Local fields. This repository of the local virtual environment stores the values of fields of AGVs and transports.

Field maintenance. The local virtual environment is responsible for spreading the fields. Field maintenance encapsulates a dynamic process that maintains the local fields. It takes into account the status of the local agent(s) such as the position of an AGV or the priorities of transports and the information about AGVs and transports received from other local virtual environments. This latter information is exchanged among local virtual environments via synchronization messages.

Dealing with Local Minima. A well known problem with field-based approaches is the problem of local minima (Koren and Borenstein, 1991). We explain how FiTA deals with two common causes of local minima: the topology of the layout and the neutralization of fields.

Since AGV vehicles are restricted to follow predefined paths in the environment, the problems with local minima caused by the topology of the layout could be avoided relatively easily. Consider Fig. 7 with AGV A and two transports u and v. If the value of the fields would be based on Euclidean distance, AGV A would drive towards transport u, however, it would be trapped in a local minimum at node 1.

By making the strength of the field on a particular position proportional to the shortest path distance between this position and the source of the field, local minima are avoided. Applied to the example in Fig. 7: since the shortest path distance from AGV A to transport v is much smaller as to transport `u`, the attracting field of transport v will be much smaller than that of transport u. As such, AGV A will turn right at node 1 (gradient downhill) and drive towards transport v.

A local minimum can also arise when the attracting fields and the repelling fields sensed by an AGV neutralize each other. Consider the situation in the left part of Fig. 8. When AGV A computes its coordination-field from node 2, the attracting fields of transport v and w may be equal and smaller than the

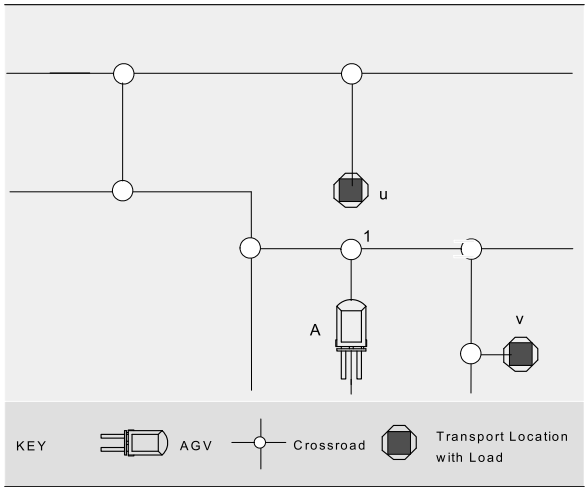

Figure 7. Avoidance of local minima in FiTA. The attracting fields of transports *u* and *v* are proportional to the shortest path distance between AGV A and the transports. As such, AGV A will be guided towards transport *v*.

field of transport *u*. In such a case, the AGV will select randomly one of the minimum fields to follow its route. Another situation is shown in the right part of Fig. 8. In this case, the attracting field of transport *x* may accidentally be equal to the sum of the attracting field of transport *y* and the repelling field of AGV B. Again, the AGV will select one of the options.

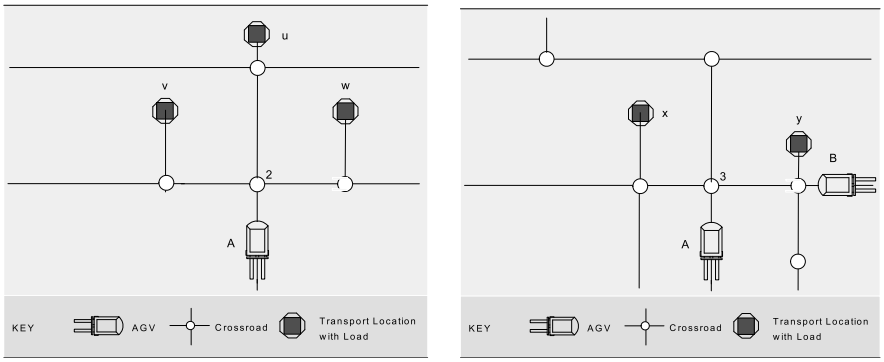

Figure 8. Left: AGV A selects randomly between task *v* and *w* in node 2. Right: Temporal neutralization of fields in node 3.

3.2. DynCNET Protocol

DynCNET is an extension of the well-known CNET protocol that enables the agents to regularly reconsider the situation in the environment and adapt

the assignment of tasks when circumstances change. We start by explaining a number of general properties of the DynCNET protocol. Then we give an overview of the default sequence of the protocol. Next we explain how the agents involved in a protocol can switch the assignment of tasks. We will use the AGV transportation scenario depicted in Fig. 10 to illustrate the steps of the protocol. The DynCNET protocol describes the behavior of AGV agents and transport agents to realize adaptive task assignment. This behavior is encapsulated by the agents' communication module, see Fig. 2.

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. As an example, consider the scenario shown in Fig. 10. In the AGV transportation system, an initiator corresponds with a transport agent that represents a task 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. 10 show the current areas of interest of AGV A (top) and task x (bottom). For task x, there are currently two candidate AGVs to execute the task: F and G (AGV E is delivering a load). For AGV A on the other hand, there are three possible tasks to execute: u, v, and w. Because of the dynamics in the system, the set of candidate tasks (initiators) and agents that can execute a task (participants) can change over time. For example, in the right part of Fig. 11, AGV E has just dropped its load and becomes a candidate to execute task x.

Default Sequence. Fig. 9 shows an AUML interaction diagram (Huget et al., 2006) 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 provisional agreement between initiator and participant, and the possible revision of the assignment of the task 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 of Fig. 12. This state diagram shows a compact representation of the behavior of the initiator and participant agents in the protocol. When a task enters the system and it is ready to be executed (`task-ready`), the corresponding initiator enters the `Active` state in which it remains until the task is completed (`task-completed`). As soon as a participant is

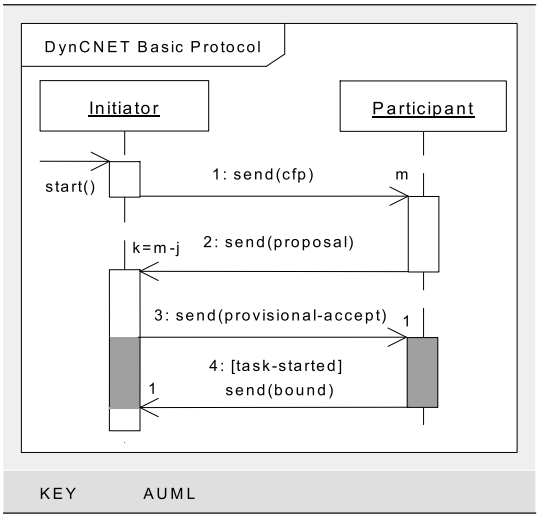

Figure 9. High-level diagram of the DynCNET protocol. Shaded zones in the activation boxes represent periods in the protocol when agents can switch the provisional agreement.

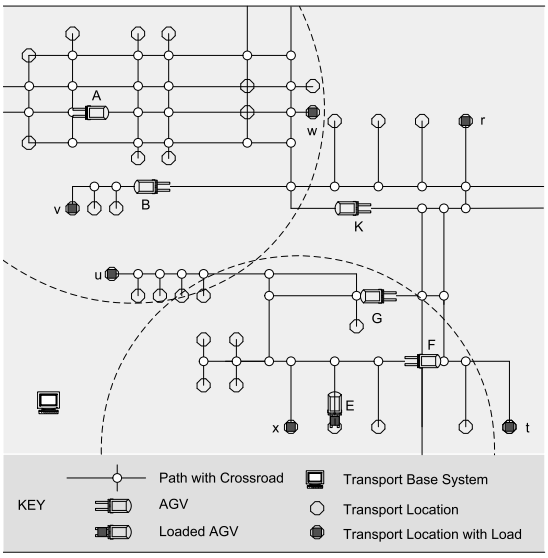

Figure 10. Scenario to illustrate DynCNET. The dotted circle at the top left demarcates the current area of interest of AGV A. The circle at the bottom demarcates the current area of interest of task x.

`ready-to-work` it enters the `Working` state in which it remains until the task is executed (`ready`). To explain the adaptability of DynCNET, we first look at the protocol from the perspective of the participant, then we look

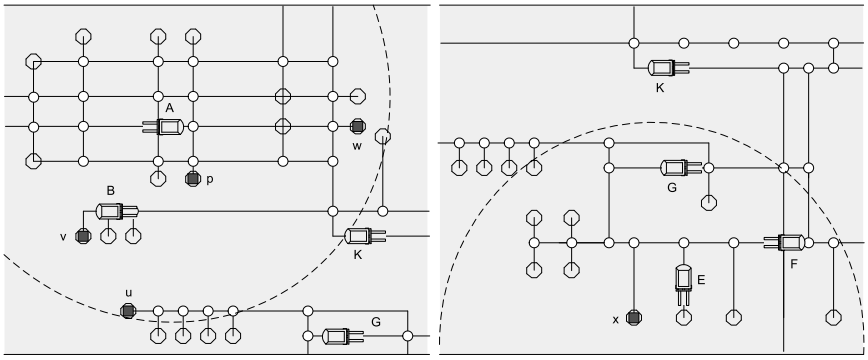

Figure 11. Left: task *p* provides an opportunity for AGV A to switch tasks. Right: AGV E becomes available for task *x*. We have used the same key as in Fig. 4.

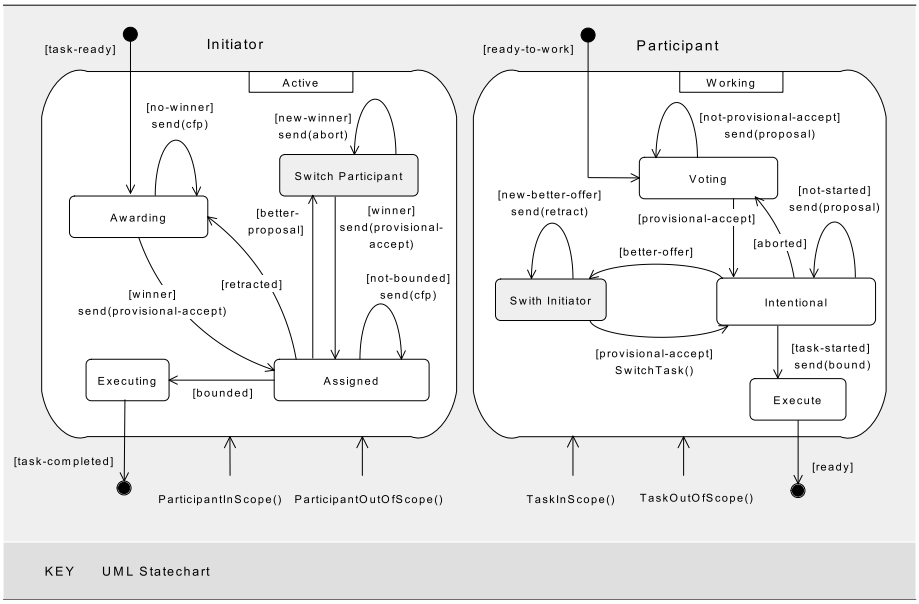

Figure 12. High-level description of the DynCNET protocol. In the shaded states, agents can switch the provisional agreement.

from the point of view of the initiator.

Switching Initiators. Consider the situation in Fig. 10 where we assume that AGV A has a provisional agreement to execute task w. While AGV A drives toward the pick location of task w, a new task p enters the system, see the left part of Fig. 11. This new task is an opportunity for AGV A to switch tasks. DynCNET enables participants to switch initiators and exploit such opportunities. When a participant is ready to execute a task, it enters the `Voting` state. As long as the participant has not received a provisional

acceptance (not–provisional–accept), it answers cfp's with proposals. When the participant receives a provisional–accept message (step 3 in Fig. 9), it enters the Intentional state, see Fig. 12. As soon as the participant starts the task (task–started), it sends a bound message to the initiator to inform the latter that the execution of the task is started. The participant is then committed to execute the task.[3] 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. Based on the new–better–offer, the participant retracts from the earlier provisional task assignment (send(retract)), and switches (provisional–accept holds) to the more suitable task (SwitchTask()) entering again the Intentional state.

Switching Participants. Consider the situation in Fig. 10 where we assume that task x has a provisional agreement with AGV G, and task t with AGV F. While AGV G drives toward the pick location of task x, AGV E drops the load it is carrying and becomes available, see the right part of Fig. 11. This new AGV is an opportunity for transport x to switch AGVs. DynCNET enables initiators to switch participants and exploit such opportunities. As long as the initiator has not selected a participant to execute the task (no–winner), it sends cfp's to the participants in scope. Based on the received proposals from the participants, it selects a winner, sends a provisional–accept message (step 3 in Fig. 9), and enters the Assigned state, see Fig. 12. As soon as the initiator receives a bound message from the selected participant (bounded holds), 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 proposal from a participant (better–proposal), the initiator changes to the Switch Participant state. Based on the condition new–winner in this state the initiator sends an abort message to the provisionally assigned participant, and sends a provisional–accept message to switch to the more suitable participant (new winner).

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, see Fig. 2. Similarly, the functions ParticipantInScope() and ParticipantOutOfScope() notify the initiator when new participants enter and leave its area of interest. In the AGV transportation system, monitoring of the area of interest of tasks and AGVs is supported by views provided by the ObjectPlaces middleware (Boucké et al., 2006). In the

[3] The condition bounded initiates the initiator's state transition from Assigned to Executing when it receives the bound message from the participant.

simulation, these functionalities are included in the simulated environment.

Convergence. A potential risk of DynCNET is that the assignment of tasks oscillates between participants and no tasks are executed. To ensure progress, both temporal and spatial windows are used in the protocol. Temporal windows refer to the periodicity by which call for proposals and proposals are sent. In the AGV application, the transport agents use the same frequency to send call for proposals as the AGV agents use to send proposals in the Voting state. However, as soon as an AGV agent enters the Intentional state it increases the frequency with a factor 5, providing the transport agent of the load it intends to pick with up-to-date information about the approaching AGV. Spatial windows refer to the size of the areas of interest for initiators and participants. In the AGV application, the area of interests of transport agents (initiators) covered up to 1/10th of the total area of the map (depending on the actual priority of the transports) and the area of AGV agents (participants) was 4 times smaller as that of transport agents. As a result of the asymmetry, transport agents select preferable AGVs, while AGV agents only switch transports when a new transport appears close to the actual position of the AGV.

The choice for the specific parameter settings was determined before the simulation tests, see section 4.1. The results of the experiments show that the protocol converges under the indicated conditions for the studied case. However, the specific solution for convergence applied in the AGV application can not be generalized. A formal proof of convergence for DynCNET is required which is subject of our future work. A possible starting point to produce such a proof is described in (Aknine et al., 2004). In that paper, the authors formally prove the termination of an adapted CNET protocol.

Synchronization messages. To avoid overloaded diagrams, we have made abstraction of a number of synchronization issues in the high-level description of the DynCNET protocol in Fig. 12. We explain the main synchronization problem that is related to a participant that has started executing a task while an initiator has sent an abort message to that participant.

When an initiator receives a better proposal it switches participants (`better_proposal`). Therefore, it sends an `abort` message to the participant that has provisionally accepted, see Fig. 12. However, this participant may already have started executing the task (`task-started`) while the `bound` message of that participant has not been received by the initiator yet. Since the participant has started executing the task (e.g., an AGV has started to pick the load), participants can no longer be switched.

To deal with this synchronization problem, an additional synchronization message is used, see Fig. 13. When the initiator receives a better proposal, it enters the `Aborting` state where it sends an `abort` message to the participant that has provisionally accepted to execute the task. The initiator then

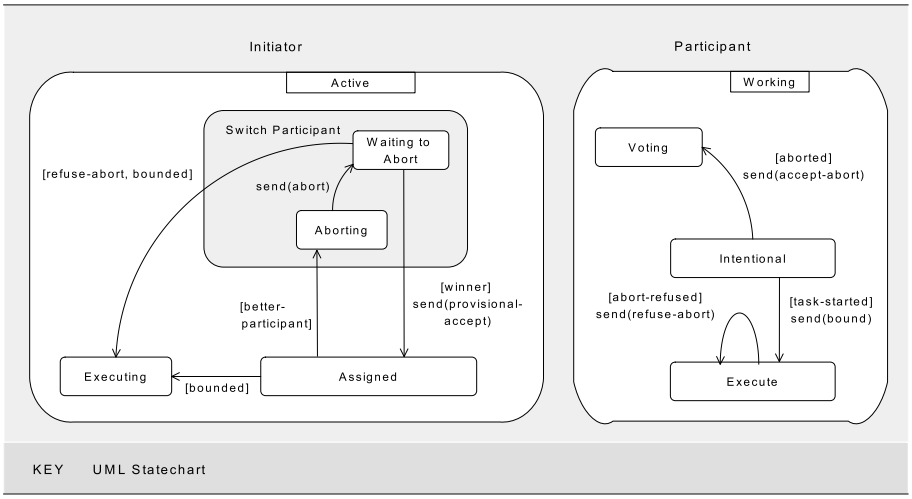

Figure 13. Synchronization of bound and abort (refined part of the DynCNET protocol description of Fig. 12).

enters the `Waiting to Abort` state. In case the participant has not started to execute the task it accepts (`aborted`) and replies with an `accept–` abort message. However, if it has started executing the task (`task–started`), it refuses (`abort–refused`) and sends a `refuse–abort` message to the initiator. When the initiator receives the abort confirmation it switches participants (`winner`) and sends a provisional accept message to the participant with the better proposal. Otherwise, it waits for the `bound` message of the initial participant (`bounded`). As soon as it receives the confirmation, it enters the `Executing` state.

Other synchronization issues are related to participants that leave the scope of interest of initiators. For details, we refer to (Weyns et al., 2007).

4. FiTA and DynCNET Applied in an AGV Transportation System

This section discusses the test results obtained from applying DynCNET and FiTA in a simulated AGV transportation system. After introducing the test setting, we present the results of the various tests and we reflect on the test results. For a number of additional test results we refer to (Weyns et al., 2007).

4.1. Test Setting

AGV Transportation System. All simulation tests are performed on the map of an AGV transportation system that is implemented by Egemin at

EuroBaltic, a fishing prcocession center in Rugen, Germany, see Fig. 14. 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

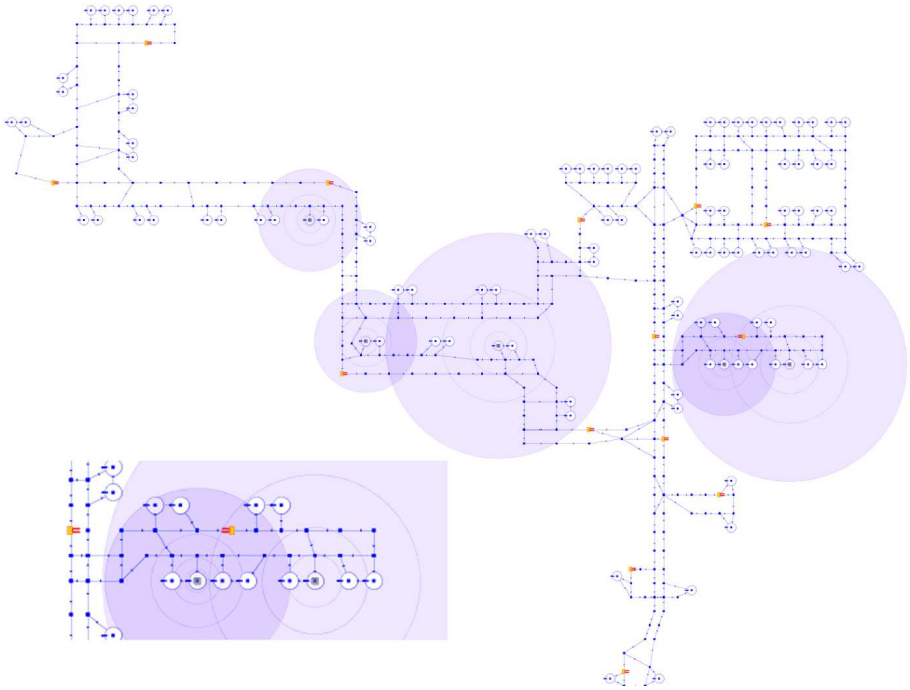

Figure 14. Test map of the AGV transportation system that is implemented by Egemin at EuroBaltic. The snapshot is taken from a test with FiTA. The part at the bottom left zooms in on a small part of the map.

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 random priority that increases over time. In the simulation, we used 14 AGVs just as in the real application. The average speed of driving AGVs is 0.7 m/s, while pick and drop actions take an average amount of time of 5 s. Every simulation[4] 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 30 simulation runs.

Reference Protocol. In the tests, we use standard CNET as a reference protocol. In CNET, an initiator calls for proposals and participants offer proposals to perform the task. When the initiator has received the proposals from all

[4] For the tests, we used an AGV simulator (http://www.cs.kuleuven.ac.be/∼distrinet/task-forces/agentwise/agvsimulator/) that uses a framework for time management (Helleboogh et al., 2005) to ensure that the simulation results are independent of performance characteristics of the execution platform. Tests where executed on a cluster of 40 machines: P4 2Ghz, 512MB RAM, Debian Stable 3.0.

participants, it evaluates the proposals and assigns the task to the participant with the best offer. In the tests, a transport that enters the system is assigned as soon as possible to the most suitable AGV, i.e. an idle AGV for which the cost to reach the pick location is minimal. When transports can not be assigned immediately, they enter a waiting status. All waiting transports are ordered by priority, and this priority determines the order in which transports are assigned.

Parameter Settings. Preceding to the tests, we determined the most suitable set of values for the parameters of the three tested task assignment approaches. Parameter tuning is a labour-intensive job. For most of the parameters, for FiTA as well as for DynCNET, we were able to determine a range of values from which we could select one without significantly affecting the performance of the protocol. We presume that the constrained nature of the problem (in particular the restrictions imposed by the layout) accounts for this relaxation. A thorough discussion of parameter setting however, would lead us too far. In (Weyns et al., 2007), parameter setting is discussed in depth.

4.2. Test Results

We focus on the evaluation of two important properties of the task assignment approaches: performance and robustness to message loss. Performance evaluation consists of two parts: communication load and completion of tasks over time. Communication load (number of messages sent per transport) is a crucial factor in agent systems since decentralization of control requires more communication and thus additional bandwidth (Ong, 2003). Evaluation of the completion of tasks over time is important to demonstrate the flexibility of the task assignment approaches. To evaluate the completion of tasks over time, we measured 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). Besides the test with a standard test profile, we have performed a stress test in which AGVs have to handle as quickly as possible a fixed number of transports from a limited number of locations. Robustness to message loss is another important criterion in decentralized systems, in particular in mobile systems that communicate via a wireless network. DynCNET is not robust to message loss since the protocol prescribes a particular sequence of message exchange. When a message get lost, this sequence is disrupted and the interaction blocks[5]. Therefore, we have only tested robustness to message loss of FiTA. To demonstrate the robustness to message loss, we have measured the reaction time and throughput for different degrees of message loss. (Weyns et al., 2007) discusses a number of additional tests.

[5] In fact, some of the messages may get lost without blocking the interaction. For example, the protocol will not fail when a call for proposals message is lost.

Since tasks are generated randomly and priorities are assigned randomly, we have verified the the statistical significance by calculating the 95% confidence interval (Weisstein, 2006) for the main test results. The confidence intervals are denoted with error bars in the figures. The relative 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. 15 shows the results of the test. The number of messages of

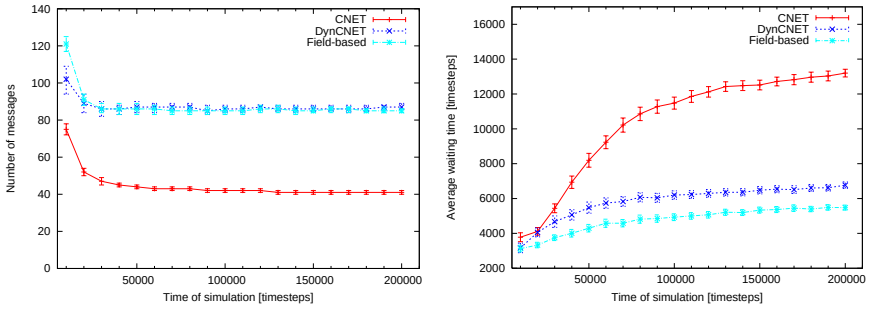

Figure 15. Left: amount of messages being sent per finished transport. Right: average waiting time

DynCNET and FiTA are approximately the same, while the communication load of CNET is about half of the load of the dynamic mechanisms. However, an important difference exists between the type of messages sent. Fig. 16 summarizes the number of unicast and broadcast messages sent by the three mechanisms. For CNET, more than 90 % of the communication are unicast

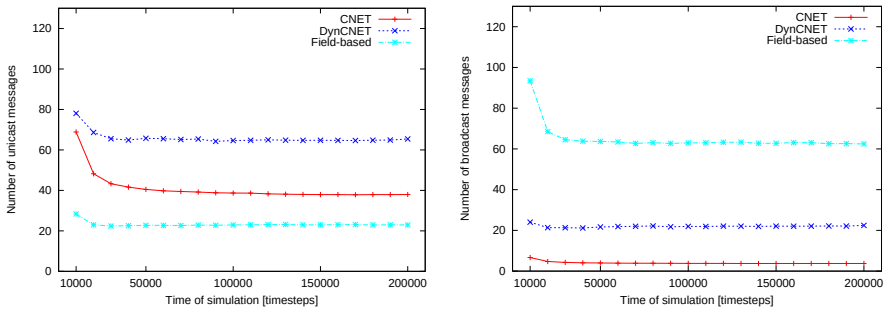

Figure 16. Left: number of unicast messages. Right: number of broadcast messages

messages. For DynCNET the balance unicast–broadcast messages is about 75 % – 25 %, yet, for FiTA this balance is about 25 % – 75 %. This difference is an important factor for selecting appropriate communication infrastructure for a particular task assignment mechanism and vice versa.

Average Waiting Time. The right part of Fig. 15 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 of approximately 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.

Number of Finished Transports. The left part of Fig. 17 shows the number of transports finished by each of the protocols during the test run. The re-

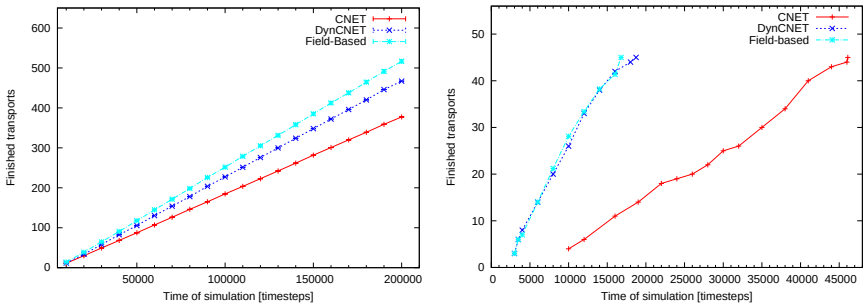

Figure 17. Left: number of finished transports. Right: number of finished transports in the stress test

sults confirm the measures of the average waiting time per finished transport. DynCNET handles more transports than CNET, but less than FiTA. After four hours in real-time, on average, CNET has handled 380 transports, DynCNET 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.

Stress Test. In addition to the standard transport test profile, we have performed a stress test in which 45 transports are created at a limited number of locations in the beginning of the test. These transports have to be dropped at a particular set of destinations. The test simulates for example the arrival of a truck with loads that have to be distributed in a warehouse. The task of the AGVs is to bring the loads as quickly as possible to the right destinations. The transport test profiles for the three mechanisms was identical. The right part of Fig. 17 shows the test results. The slopes of the curves of FiTA and DynCNET are similar but much steeper than the curve of CNET. The results demonstrate that CNET requires about 2.5 times more time to complete the 45 transports than the adaptive approaches.

Robustness to Message Loss. DynCNET is not robust to message loss since

the protocol prescribes a particular sequence of message exchange that can not be disrupted without blocking the interaction. Therefore, we have only tested robustness to message loss of FiTA. To demonstrate the robustness, we have measured the reaction time and throughput for different degrees of message loss. The left part of Fig. 18 shows the average waiting time per finished transport for different percentages of message loss. The right part of the figure shows the corresponding number of finished transports over time.

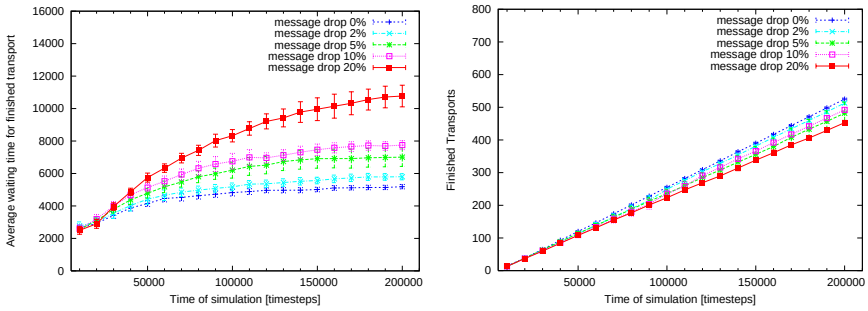

Figure 18. Left: average waiting time for different percentages of message loss. Right: the corresponding number of finished transports

The test results show a graceful degradation of the performance of FiTA with increasing message loss. The average waiting time of transports systematically increases and the number of finished transports over time decrease with higher message loss rates. In practical AGV transportation system, message loss in typical 1–2 % with a maximum of 5 %. The test resuls show that the impact of message loss of 2 % is fairly limited. Even with 20 % message loss, FiTA performs still better as CNET without message loss (compare the left part of Fig. 18 with the right part of Fig. 15, and the right part of Fig. 18 with the left part of Fig. 17.

4.3. TRADEOFF ANALYSIS

We now reflect on the test results and make a tradeoff analysis of the approaches for task assignment. First we zoom in on a number of important quality properties. Then we compare a number of engineering aspects.

Quality Attributes. DynCNET and FiTA have similar performance characteristics. Both outperform CNET on all performance measures, the cost is a doubling of required bandwidth. Since DynCNET explicitly defines the mechanism for agents to switch tasks, we expected that–when fine tuned well–DynCNET would be able to outperform FiTA. However, the experiments show that this is not the case, at best DynCNET is able to equal the performance of FiTA. Fig. 19 compares several additional quality attributes

	Flexibility during delayed commencement	*Openness during delayed commencement*	*Robustness to message loss*
CNET	No: one shot assignment of tasks	Not supported	Requires additional support
FiTA	Yes: combination of fields determine the participants' choices	Inherent to the approach	Inherent to the approach
DynCNET	Yes: explicit points of choice for initiators and participants	Explicitly built-in	Requires additional support

Figure 19. Summary of quality attributes of the three approaches.

of the three task assignment approaches.

Flexibility. We consider flexibility as the agents' ability to adapt their behavior to dynamics that happen in the process of task assignment. Both DynCNET and FiTA support flexible assigning of tasks with delayed commencement. In FiTA, the choices of the participant agents are implicitly determined by the combination of the sensed fields. DynCNET provides explicit points of choice for initiators and participants. The points of choice are abstractly defined in the protocol and need to be instantiated according to the requirements of an application at hand. In the AGV application, agents use the priorities of tasks and the distance between AGVs and loads to switch tasks. More advanced approaches can be considered, e.g., participants may (to some extent) favour tasks that are located near other tasks increasing the chance to find a closely located task when the original assignment of tasks for some reason switches.

Openness. With openness, we refer to the agents' self-managing abilities to take into account other agents that enter and leave the system in the process of task assignment. Both DynCNET and FiTA support openness during delayed commencement, i.e. both mechanisms allow initiators to take into account new participants that become available and participants can participate in the assignment of new tasks that become available. 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 current area of interest. Neither flexibility, nor openness is supported by CNET.

Robustness. 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 perceived 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 value of the field is used. Yet, to determine the combined field that guides the agent, less importance is given to older field values. 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 requires additional support for robustness to message loss. Exception handling in protocol design is a non-trivial problem (Mallya and Singh, 2005) and may cause a significant increase of message exchange.

Engineering Aspects. Fig. 20 compares a number of engineering aspects of the three task assignment approaches.

	Mechanism engineering	*Parameter tuning*	*Type of communication*
CNET	Use of common engineering diagrams	Limited number of parameters; tuning is easy.	Mainly unicast
FiTA	No common engineering approaches available	Various parameters; tuning requires considerable efforts	Mainly broadcast
DynCNET	Use of common engineering diagrams	Various parameters; tuning requires considerable efforts	Mainly unicast

Figure 20. Summary of engineering aspects of the three approaches.

Mechanism Engineering. No common engineering approaches are currently available for designing and developing FiTA. On the other hand, DynCNET allows to specify the behavior of the agents by means of common engineering diagrams such as interaction diagrams and state charts. We used UniMod (UNiMod, 2006) to design the DynCNET protocol as a state machine. UniMod enables to draw the state machine and export the diagram to an XML file. This XML file was used to interpret the state machine in the agent program.

Parameter Tuning. Parameter tuning is typically associated with stigmergy-based solutions such as FiTA. However, parameter tuning of DynCNET requires similar efforts as in FiTA. Examples are the range of interest of both types of agents, the growth rate to extend this range when no suitable candidates are found, the pace to send cfp and proposals, etc. Our experiences indicate that a flexible agent-interaction protocol that deals with dynamics and change in the system also requires considerable efforts to tune parame-

ters.

Type of Communication. A significant difference exists in the ratio unicast–broadcast messages that are used in the three task assignment mechanisms. This difference is important for selecting appropriate communication infrastructure for a specific task assignment mechanism and vice versa. Further research is needed to investigate the implications of the type of communication of the different mechanisms.

5. Related Work

Task assignment is an extensive domain of research. In this section, we discuss a number of related field-based approaches and mechanisms for task assignment that are based on CNET.

5.1. Field-Based Approaches

Techniques based on fields have been used extensively for the coordination of software agents in a metric space. Zeghal and Ferber (Zeghal and Ferber, 1993) use vector fields to control the landing and movements of a large group of aircrafts in a simulation. In this approach, each agent is guided by a potential field that it constructs based on attracting and repelling forces resulting from goals and obstacles (including other agents) respectively. Reynolds demonstrates flocking behavior between a set of agents (Reynolds, 1996). The aggregate behavior of the agents emerges from the interaction of multiple agents that each follows a set of simple behavioral rules. Mataric adopted these techniques to real robots (Mataric, 1994), showing how a set of robots produced pack behavior. Each robot is provided with a set of simple behaviors from which it selects the most suitable behavior according to its current environmental context, i.e. its current position relative to other robots.

More recently, Mamei and colleagues apply the idea of flocking behavior of birds to guards who have to patrol a museum, keeping a certain distance between each other to cover more ground (Mamei et al., 2004). The tourists in the museum are provided with a software agent running on some wireless handheld device, giving suggestions on how and where to move. A computer network with a topology that mimics the topology of the museum plan is embedded in the museum walls. The hosts in the network are associated to each room and are capable of communicating with each other and the mobile devices in their proximity. Fields can be injected at a host, after which they will be diffused hop-by-hop across the network, modulating the values of the fields as necessary. The latter is achieved by supporting middleware, such a TOTA (Tuples On The Air) (Mamei and Zambonelli, 2004).

A related example in which fields are successfully applied is the"Quake 3" game (Mamei and Zambonelli, 2003). In this application, fields are used to guide the behavior of non-human characters in a virtual game.

Shehory and colleagues (Shehory et al., 1999) describe a physics oriented model for cooperative goal-satisfaction with simple agents in a large-scale cooperative MAS. Different kinds of gradient field approaches have also been used in the context of RoboCup, a recent example is (Buchman et al., 2005). Breton and colleagues (Breton et al., 2004) discuss 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 systems. Another variation is described in (Paoli and Vizzari, 2002), where the MMASS (Multilayered Multi Agent Situated System) model for multi-agent coordination is used. In this model, the environment is represented as a multi-layered graph in which the agents can spread abstract fields. In the standard field-based approach, agents combine perceived fields and are constantly guided by the fields, while in MMASS fields are in principle considered independent from each other and are exploited only to trigger one shot reactions.

Finally, field-based approaches are an extensively studied field in robotics. A lot of attention in this research is given to solving particular problems such as trap situations due to local minima, the problem of passage between closely spaced obstacles, oscillations in the presence of obstacles and in narrow passages, the coordinated movement of multiple robots, etc. Because AGVs are constrained in their movements to a predefined layout, these problems are not applicable to AGV transportation systems. In contrast, the main problem we have tackled in this paper is adaptive task assignment among a set of AGVs. For more details about field-based approaches in robotics, we refer to a number of reference books (Borenstein et al., 1996, Arkin, 1998, Donald et al., 2000).

5.2. CNET-Based Approaches

Contract Net (CNET) is a widely known and extensively used protocol that uses an auction-like mechanism to achieve task assignment. CNET was originally proposed by Smith and Davis (Smith, 1980) and is included in a FIPA-standard (FIPA TC Communication, 2002a). The DynCNET protocol is an extension to CNET with two distinctive characteristics: (1) it enables $m \times n$ negotiations, i.e. a participant can manage concurrently multiple negotiation processes with the initiators and the initiator can manage negotiation processes with multiple participants; (2) it supports changes on the Initiator and the Participant side (dynamism) and delays the definitive assignment until the task is effectively started (i.e. it provides support for delayed commencement of tasks). In this section, we list several variants to CNET and show how they relate to DynCNET.

Several extensions of CNET exist that support $m \times n$ negotiations. Knabe and colleagues (Knabe et al., 2002) describe a protocol that allows a bidder to place bids for multiple unassigned tasks. This is achieved by giving the bidder the option to refuse the task in step 3 of the original protocol (compare Fig. 9). The manager will then award the contract to the second best bidder, who can again accept it or refuse it, etc. This protocol does not support changes at both sides or delayed commencement.

Fisher and colleagues (Fischer et al., 1995) discuss an integrated multi-agent based approach for cooperative planning and scheduling of transportation tasks in a society of shipping companies. The authors introduce ECNP, the Extended Contract Net Protocol, in which grant and reject are replaced by temporal grant, definitive grant, temporal reject, and definitive reject. This allows to temporally assign a set of related bids to a set of agents that can be evaluated as a whole and then be assigned definitively (e.g., a set of trucks are assigned parts of a transport task and after the client agrees with the assignment, the task as a whole is allocated). To allow adaptation after the tasks are definitely assigned (e.g., trucks that have to deal with traffic jams), an auction-based mechanism called simulated trading is used. The approach was implemented at DFKI and it was shown that it was able to solve complex industrial scheduling problems. ECNP supports the allocation of composite tasks, something that is not considered for DynCNET in this paper. Whereas ECNP provides a limited time window during which the protocol allows to change the allocation of tasks, DynCNET supports the adaptation of task assignment until a task is actually started to be executed. In (Fischer et al., 1995), such dynamics is supported by an additional auction-based simulation mechanism.

In (Schillo et al., 2002), Schillo and colleagues consider the problem how agents participating in a contract net should allocate their resources if a large number of contract net protocols is performed concurrently. Agents should find a good strategy to allocate resources. The authors present several strategies for solving this problem and give criteria for the decision which of the strategies is best selected for a given problem domain. DynCNET allows agents to apply complex strategies to select tasks. However, in this paper, we only have considered simple strategies in which agents consider the assignment of one task at a time. A more advanced strategy could take into account the assignment of multiple tasks over time. On the other hand, in a highly dynamic setting as an AGV transportation system, defining such a strategy is a complex task.

FIPA proposes the Iterated CNET protocol, allowing multi-round iterative bidding (FIPA TC Communication, 2002b). The protocol is multi-round in the sense that the initiator can decide to issue a new call for proposals (and thus start a new round) after the bidding phase instead of accepting a proposal. There is no partial commitment during the iterations and

the protocol does not allow to reconsider the situation ones a proposal is accepted. Related to the Iterated CNET protocol are leveled-commitment contracts (Sandholm and Lesser, 2002). The idea of a leveled-commitment contract is that any of the agents in a contract can de-commit by paying an agreed penalty to the other agent(s) in the contract. A leveled-commitment contract is not a protocol on itself, but this type of contracts can used in negotiation protocols between self-interesting agents. De-comitting penalties could be added in the DynCNET protocol in the switching phase.

The protocol in (Aknine et al., 2004) focusses on self-interested agents supporting several negotiation processes in parallel. The approach introduces two levels of bidding, i.e. a pre-bidding phase in which the participants can still change their commitment and a definitive bidding phase in which the task is effectively assigned to a single participant. The protocol allows to reconsider commitments during the pre-bidding phase, but the window in which commitments can be changed is small. As soon as all agents answered in the pre-bidding phase, the definitive bidding phase starts and the participants can no longer change their commitment. An interesting contribution of (Aknine et al., 2004) is that the authors formally prove the converge of the protocol.

Finally, several researchers combine the original CNET protocol with other strategies to provide support for dynamism. In (Maturana et al., 1999), a mediator architecture is presented for dynamic scheduling that combines mediation and sub-tasking using CNET to produce a robust schedule. Mediator agents are used to coordinate the resource agents using the CNET protocol. In case of breakdowns, the system is rescheduled by re-running the CNET protocol for the task to find and available slots in the schedule. Other work that combines a mediator architecture and the CNET protocol is described in (Shen and Norrie, 1998) and (Ouelhadj et al., 2003). In this work, different re-schedule techniques for different real-time events are proposed. For example, the re-scheduling technique for rush orders (try to replace a task by another task) differs from a machine breakdowns (find the nearest free slot in the schedule). These mechanisms allow for some flexibility, but task rescheduling is time consuming making the approaches more suitable for systems with limited dynamics.

6. Conclusions

Task assignment in decentralized systems is a complex coordination problem. In this paper, we presented DynCNET and FiTA as two alternative approaches for adaptive task assignment. Our focus was on homogeneous tasks that can be executed by individual agents. In FiTA, agents follow fields in the environment that guide them toward tasks providing an emergent solution for

task assignment. DynCNET is an extension of the Contract Net protocol that allow agents to reconsider provisionally agreed assignments of tasks when circumstances in the environment change. Contrary to other Contract Net extensions that typically provide a specific time window in which tasks can be reassigned, DynCNET allows agents to adapt task assignment from the moment the task enters the system until its execution is started. We have applied DynCNET and FiTA in a simulation of an implemented AGV transportation system. Our experiences yield the following conclusions:

- DynCNET and FiTA have similar performance characteristics and outperform CNET. Contrary to our expectations, DynCNET was not able to outperform FiTA.
- Whereas FiTA is inherently robust to message loss, DynCNET is not and requires substantial additional support to deal with message loss.
- Parameter tuning for DynCNET has similar complexity as for FiTA.
- DynCNET explicitly defines the task assignment process among the agents while in FiTA task assignment is implicitly enclosed in the fields.

The tradeoff between support for robustness and engineering comfort–in particular the fact that DynCNET allows engineers to reason on the assignment of tasks–is an important criteria for selecting a task assignment approach in practice. Egemin, our partner in the AGV transportation project, currently considers to incorporate an adaptive agent-based task assignment approach in there basic architecture for AGV control. The engineers tend to give preference to engineering comfort and therefore prefer DynCNET over FiTA. Still, extensive tests are necessary before an eventual decision can be made.

DynCNET and FiTA 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. The approaches assume continual communication access. We believe that DynCNET is applicable in other domains that share these properties. DynCNET requires that no messages get lost. In domains where this condition can not be met, additional support to deal with message loss must be provided. FiTA requires that the strength of the fields can be expressed proportional to the shortest path distance between tasks and agents rather than to the Euclidean distance. This constraint ensures that agents that use FiTA do not get stuck in local minima. As such, the approach may be less suitable for domains where agents are less restricted in their movements in space. Currently, we only have considered domains with homogeneous participants and tasks with different priorities. Further research is needed to study how DynCNET and FiTA are useful in domains with more complex tasks, for

example an AGV transportation system in which AGVs can carry multiple loads simultaneously.

Acknowledgement

Danny Weyns is supported by the Funding for Scientific Research in Flanders (FWO). Nelis Boucké is supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). We are grateful to Kurt Schelfthout and the employees of Egemin, in particular Jan Wielemans and Tom Lefever for the collaboration in the EMC^2 project. Thanks also to Wannes Schols and Bart Demarsin. Finally, we thank the reviewers for their critical and useful remarks.

References

Aknine, S., S. Pinson, and M. F. Shakun: 2004, 'An Extended Multi-Agent Negotiation Protocol'. *Journal on Autonomous Agents and Multi-Agent Systems* **8**(4).

Arkin, R.: 1998, *Behavior-Based Robotics*. MIT Press.

Bandini, S., S. Manzoni, and C. Simone: 2002, 'Dealing with Space in Multiagent Systems: A Model for Situated Multiagent Systems'. In: *1st International Joint Conference on Autonomous Agents and Multiagent Systems*. ACM Press.

Borenstein, J., B. Everett, and L. Feng: 1996, *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Wellesley.

Boucké, N., D. Weyns, K. Schelfthout, and T. Holvoet: 2006, 'Applying the ATAM to an Architecture for Decentralized Contol of a AGV Transportation System'. In: *2nd International Conference on Quality of Software Architecture*. Springer.

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

Buchman, G., D. Cohen, P. Vernaza, and D. Lee: 2005, 'University of Pennsylvania Robocup 2005 Legged Soccer Team'. http://www.cis.upenn.edu/robocup/UPenn05.pdf.

Bussmann, S. and K. Schild: 2000, 'Self-Organizing Manufacturing Control: An Industrial Application of Agent Technology'. In: *4th International Conference on Multi-Agent Systems*.

Donald, B., K. Lynch, and D. Rus: 2000, *Algorithmic and Computational Robotics: New Directions*. A. K. Peters, Ltd., Wellesley.

Ferber, J. and J. Muller: 1996, 'Influences and Reaction: a Model of Situated Multiagent Systems'. *2nd International Conference on Multi-agent Systems, Japan, AAAI Press*.

FIPA TC Communication: 2002a, 'FIPA Contract Net Interaction Protocol Specification, Doc. SC00029'.

FIPA TC Communication: 2002b, 'FIPA Iterated Contract Net Interaction Protocol Specification, Doc. SC00030'.

Fischer, K., J. P. Muller, M. Pischel, and D. Schier: 1995, 'A Model for Cooperative Transportation Scheduling'. In: *Proceedings of the First International Conference on Multiagent Systems.* Menlo park, California, pp. 109–116, AAAI Press / MIT Press.

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

Helleboogh, A., T. Holvoet, D. Weyns, and Y. Berbers: 2005, 'Extending Time Management Support for Multi-agent Systems'. In: *Multiagent and Multiagent-based Simulation, New York, USA*.

Huget, M.-P., J. Odell, and B. Bauer: 2006, 'The AUML Appraoch'. In: F. Bergenti, M.-P. Gleizes, and F. Zambonelli (eds.): *Methodologies and Software Engineering for Agent Systems, The Agent-Oriented Software Engineering Handbook*. Kluwer.

Knabe, T., M. Schillo, and K. Fischer: 2002, 'Improvements to the FIPA Contract Net Protocol for Performance Increase and Cascading Applications'. In: *Workshop on Multiagent Interoperability*.

Koren, Y. and J. Borenstein: 1991, 'Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation'. In: *Proceedings of the IEEE Conference on Robotics and Automation*.

Maes, P.: 1994, 'Modeling Adaptive Autonomous Agents'. *Artificial Life Journal* **1**(1-2), 135–162.

Mallya, A. and M. Singh: 2005, 'Modeling Exceptions via Commitment Protocols'. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, USA, ACM Press*.

Mamei, M. and F. Zambonelli: 2003, 'Motion Coordination in the Quake 3 Arena Environment: a Field-Based Approach'. *First International Conference on Environments for Multiagent Systems*.

Mamei, M. and F. Zambonelli: 2004, 'Programming pervasive and mobile computing applications with the TOTA middleware'. In: *2nd International Conference on Pervasive Computing and Communications*. IEEE Computer Society, Washington, DC, USA.

Mamei, M., F. Zambonelli, and L. Leonardi: 2004, 'Co-Fields: A Physicall Inspired Approach to Distributed Motion Coordination'. *IEEE Pervasive Computing*.

Mataric, M.: 1994, 'Leaning to Behave Socially'. In: *From Animals to Animats, 3th International Conference on Simulation of Adaptive Behavior*. MIT Press.

Maturana, F., W. Shen, and D. Norrie: 1999, 'MetaMorph: An Adaptive Agent-Based Architecture for Intelligent Manufacturing'. *Journal of Production Research* **37**(10).

Ong, L.: 2003, 'An Investigation of an Agent-Based Scheduling in Decentralised Manufacturing Control'. *Ph.D Disseration, University of Cambridge*.

Ouelhadj, D., P. Cowling, and S. Petrovic: 2003, 'Intelligent Systems Design and Applications'. Springer-Verlag.

Paoli, F. D. and G. Vizzari: 2002, 'Context dependent management of field diffusion: an experimental framework'. *Workshop Dagli Oggetti agli Agenti, Villasimius, Italy*.

Reynolds, C.: 1996, 'Flocks, Herds and Schools: A Distributed Behavior Model'. *Computer Graphics* **21**(4), 25–34.

Sandholm, T. and V. Lesser: 2002, 'Levelled-commitment contracting: A backtracking instrument for multiagent systems'. *AI Magazine* **23**(3), 89100.

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

Schelfthout, K. and T. Holvoet: 2005, 'Views: Customizable abstractions for context-aware applications in MANETs'. *Software Engineering for Large-Scale Multi-Agent Systems, St. Louis, USA*.

Schelfthout, K., D. Weyns, and T. Holvoet: 2006, 'Middleware that Enables Protocol-Based Coordination Applied in AGV Control'. *IEEE Distributed Systems Online* **7**(8).

Schillo, M., C. Kray, and K. Fischer: 2002, 'The eager bidder problem: a fundamental problem of DAI and selected solutions'. In: *AAMAS '02: Proceedings of the first international*

joint conference on Autonomous agents and multiagent systems. New York, NY, USA, pp. 599–606, ACM.

Shehory, O., S. Kraus, and O. Yadgar: 1999, 'Emergent cooperative goal-satisfaction in large scale automated-agent systems'. *Artificial Intelligence* **110**(8), 1–55.

Shen, W. and D. Norrie: 1998, 'An Agent-Based Approach for Dynamic Manufacturing Scheduling'. In: *Agent-Based Manufacturing, Minneapolis*.

Smith, R.: 1980, 'The contract net protocol: High level communication and control in a distributed problem solver'. In: *IEEE Transactions on Computers, C-29(12)*.

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

UNiMod: 2006. *http://unimod.sourgeforce.net*.

Weisstein, E.: 2006, 'Confidence Interval, Probability and Statistics, MathWorld'.

Weyns, D., N. Boucké, and T. Holvoet: 2006, 'Gradient Field Based Transport Assignment in AGV Systems'. In: *5th International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS, Hakodate, Japan*.

Weyns, D., N. Boucké, T. Holvoet, and B. Demarsin: 2007, *DynCNET: A Protocol for Flexible Transport Assignment in AGV Transportation Systems*. Katholieke Universiteit Leuven, Belgium: Technical Report CW 478. http://www.cs.kuleuven.ac.be/publicaties/rapporten/CW/2007/.

Weyns, D. and T. Holvoet: 2006, 'From Reactive Robotics to Situated Multiagent Systems: A Historical Perspective on the Role of Environment in Multiagent Systems'. In: *Engineering Societies in the Agents World VI, 6th International Workshop, ESAW, Kusadasi, Turkey*. Springer-Verlag.

Weyns, D. and T. Holvoet: 2007, 'Architectural design of a situated multiagent system for controlling automatic guided vehicles'. *International Journal of Agent-Oriented Software Engineering* **1**(4).

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

Zeghal, K. and J. Ferber: 1993, 'CRAASH: A Coordinated Collision Avoidance System'. In: *European Simulation Conference*. Lyon, France.