The Packet-World: A Test Bed for Investigating Situated Multi-Agent Systems

Danny Weyns, Alexander Helleboogh and Tom Holvoet

Abstract. Research on situated multi-agent systems investigates how to model a distributed application as a set of cooperating autonomous entities (agents) which are situated in an environment. Many fundamental issues remain unrevealed in this research area. A profound understanding of these issues, however, is necessary before situated multi-agent systems can be applied to industry-strength applications. We use the abstract application called the Packet-World quite extensively as a test bed for investigating, experimenting and evaluating fundamental concepts and mechanisms. Examples are active perception, decision making of situated agents, synchronization of simultaneous actions and indirect coordination. The Packet-World has direct connections with real-world applications, such as the decentralized control of a warehouse transportation system through unmanned vehicles. In this article, we describe the Packet-World and we give an overview of our research for which we have used the Packet-World as a test bed.

Keywords. test bed, situated multi-agent system, environment, perception, action selection, protocol-based communication, synchronization, simultaneous actions, stigmergy, automated warehouse transportation system, automatic guided vehicle.

1. Introduction

In the last 15 years, multi-agent systems (MASs) have been put forward as a paradigm to tackle the increasing complexity of distributed applications. An agent as an autonomous entity, capable of interacting with other agents in order to satisfy its design objectives, is a natural concept to manage complexity in a decentralized manner. Agents encapsulate their own behavior and are able to adapt to changes in

This work was completed with the support of the Concerted Research Action on Agents for Coordination and Control project and the Egemin Modular Controls Concept project.

their environment. Although MASs have already been applied with success in practice, many issues remain open for further research. One thing that researchers and application developers have made clear is that MASs are very complex systems. Test beds are important for investigating, experimenting and evaluating fundamental concepts and mechanisms of MASs. In the end, however, the benefits of obtained research results should/must be demonstrated in real-world applications.

Our research focusses on situated MASs, i.e. MASs in which agents are explicitly placed in an environment. In this article, we present the Packet-World. The Packet-World is a test bed for investigating situated MASs, developed in Java. We show how the Packet-World has inspired our research during the last three years, and we explain how we have used the test bed for the evaluation of our work. Examples concepts and mechanisms we discuss are active perception, decision making of situated agents, synchronization of simultaneous actions and indirect coordination. In a current research project with an industrial partner, we investigate how the paradigm of situated MASs can be applied to the control of automated transportation systems that use automatic guided vehicles (AGVs) to transport loads through a warehouse. In this project, we can validate many of our research results obtained in the Packet-World in a complex real-world case. The AGV case shows that the Packet-World can serve as an abstract application that represents a family of real-world applications for which situated MASs may be a suitable solution.

This article is structured as follows. Section 2 briefly introduces situated MASs. In Section 3 we present the Packet-World. Section 4 discusses the underlying reference architecture of the Packet-World. Next, we elaborate on the main architectural concerns in the Packet-World in Section 5. Section 6 zooms in on advanced forms of collaboration. In Section 7, we illustrate the analogy between the Packet-World an an automated warehouse transportation system. Finally, we draw conclusions.

2. Situated Multi-Agent Systems

A situated MAS is a computing system composed of a (distributed) environment populated with a set of localized agents that cooperate to solve a complex problem in a decentralized way. Situated agents are entities that encapsulate their own behavior and maintain their own state. They have local access to the environment, i.e. each agent is placed in a local context which it can perceive and in which it can act and interact with other agents. A situated agent does not use long-term planning to decide what action sequence should be executed, but instead it selects actions on the basis of its position, the state of the world it perceives and limited internal state. In other words, situated agents act in the present, “here” and “now”. Intelligence in a situated MAS originates from the interactions between the agents, rather then from their individual capabilities.

The approach of situated MASs has a long history. R. Brooks [6][7] identified the key ideas of *situatedness*, *embodiment* and *emergence of intelligence*. L. Steels [25] and J. L. Deneubourg [11] introduced the basic mechanisms for agents to coordinate through the environment: *gradient fields* and *marks*. P. Maes [19] adopted the early robot-oriented principles of reactivity in a broader context of software MASs. K. Rosenblatt and D. Payton [23], T. Tyrrell [26], T. Balch and R. Arkin [3] and many others explored the underlying fundamentals of reactivity and situatedness and developed new architectures for situated agents that enable better adaptive behavior and support more flexible design than the early-days hard-wired stimulus-response structures. A. Drogoul [12], M. Dorigo [9], V. Parunak [20] and many other researchers drew inspiration from social insects and adopted the principles in situated MASs.

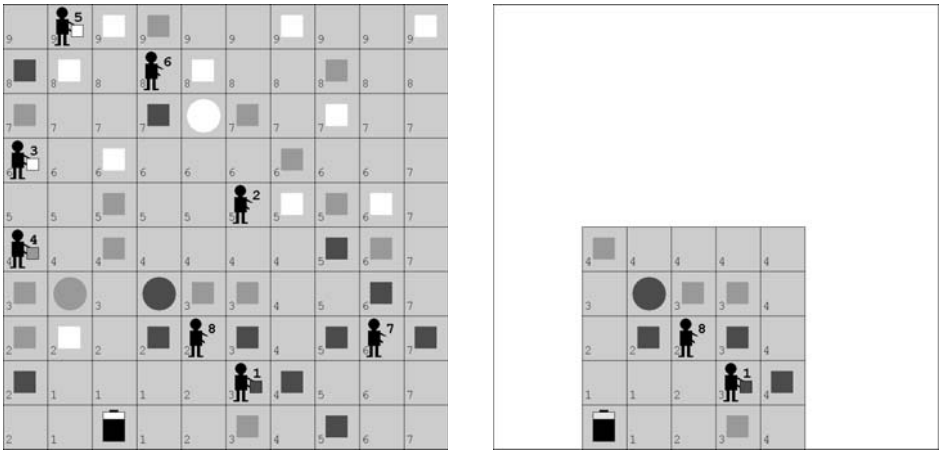
Situated MASs have been applied with success in practical applications over a broad range of domains. Some examples are: manufacturing control [21], supply chains systems [24], network support [5] and peer-to-peer systems [2]. The benefits of situated MAS are well known, the most striking being efficiency, robustness and flexibility. In [35], M. Wooldridge points to a number of limitations of situated MASs. Wooldridge argues that situated agents take into account only local, current information and thus inherently must take a "short-time" view for decision making. However, complex problem domains suitable to apply agent-technology, such as manufacturing control or ad-hoc networks, are by their very nature distributed and highly dynamic. In such domains it is questionable whether it is feasible or even useful for agents to collect global information or to have a "long-term" view on the situation. Another problem raised by Wooldridge is that there is no principled methodology to engineer situated agents, in particular with respect to desired overall behavior of the system. The relationship between the local interactions of agents and the global behavior of the MAS is indeed a complex open problem in need of extensive further research.

3. The Packet-World Test Bed

In [17], M. Huhns and L. Stephens propose a research exercise to tackle a number of open research questions regarding situated MASs. The problem domain of this exercise is composed of a two-dimensional grid consisting of packages and destinations. In this domain, robots must move the packages to the correct destination. The goal of the exercise is to investigate under which conditions the robots will develop social conventions and how the robots can take advantage of information communicated with each other. This exercise was our inspiration for developing the Packet-World.

3.1. Basic Setup of the Packet-World

The basic setup of the Packet-World consists of a number of differently colored packets that are scattered over a rectangular grid. Agents that live in this virtual world have to collect these packets and bring them to the correspondingly colored



(a) A Packet-World of 10x10. (b) Local view of agent 8.

FIGURE 1. Example of the Packet-World.

destination. We call a *job* the task of the agents to deliver all packets in the world. Fig. 1(a) shows an example of a Packet-World of size 10x10 with 8 agents. Colored rectangles symbolize packets that can be manipulated by the agents and circles symbolize destinations.

In the Packet-World, agents can interact with the environment in a number of ways. Agents are allowed to make one step at a time to a free neighboring cell. If an agent is not carrying any packet, it can pick up a packet from one of its neighboring cells. An agent can put down a packet it carries at one of the free neighboring cells, or of course at the destination point of that particular packet. Finally, if there is no sensible action for an agent to perform, it may wait for a while and do nothing. Besides acting in the environment, agents can also send messages to each other. In particular agents can request each other for information about packets or destinations, or ask to set up collaborations. Performing actions requires energy. Therefore agents are equipped with a battery. The energy level of the battery is of vital importance to the agents. The battery can be charged at one of the available battery chargers. Each charger emits a gradient. The gradient values of all battery chargers are combined into a single gradient field. To navigate towards a battery charger, the agents follow the field in the direction of decreasing gradient values. In the example of Fig. 1 there is only one charger, indicated by a battery symbol. The value of the gradient field is indicated by a small number in the bottom left corner of each cell. The intensity of the field increases further away from the charger.

It is important to notice that each agent of the Packet-World has only a limited view on the world. The *view-range* of the world expresses how far, i.e. how

many squares, an agent can perceive its neighborhood. Figure 1(b) illustrates the limited view of agent 8, in this example the view-range is 2.

The goal of the agents is to perform their job efficiently, i.e. with a minimum number of steps, packet manipulations and message exchanges. We monitor the Packet-World via two counters that measure the efficiency of the agents in performing their job. A first counter measures the energy consumed by the agents. Stepping with a packet or without a packet, picking up a packet or putting it down and communicating messages all have an energy cost. As a default, when an agent makes a step without carrying a packet it consumes one unit of energy, stepping with a packet requires two units of energy. The energy required to pick up a packet or to put it down is also one unit. Finally, waiting and doing nothing is free of charge. The second counter measures the number of messages sent. By default, this counter simply increments for each message that is transferred between two agents. The overall performance can thus be calculated as a weighted sum of all energy-consuming activities.

3.2. Objectives of the Packet-World

In the past, several other test beds for MASs have been developed. The most famous is probably the "Tileworld" [22]. The original Tileworld consists of a grid of cells on which an agent has to pick up and move tiles toward holes. The tiles and holes appear and disappear at rates determined by parameters of the simulator. These parameters enable the user to tune the experiments in order to examine particular aspects of interest. The Tileworld has been used for evaluating several kinds of agent architectures, see e.g. [18].

In contrast to the Tileworld, in which only one agent operates, in the Packet-World a collection of agents has to solve the problem. Central to the Packet-World is global problem solving using local interaction between the situated agents. The focus of the Packet-World is on "conceptual exploration" of situated MASs, rather than on "testing". During the last three years, we have applied the Packet-World as a test bed for investigating the following issues in situated MASs: (1) perception of the environment; (2) (simultaneous) actions; (3) direct and indirect communication; (4) timing issues and execution control; (5) different forms of collaborations; and (6) adaptability. In the course of this research, we have introduced several extensions to the Packet-World, such as "heavy packets" that must be manipulated by two agents simultaneously, or pheromones, flags and gradient fields to enable agents to coordinate indirectly, etc.

4. Underlying Reference Architecture of the Packet-World

In this section, we discuss the underlying reference architecture of the Packet-World. First we clarify the main characteristics of the reference architecture. Then we give a graphical overview of the architecture and briefly explain the essential building blocks and their interrelationships.

4.1. Characteristics of the Reference Architecture

The early school of reactive MASs originated from the rejection of classical agency based on symbolic AI. Nowadays, the original opposition tends to evolve towards convergence and integration. A pioneer of this synergetic vision on MASs is J. Ferber [13]. In line with this evolution we developed a new perspective on situated MASs, resulting in a reference architecture with the following main characteristics:

1. The environment is modeled as a first-class entity with its own processes that manage the state of the environment. The environment is observable to the agents and serves as a regulating entity, i.e. it defines the rules for, and enforces the effects of, the agents' (inter)actions.
2. The situated agents as well as other processes can be active in the environment, asynchronously as well as simultaneously.
3. The situated agents are approached as social entities capable to commit to one another in their situated context.
4. The reference architecture is developed according to state-of-the-art software engineering principles and complies with the rules of separation of concerns and reuse-ability.

4.2. Overview of the Reference Architecture

Figure 2 depicts a high-level overview of the reference architecture for situated multi-agent systems [30]. We use a model for action that is based on Ferber's theory of influences and reactions [13]. According to this theory, agents produce influences in the environment and subsequently the environment reacts by combining the influences to deduce a new state of the world from them. The reification of actions as influences enables the environment to combine simultaneously performed activity in the system.

The architecture integrates three primary abstractions: agents, ongoing activities and the environment. First we look at the agent architecture. The $Perception_i$ module maps the local state of the environment onto a percept for the agent. We use a model for active perception that enables an agent to direct its perception at the most relevant aspects in the environment according to its current task. We discuss perception in the Packet-World in Section 5.2. The $KnowledgeIntegration_i$ module uses the most recent percept to update the current knowledge of the agent. The $Decision_i$ module is responsible for action selection. The decision module is set up as a free-flow activity tree. To enhance the social behavior of the agents, we extended free-flow trees with the concepts of a role and a situated commitment. A role maps on a subtree that covers a logical functionality of the agent. A situated commitment enables an agent to bias its action selection towards the actions of the role it plays in the commitment. We elaborate on agent's decision making in the Packet-World in Section 5.3. The $Communication_i$ module takes care of the communicative interactions. The communication module processes incoming messages and produces outgoing messages according to well-defined communication protocols. Agents typically modify their state (current knowledge and/or situated

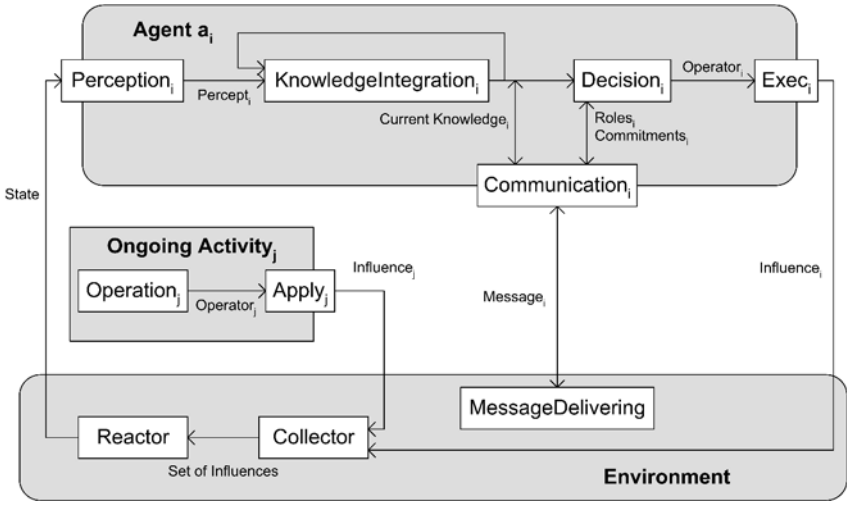


FIGURE 2. Overview of the Reference Architecture.

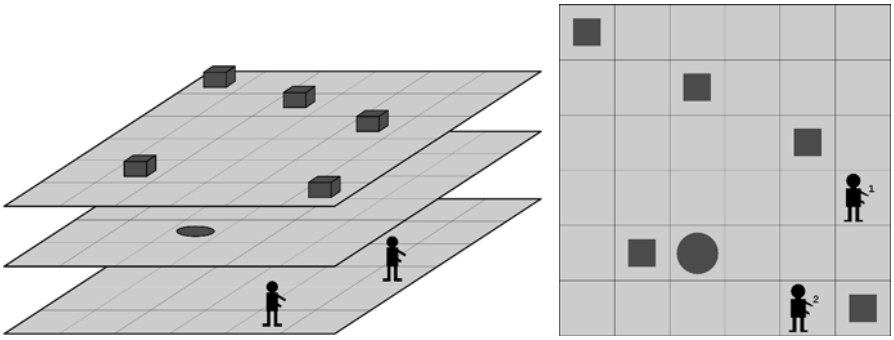
commitments) based on the commutative interactions. We discuss communication in the Packet-World in Section 5.4.

Next to agents, the architecture integrates the concept of an ongoing activity to model other processes that may produce activity in the system. Examples of ongoing activities are moving objects, evaporating pheromones or environmental variables such as temperature. An ongoing activity is defined by an $Operation_j$. Ongoing activities produce influences in the environment depending on the current state of the environment. We discuss several kinds of ongoing activities in the Packet-World in Section 6.2.

The *MessageDelivering* module of the environment handles message transport. The *Collector* module collects the influences of agents and ongoing activities in the MAS and passes sets of simultaneously performed activity to the *Reactor* module. The *Reactor* calculates, according to a set of domain specific laws, the reaction, i.e. state changes in the environment. Dealing with actions in the Packet-World is the subject of Section 6.1. To determine the simultaneity of actions, the architecture supports three forms of synchronization: (1) global synchronization, i.e. all agent act in lock step; (2) regional synchronization, i.e. agents form synchronized groups on the basis of their current locality; and (3) fine-grained synchronization based on logical time. We elaborate on synchronization in the Packet-World in Section 5.5.

5. Architectural Concerns in the Packet-World

In this section, we discuss the architecture of the Packet-World. The architecture of the Packet-World is a concrete instantiation of the reference architecture for



(a) Environment structure decomposed in layers.

(b) Integrated representation in the Packet-World.

FIGURE 3. Layered model of the environment.

situated MAS as presented in the previous section. Subsequently, we zoom in on the following concerns: structure of the environment, perception, agent's decision making, communication, execution control and timing.

5.1. The Structure of the Environment

A first concern we have investigated with the Packet-World is the structure of the environment. The environment of the Packet-World has a grid structure. We have modeled the environment as a collection of grid layers [27], see Fig. 3(a). Each layer hosts a particular kind of item[1]. Examples are an agent-layer, a packet-layer, a pheromone-layer etc. Relations can be defined between items in different layers, e.g. an agent in the agent-layer can hold a packet in the packet-layer. The aggregate of layers populated with items and their interrelationships represent the state of the world as it can be perceived by the agents. Each layer defines a set of constraints on the items for that layer, e.g. two packets can not be located on the same location in the packet-layer. In addition, the combined locations of items in different layers can be constrained. An example of an inter-layer constraint is: an agent can not be located on the same position as a destination (since agents are not allowed to move across a destination).

The layered structure of the environment has several advantages. First of all, it improves extensibility and re-usability. It is relatively simple to add a new layer to the environment when needed, and layers can be reused over different versions of the Packet-World. Second, the layered structure simplifies the generation of agent's perception and the calculation of effects of actions. Since we allow an agent to perceive its environment selectively, only the corresponding layers of the environment have to be taken into account to generate a perception. The laws that apply when agents act in the environment are made explicit by means of the

[1]Item is the generic type for objects and agents in the Packet-World.

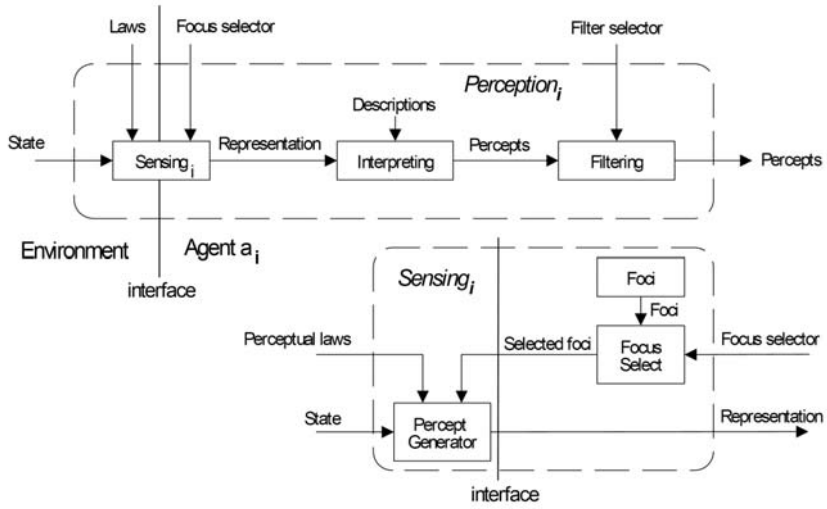


FIGURE 4. *Perception*$_i$ module at the top, with a detail of *Sensing*$_i$ module at the bottom.

constraints defined for each particular layer, and the constraints defined between layers. We further elaborate on these issues in the next sections.

5.2. Perception

Perception in software MASs is a relatively unexplored research domain. We investigated perception in the Packet-World and developed a model for active perception that enables an agent to direct its perception according to its current tasks. Figure 4 gives an overview of the perception module of an agent [33]. The module for perception is decomposed into three functional modules: sensing, interpreting and filtering.

Sensing$_i$ maps the state of the environment to a representation. The mapping of state to a representation depends on two factors. First the agent can select a set of *foci*. Focus selection enables an agent to direct its perception, it allows the agent to sense the environment for specific types of information. Examples of foci in the Packet-World are $see(a_i, range)$ and $smell(a_i, range)$. The focus $see(a_i, range)$ expresses that agent a_i intends to perceive all "visible" items within a distance defined by *range* measured from its current position. Examples of visible items in the Packet-World are packets, destinations, battery chargers or other agents. With the $smell(a_i, range)$ focus, the agent expresses its intention to smell its neighborhood, typically to sense pheromones (see Section 6.2). Second, the representation of the environment state is composed according to a set of *perceptual laws*. A perceptual law constrains the composition of a representation according to the requirements

of the modeled domain. As such, perceptual laws are an instrument for the designer to model domain-specific constraints on perception. Contrary to physical sensing that incorporates such constraints naturally, in virtual environments we have to model the constraints explicitly. An example of a perceptual law in the Packet-World is a law that specifies which items agents are able to "see" in their neighborhood. If e.g. an agents a_i requests a perception with a focus $see(a_i, 5)$ and the general view-range is four, the law will cut off the perceived area to a range of four cells relative to the agents current location. But if the agent requests a perception with focus $see(a_i, 2)$, the returned representation will contain all visible items within a range of two cells. The generation of a representation is a responsibility of the environment and is handled by the *PerceptGenerator*, see Fig. 4.

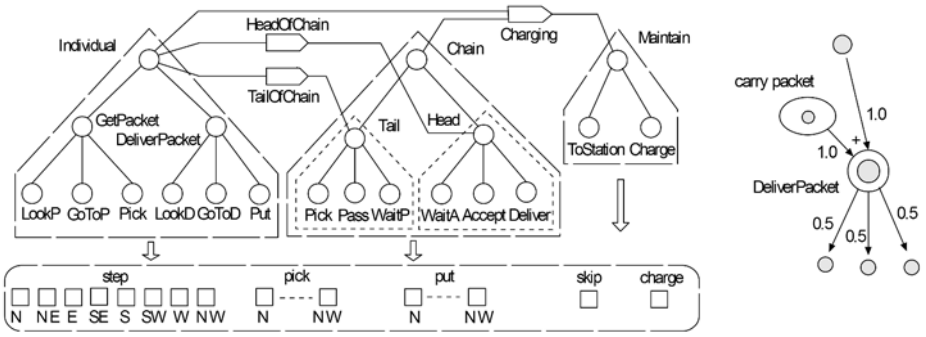
The second functionality of perception is *Interpreting*. Interpreting maps a representation to a percept. To interpret a representation, agents use *descriptions*. Descriptions are blueprints that enable agents to extract percepts from representations. Percepts describe the sensed environment in the form of expressions that can be understood by the internal machinery of the agent. An example is a representation that contains a number of packets in a certain area. The agent that interprets this representation may choose a description to interpret the distinguished packets or another description to interpret the group of packets as a cluster.

The third and final functionality of active perception is *Filtering*. By selecting a set of *filters*, an agent is able to select those items of a percept that match specific selection criteria. Each filter imposes conditions on the elements of a percept. These conditions determine whether the elements of a percept can pass the filter or not. For example, an agent that has selected a focus to visually perceive its environment and that is currently interested in the agents within a range of two cells can select an appropriate filter $select(\text{"}Agent\text{"}, 2)$.

5.3. Agent's Decision Making

Another architectural concern we have investigated with the Packet-World is decision making. The model for agent's action selection in the Packet-World is based on free-flow trees [23]. Since existing free-flow trees are designed from the viewpoint of individual agents, they lack support for *explicit* social behavior. To enable explicit social behavior for situated agents, we have extended free-flow trees with the concepts of a *role* and a *situated commitment* [31]. Fig. 5(a) depicts a simplified partial action selection model for an agent in the Packet-World.

The tree is composed of *nodes* which receive information from internal and external stimuli in the form of *activity*. The activity values of internal stimuli are directly derived from the agent's current knowledge, the values of external stimuli are indirectly derived from perception, via the *KnowledgeIntegration* module. The nodes feed their activity down through the hierarchy. When the activity flow arrives at the *action nodes*, i.e. the leaf nodes of the tree, a winner-takes-it-all process decides which action is selected. Fig. 5(b) depicts in detail how the *DeliverPacket* node collects its activity from a parent node and the "carry packet"" stimulus, and feeds the combined activity down in the hierarchy.



(a) Free-flow tree. (b) Detail of the flow.

FIGURE 5. Action selection for an agent in the Packet-World.

We define a role as a subtree in the hierarchy that covers a logical functionality of the agent. The root node of such a subtree is denoted as the *top node* of the role. A role is named after its top node. A role may consist of a set of sub-roles, and sub-roles of sub-sub-roles, etc. All roles of the agent are constantly active and contribute to the final decision making by feeding subsets of actions with activity. However the contribution of each role depends on the activity it has accumulated from the affecting stimuli of its nodes. In the example, there are three roles demarcated by dotted triangles. In the role *Individual*, the agent searches for packets and brings them to the destination. The role of *Chain* is composed of two sub-roles: *Head* and *Tail* denoting the two roles of agents in a collaboration to pass packets along a chain. Such a collaboration enables these agents to deliver packets more efficiently at the destination. Finally, in the role of *Maintain* the agent recharges its battery.

A situated commitment defines a relationship between one role, i.e. the *goal role*, and a non-empty set of other roles of an agent, i.e. the *source roles*. Each link between a source role and the commitment has a *weight factor* that determines the extent of influence of the associated role on the situated commitment. Situated commitments have a name. Explicitly naming the commitments enables agents to set up mutual commitments in a collaboration. However, a single agent can also commit to itself. The connector *Charging* in Fig.5(a) denotes the situated commitment of an agent to itself to recharge its battery. The connectors $HeadOfChain$ and $TailOfChain$ denote the mutual situated commitments of two agents to collaborate in a chain. The situated commitment $HeadOfChain$ in the example, connects the single source role *Individual* with the goal role *Head*. *Charging* on the other hand connects two source roles with one goal role.

Besides a name, each situated commitment is characterized by a *relations* set, a *context*, an *activation condition*, a *deactivation condition*, a *status* (activated or deactivated) and an *addition function*. To illustrate these characteristics,

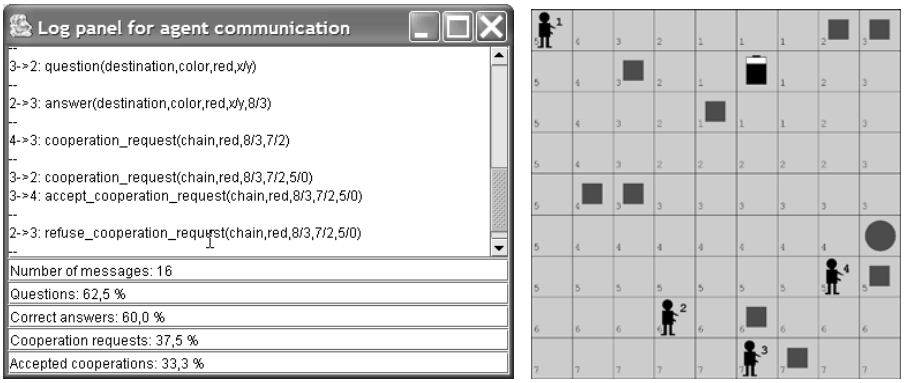
consider agent 6 in Fig.1(a) that commits to be $HeadOfChain$ in a collaboration to pass yellow packets along a chain with agent 5. The relations set contains the identity of the related agent(s) in the situated commitment. The relations set of agent's 6 commitment is the singleton containing agent 5. The context describes contextual properties of the situated commitment. Applied to the example, the context of the situated commitment denotes that yellow packets are passed along the chain. Activation and deactivation conditions are boolean expressions based on internal state, perceived information and/or received data of a message. When the activation condition becomes true, the situated commitment is activated. The situated commitment then injects an additional amount of activity in the goal role defined by the addition function. The weight factors of the links from the source roles determine the fraction of the activity level of the top node of each source role that is taken into account by the addition function. The top node of the goal role combines the additional activity of the situated commitment with the regular activity accumulated from its stimuli. The activation condition for the situated commitment of agent 6 in the example, is the receipt of agent's 5 confirmation to collaborate. As soon as the deactivation condition becomes true, the situated commitment is deactivated. Then the situated commitment no longer influences the activity level of its output node. The deactivation condition of the commitment of agent 6 is a change in the environment that indicates that the collaboration has finished, e.g. agent 5 has left its post to recharge its battery.

In general, one agent can be involved in different situated commitments at the same time. The top node of one role may receive activity from different situated commitments and may pass activity to different other situated commitments. Activity received through different situated commitments is combined with the regular activity received from stimuli into one result.

Summarizing, agents typically agree on mutual situated commitments in a collaboration via direct communication, which is discussed in the next section. Once activated, the situated commitment will affect the selection of actions. The situated commitment induces extra activity in the hierarchy, favoring action selection described by the goal role of the situated commitment. Traditional approaches of commitment oblige agents to communicate each other explicitly when the conditions for a committed cooperation no longer hold. For a situated commitment it is typically the local context in which the involved agents are placed that regulates the duration of the commitment. This approach fits the general principles of situatedness and robustness of situated multi-agent systems.

5.4. Protocol-Based Communication

Communication in multi-agent systems is traditionally based on speech act theory [1]. Speech act theory treats communication as actions and these communicative acts are considered in isolation. In practice, however, speech acts are mostly part of series of logically related communicative acts. We used the Packet-World to study communication in terms of protocols. Communication protocols emphasize the relationship between the exchanged messages in communicative interactions.



(a) Communication log. (b) Packet-World situation.

FIGURE 6. Communication in the Packet-World.

We define a *communication protocol* as a set of *protocol steps* [32]. A protocol step is a tuple *(conditions,effects)*, with *conditions* a set of boolean expressions that determine whether the protocol step is applicable. Conditions are based on received messages, the agent's available roles, and the agent's state (i.e. its current knowledge and the status of its situated commitments). *Effects* are the results of the application of the protocol step, i.e. the composition of a new message and/or the modification of the agent's state. A communicative interaction (conversation) is initiated by the *initial* step of a communication protocol. At each stage in the conversation there is a limited set of possible protocol steps. Terminal states determine the end of a conversation.

As an example, let us look at the communication protocol to set up a chain for passing packets in the Packet-World, see Fig. 6. In the depicted situation, the conditions for agent 4 to set up a chain are fulfilled. Therefore it sends a *cooperation_request* to agent 3, the candidate tail. *cooperation_request* is the initial step of the communication protocol. In the Packet-World agents use a simple FIPA-ACL-like communication language. The basic version of this language allows agents: (1) to request each other for information about packages or destinations, and (2) to set up chains to pass packets to each other. The third line in Fig. 6(a) depicts the cooperation request of agent 4. *chain* refers to the kind of cooperation that is requested, *red* is the color of the packets to be passed, 7/2 are the current x/y-coordinates of agent 4 and 8/3 are the coordinates of the destination for red packets. Agent 3 then investigates the proposal. Since it is able to pass two red packets to agent 4, agent 3 sends an *accept_coopeartion_request* and activates the situated commitment $TailOfChain$, as shown in Fig. 5(a). The fifth line in Fig. 6(a) shows that agent 3 accepts the request for cooperation. After receiving the acceptance, agent 4 activates the situated commitment $HeadOfChain$. In the mean time, agent 3 itself has requested agent 2 to cooperate in the chain, see line

4. But, since agent 2 has no packets to pass, it refuses the cooperation and sends a *refuse_cooperation_request*, see the bottom line in the log panel.

After the mutual commitments between agent 4 and agent 3 are activated, the cooperation is settled and continues until the situated commitments get deactivated. Deactivation may be communicated explicitly by "end of cooperation" messages, but may also be induced by changes in the environment. For example, in case agent 3 has passed all packets and leaves its position to find new work, or in case one of the agents runs out of energy and leaves its position to recharge its battery, the other agent will detect this change and will *terminate* the conversation and that ends the cooperation.

5.5. Execution Control and Timing

A radically different concern investigated in the Packet-World, is execution control and timing for situated MASs. The relative timing and order in which various agents perform activities in the Packet-World, is completely arbitrary due to thread scheduling, message transport delays and variable processor loads in the execution platform. Providing execution control mechanisms is necessary to enforce requirements with respect to relative timing and order, e.g. for enabling repeatable simulation results and reliable testing. Three different approaches of execution control are supported in the Packet-World: global synchronization, regional synchronization and synchronization based on logical time.

5.5.1. Global Synchronization. Global synchronization enforces all agents in the Packet-World to act at one global pace. All agents are synchronized with each other and perform actions simultaneously. In each cycle, each agent performs one action, after which the actions of all agents are processed and the cycle restarts. The main advantage of this approach is its simplicity. However, global synchronization severely limits the agent's autonomy, since agents cannot decide themselves when to perform an action. Moreover, as all agents are forced to act at the pace of the slowest agent in the entire MAS, the efficiency of acting is low. Finally, in a distributed setting this approach scales badly since the centralized synchronizer is a bottleneck and a single point of failure.

5.5.2. Regional Synchronization. Compared to global synchronization, regional synchronization [28, 8] allows synchronization to be more selective, increasing the efficiency of acting for the agents. Regional synchronization is based on the characteristic that agents within a situated MAS typically perceive and act locally. In the Packet-World, a regional synchronization algorithm allows each agent to synchronize with all agents within its perceptual range, and in turn, these agents synchronize with all agents within their perceptual range, and so on. Applied to an example: with a view size of 2 in Fig. 7, we have three regions of synchronized agents, indicated by different colors: region 1 consisting of agent 1, agent 4 and agent 5, region 2 consisting of agent 2, agent 3 and agent 6, and finally agent 7 forms region 3 on its own, since it is currently out of range of all other agents. All

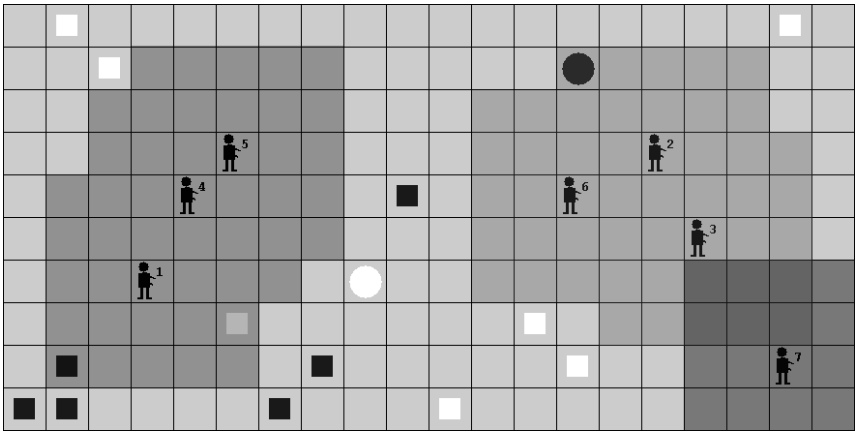


FIGURE 7. Regional synchronization in the Packet-World.

agents within the same region are synchronized with each other and act simultaneously, performing actions at the same pace. Different regions on the other hand are asynchronous: the agents within region 1 act at their own pace, independent of the pace of the agents in region 2 or region 3. Note that the regions are dynamic and have to be kept up-to-date as agents move. For instance, if agent 7 performs a step in the northern direction, it enters the perceptual range of agent 3, and hence regions 2 and 3 merge. The cost of regional synchronization is an overhead of communication to maintain regional groups of synchronized agents. The advantage of regional synchronization is its scalability, as no central synchronizer is employed. However, since the execution of different regions is asynchronous, the relative timing and order of actions performed by agents belonging to different regions is not guaranteed. For a quantitative evaluation of regional synchronization we refer to [28].

5.5.3. Synchronization Based on Logical Time. Global and regional synchronization does not take into account the nature of the actions agents perform, the handling of (simultaneous) actions does not reflect the characteristics of the actions in the real world. For example, suppose that in the real-world a step represents travelling a distance of 10 meters at a speed of 1 meter per second, and picking up a packet takes 5 second. In this case, it takes an agent 2 times longer to perform a step than to pick up a packet. Synchronization based on logical time [16] allows the developer to customize the timing for each action and for each agent, such that the characteristics of the real world are reflected in the model.

In the Packet-World, synchronization based on logical time is supported by means of (1) *semantic duration models* which allow the developer to describe the desired timing characteristics for the Packet-World, and (2) a mechanism that integrates all synchronization functionality *transparently* into the MAS. Consequently,

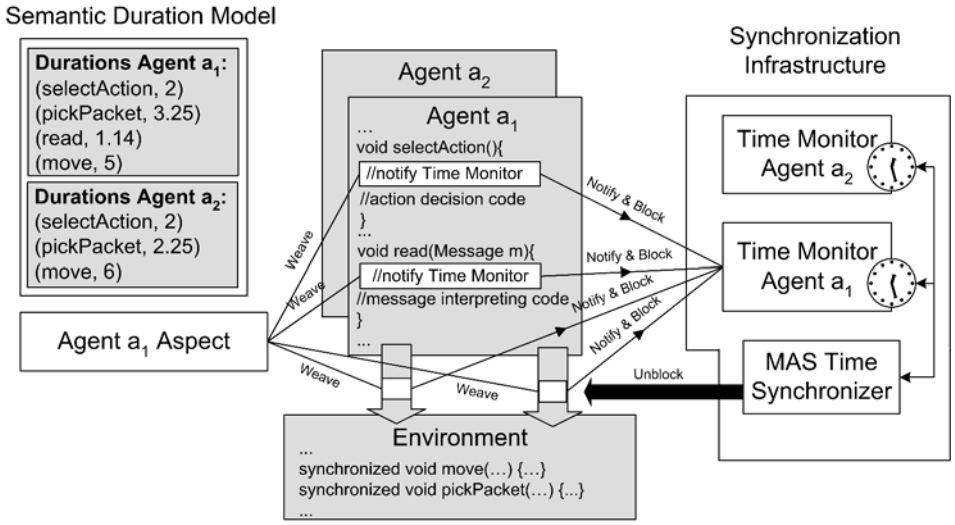


FIGURE 8. Synchronization based on logical time. The white parts of the infrastructure are hidden from the developer.

the timing of the Packet-World can be changed without requiring changes in the agents. Moreover, the complexity of all underlying synchronization infrastructure can be hidden from the developer.

We explain the working of a prototype [15] that was developed and that uses AspectJ to integrate this approach of synchronization in the Packet-World, see Fig. 8. First, the developer specifies a particular *Semantic Duration Model* for each agent within the MAS. Semantic duration models enable the developer to express a duration for each of the activities[2] the agent can perform. The duration of an activity of an agent is the period of logical time it takes until the effects of that activity are noticeable. For each agent a_i, the semantic duration model is described in terms of a list of (c_j^i, r_j^i)-tuples, with c_j^i mapping to a Java method that the agent executes to perform a particular activity with semantic meaning, and r_j^i a constant denoting the logical duration of that activity. For example, in Fig. 8, the model specifies that performing a move action in the Packet-World takes a logical time of 5 units for agent a_1, compared to 6 units for agent a_2.

Based on a semantic duration model of all agents within the MAS, an *Aspect* and a *Time Monitor* are generated for each agent (see Fig. 8). The *Time Monitor* of agent a_i contains a logical clock for that agent. At the start of the simulation, the logical clocks of all agents are zero. The goal of the *Aspect* on the other hand is to notify the *Time Monitor* of all activities the corresponding agent executes.

[2]With the term *activities*, we refer to all internal deliberation an agent can perform, as well as all actions on the environment and all perception of the environment, insofar they are considered semantically relevant for the simulation.

Therefore, the *Aspect* weaves code into all methods that are defined as activities c_j^i of the agent. The goal of the inserted code is to block the execution of the agent as soon as it decides to perform an activity c_i^j and to notify the *Time Monitor*, which then advances the agent's logical clock with a value of r_i^j. For example, a simulation starts and agent a_1 performs a move action while agent a_2 picks up a packet. Both actions are intercepted and blocked, causing the logical clock of agent a_1 to be advanced to 5 and that of agent a_2 to 2.25. The notification of the *Time Monitor* and the blocking of an agent's execution by the inserted code is represented graphically for agent a_1 by the arrowed lines in Fig. 8. The *MAS Time Synchronizer* ensures that all activities are not executed out of logical clock order. The *MAS Time Synchronizer* continuously inspects the logical clocks of all agents, and unblocks agents for which it is safe to proceed. In our example, agent a_2 has the smallest logical clock and hence is allowed to proceed. Consequently, agent a_2 is allowed to perform the pick-up packet action that was previously blocked, and a_2 continues by performing a move afterwards. This causes the execution of agent a_2 to be intercepted again, and its logical clock to be advanced to 8.25. The *MAS Time Synchronizer* compares this value with the logical clock of agent a_1, which still has a value of 5. This causes the execution of agent a_1 to be unblocked while agent a_2 remains blocked. Consequently, agent a_1 performs the move, followed by picking up a packet. At this moment, the logical clocks of both agents are 8.25. Both agents are unblocked and hence the pick-up packet action of agent a_1 and the move of agent a_2 are simultaneous actions as they occur at the same moment in logical time. Note that, to allow agents to perform simultaneous actions deliberately, agents must have an activity at their disposal to wait for a particular logical duration without action. This fine-grained level of synchronization comes at the cost of a scalability comparable to that of global synchronization.

6. Advanced Collaborations in The Packet-World

In the previous section, we discussed several forms of simple collaborations between agents in the Packet-World. Examples are the exchange of information about packets and destinations, or the formation of chains to deliver packets more efficiently. In this section, we zoom in on more advanced forms of collaboration in the Packet-World. First we illustrate collaborations based on simultaneous actions, after that we elaborate on different forms of collaborations based on stigmergy.

6.1. Simultaneous Actions in the Packet-World

Introducing the possibility for agents to act simultaneously opens up new perspectives on collaboration. We extended the Packet-World in several ways to enable agents to act simultaneously. One extension is the possibility for two neighboring agents to transfer a packet directly to one another. An example in Fig. 9 is agent 1 that passes the packet it carries to agent 8. Such a transfer only succeeds when the involved agents act together, i.e. agent 1 has to pass the packet *while* agent

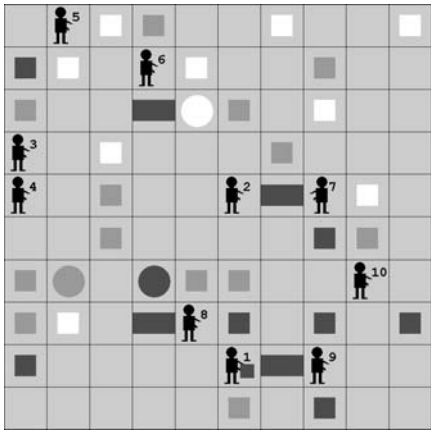


FIGURE 9. Simultaneous Actions in the Packet-World.

8 accepts the packet. Another extension is the introduction of "heavy" packets, denoted in Fig. 9 by the larger rectangles. Contrary to regular packets, to pick up a heavy packet, two agents have to lift up the packet *together*, each of them at a short side of the packet. Agents that carry a heavy packet can only move together in the same direction. As for regular packets, heavy packets can be put down at any free cell or at the delivering point of the packet. However, to put down a heavy packet, both agents have to release the packet simultaneously. An example in Fig. 9 are agents 2 and 7 that make a step with the large packet they carry. Such a step only succeeds when both agents step in the same direction. In the depicted situation, this could correspond to a direction southwest towards the destination of the packet they carry.

6.1.1. Support for Simultaneous Actions. To avoid race conditions, all access to shared state must be controlled. Current mechanisms for access control (locks, semaphores, monitors, etc.) provide support for interleaving concurrent access to shared state. As such, additional support is needed for simultaneous actions whose combined effect differs from performing these actions sequentially. To enable simultaneous actions: (1) agents must be able to act together and (2) the outcome of simultaneous actions must be in accordance with the laws of interaction that apply for the MAS.

Enabling agents to act together requires (1) a model that prescribes the conditions to determine which actions happen simultaneously, i.e. the synchronization model; (2) support to reify actions; and (3) a runtime mechanism that resolves which actions satisfy these conditions. In Section 5.5, we discussed three different models for synchronization we have applied in the Packet-World: global synchronization, regional synchronization and synchronization based on logical time.

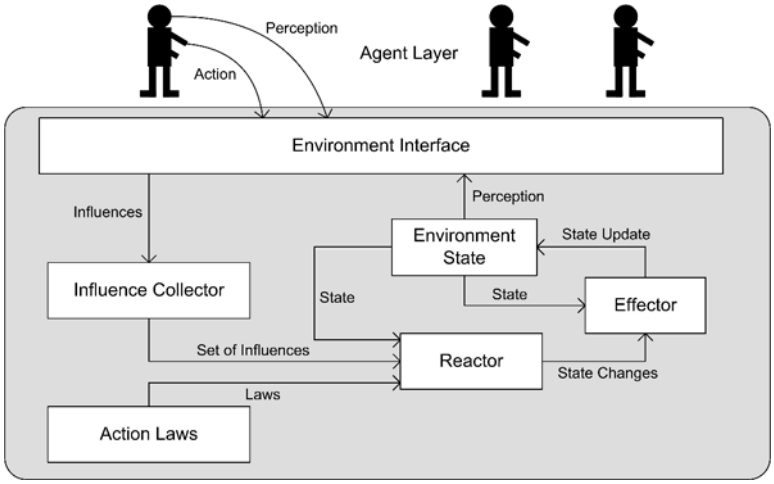


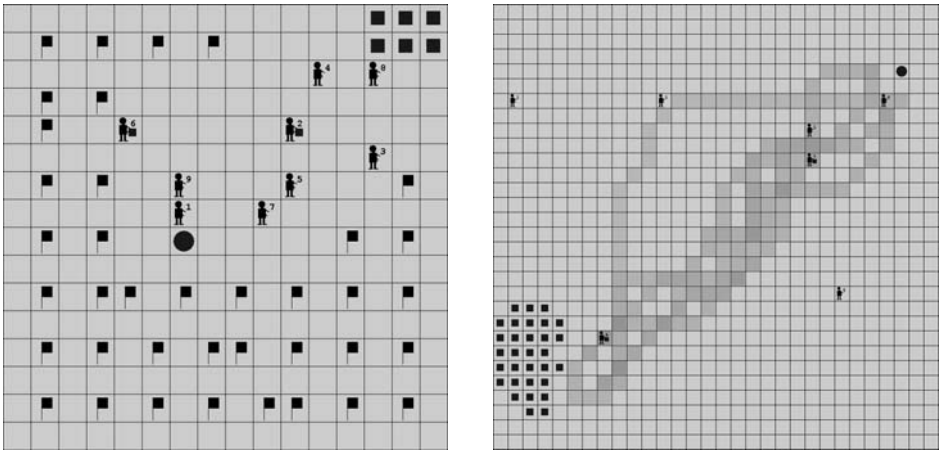
FIGURE 10. Dealing with Simultaneous Actions in the Packet-World.

Ensuring the correct outcome of simultaneous actions requires additional support of the environment. Fig. 10 depicts the model we have used to deal with simultaneous actions in the Packet-World [29]. When an agent invokes an action in the environment, that action is reified as an influence and collected by the *influence collector*. When the collector has received a complete set of influences from a set of simultaneously acting agents[3], it passes the set to the *reactor*. The reactor combines the influences with the valid action laws. The resulting effects are passed to the *effector* that updates the state of the environment. For a detailed discussion on the infrastructure for simultaneous actions we refer to [29][30].

6.2. Stigmergy: Flags, Gradient Fields and Pheromones

The concept of stigmergy was first introduced by P. Grassé [14] to describe the indirect communication among individuals in social insect colonies. In the context of MAS, stigmergy is applied as various forms of indirect communication by means of markers in the environment. Stigmergy enables agents to influence each others behavior indirectly through manipulation of the state of (marks in) the environment, while in learning approaches such as [4] agents modify their own internal state based on feedback received from the environment. To evaluate the applicability of stigmergy as a means for communication between situated agents, we studied the use of three kinds of environmental markers in the Packet-World: flags, gradient fields and synthetic pheromones [10].

[3]For global synchronization, a set contains an influence of each agent in the MAS; for regional synchronization a set contains the influences of all agents of a region; for synchronization based on logical time all the influences with the same logical time belong to a set.



(a) Flag-based stigmergy. (b) Pheromone trails.

FIGURE 11. Stigmergic communication in the Packet-World.

6.2.1. Flags. A first and most simple form of indirect communication are flags which the agent can place in the environment. In the Packet-World, the use of flags was studied as a means to solve the "sparse world" problem. The sparse world problem arises when only a couple of packets are left. The behavior of the agents then becomes inefficient. Most agents keep searching aimlessly for packets, wasting their energy. When several agents detect one of the few packets remaining, all of them run towards it, while in the end only one of them is able to pick it up. To cope with the sparse world problem, agents can use flags to mark a part of the world in which no more packets are present. By placing flags, the agents divide the world in two zones: a marked and an unmarked zone. Agents avoid the marked zone, and only consider the unmarked zone for further exploration. The agents' behavior for placing flags had to satisfy two requirements. First, the destinations must at all times be part of the unmarked zone. Otherwise, the agents would forever try to search for the unmarked zone, even if all packets were collected. Second, and for the same reason, there can only be one unmarked zone: all cells belonging to the unmarked zone are connected. We tested various behaviors for the agents with different strategies for placing flags. The best results were obtained with a behavior maintaining an unmarked zone which always has a convex shape, see Fig. 11(a). To obtain this solution, agents start placing flags in empty corners. Extra flags are placed between other flags or between (flags and) the borders of the world. The shortest path between the remaining packets on the one hand and the destinations on the other hand always lies entirely within the unmarked zone (because of its convex shape) and hence is never excluded.

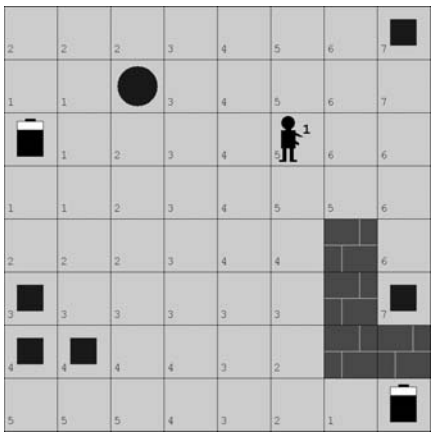


FIGURE 12. Gradient field emitted by battery chargers.

6.2.2. Synthetic Pheromones. A second important form of indirect communication is the use of synthetic pheromones, see Fig. 11(b). Analogous to flags, pheromones are markers that agents can place in the environment. However, pheromones exhibit a number of additional characteristics: evaporation, aggregation and diffusion. Evaporation means that the strength of pheromones diminishes over time. Aggregation on the other hand means that different pheromones at the same cell are combined into a single pheromone with increased strength. Finally, diffusion means that pheromones deposited at a particular cell are spread to neighboring cells over time, making it more likely that agents further away can perceive them. Evaporation and diffusion are examples of ongoing activities, see Section 4. In the Packet-World, the use of pheromones is studied to construct trails between clusters of packets on the one hand and destinations on the other hand. The agents first search for the destination, and then start forming a pheromone trail while searching for packets. In this way, the pheromone trail will lead from the destination towards the packet cluster. Once a packet has been found, the agent can easily deliver the packet by following the pheromone trail back to the destination. In this way, pheromones provide a means for stigmergic coordination between the agents which goes beyond the limitations of the agents' locality in the environment. On the way back from a packet cluster towards the destination, the pheromone trail is reinforced. Shorter trails are reinforced more regularly than longer trails, and hence tend to be more attractive. In the Packet-World, three different types of pheromones are supported: undirected pheromones, unidirectional pheromones and omnidirectional pheromones. *Undirected pheromones* are only characterized by a strength. *Unidirectional pheromones* contain a strength as well as a static direction pointing to one of the neighboring cells. *Omnidirectional pheromones* on the other hand are not limited to a single direction, but contain a probability distribution for all (eight) possible directions.

6.2.3. Gradient Fields. A third form of indirect communication are gradient fields. In the Packet-World, gradient fields are used by agents to retrieve battery chargers. All agents in the Packet-World are equipped with a battery. The battery provides each agent with the energy necessary to perform actions. At regular times, agents have to recharge their battery to prevent it from running out of energy. In order to do so, particular cells on the grid are equipped with a battery charger. Each battery charger emits a gradient that propagates throughout the environment. The effects of all battery chargers are combined into a single gradient field. Gradients fields are constructed only relying on local propagation between neighboring cells, while taking the effect of obstacles into account. In Fig. 12, the value of the gradient field is depicted in the bottom left corner of each cell. The value of the gradient field on each cell represents the minimal distance of that cell to a battery charger. Agents compare the minimal distance to a battery charger with their remaining battery level to estimate the urgency for charging the battery. To navigate towards a battery charger, the agents follow the field in the direction of decreasing gradients.

7. From the Packet-World to an Automated Warehouse Transportation System

The results of our study of the Packet-World as a test bed for situated MAS are currently being validated in a research project in cooperation with Egemin[4], an industrial expert in automating warehouse transportation systems using automatic guided vehicles (AGVs) [34]. An AGV is an unmanned, computer-controlled transportation vehicle that uses a battery as energy source. AGVs have to perform transportation tasks. Such a transportation task consists of picking up a load at a particular location in the warehouse and bringing it to a particular destination. To move from one location to another, AGVs use a complex network of predefined road segments and crossroads. The problem context is highly dynamic. First, the environment continuously changes, as road segments can get congested or blocked because of fallen products or broken down AGVs. Second, the task stream continuously changes as transportation tasks are created on demand. Third, AGVs can become temporarily unavailable for reasons of maintenance or battery charging. The responsibilities of the AGV-system thus are: (1) allocating transportation tasks to AGVs; (2) completing those tasks; (3) avoiding collisions and deadlock; and (4) charging the batteries of AGVs on time.

Today, the design of automated warehouse transportation systems is based on a centralized control system for the AGVs, using one planner which controls all AGVs. The central system gathers all relevant information and controls the actions of each AGV. The goal of the project is to investigate the feasibility of decentralize system control using a situated MAS, aiming to improve flexibility.

There is a clear connection between the Packet-World and an industrial automated warehouse transportation system. First, AGVs as well as agents in the

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

Packet-World are situated in an explicit environment. Second, the AGVs as well as the Packet-World agents have to perform transportation tasks in that environment. Third, the automated warehouse transportation system as well as the Packet-World are decentralized systems; both AGVs and Packet-World agents only have a limited, local view on the environment. Fourth, both AGVs and Packet-World agents use a battery as an energy source that has to be recharged at regular times. Fifth, both applications have a constrained topology. Whereas a grid is employed in the Packet-World, the automated warehouse transportation system has a graph-like topology consisting of nodes (crossroads) and edges (road segments). The main difference between both systems is that AGVs have to deal with the ongoing problem associated with a continuous stream of transportation tasks, whereas in the Packet-World, the stream of transportation tasks is finite and arises as agents detect packets and destinations.

8. Conclusions

In this article, we presented the Packet-World. We illustrated the use of the Packet-World as a test bed in our research to explore and evaluate a broad range of fundamental concepts and mechanisms for situated MASs. We elaborated on the structure of the environment, agents' perception, flexible action selection, protocol-based communication, execution control and timing, simultaneous actions and several forms of stigmergy.

The Packet-World can be considered as an abstract application for a family of complex distributed applications. We illustrated the direct connections of the Packet-World with an industrial automated warehouse transportation system. Currently, our research results obtained from the Packet-World are applied in this real-world application.

A Java implementation of the Packet-World is available[5] under GNU General Public License (GPL).

Acknowledgment

We would like to thank the members of the AgentWise task force at DistriNet labs, K.U.Leuven for their contribution to the work presented in this article. Many thanks also to Eva Weyns for the graphical design of the Packet-World.

References

[1] J. L. Austin, *How To Do Things With Words.* Oxford University Press, UK (1962).

[2] O. Babaoglu, H. Meling and H. Montresoret, *Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems.* 22th International Conference on Distributed Computing Systems, Vienna, Austria (2002).

[5] https://sourceforge.net/projects/packet-world

[3] T. Balch and R.C Arkin, *Communication in Reactive Multiagent Robotic Systems.* Autonomous Robots 1(1) (1994), 27–52.

[4] H.R. Berenji and D. Vengerov, *Cooperation and Coordination Between Fuzzy Reinforcement Learning Agents.* 8th IEEE Conference on Fuzzy Systems, Korea (1999).

[5] E. Bonabeau, F. Hnaux, S. Gurin, D. Snyers, P. Kuntz and G. Theraulaz, *Routing in Telecommunications Networks with Ant-Like Agents.* IATA (1998), 60–71.

[6] R.A. Brooks, *Intelligence Without Representation.* Workshop in Foundations of Artificial Intelligence, Dedham, MA (1987).

[7] R.A. Brooks, *Intelligence Without Reason.* MIT AI Lab Memo No. 1293 (1991).

[8] L. Claesen, *Regional Synchronization in the Packet-World.* Master thesis Katholieke Universiteit Leuven (2004), available in English.

[9] M. Dorigo and L.M. Gambardella, *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.* IEEE Transactions on Evolutionary Computation 1(1) (1997), 53-66.

[10] J. De Meulenaere, *Stigmergy Applied in the Packet-World.* Master thesis Katholieke Universiteit Leuven (2003), only available in Dutch.

[11] J.L. Deneubourg, A. Aron, S. Goss, J.M. Pasteels and G. Duerinck, *Random Behavior, Amplification Processes and Number of Participants: How they Contribute to the Foraging Properties of Ants.* Physics 22(D) (1986), 176–186.

[12] A. Drogoul and J. Ferber, *Multi-Agent Simulation as a Tool for Modeling Societies* Decentralized A.I. 4, Elsevier (1992).

[13] J. Ferber, *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, Great Britain (1999).

[14] P. Grassé, *La theorie de la Stigmergie: Essai d'interpretation du Comportement des Termites Constructeurs.* Insectes Sociaux, Vol. 6 (1959).

[15] A. Helleboogh, T. Holvoet and D. Weyns, *Time Management Support for Simulating Multi-Agent Systems.*, AAMAS Workshop on Multiagent and Multiagent-based Simulation, New York (2004).

[16] A. Helleboogh, T. Holvoet and D. Weyns, *Towards Time Management Adaptability in Multi-Agent Systems.* AISB 2004 Fourth Symposium on Adaptive Agents and Multi-Agent Systems, (2005).

[17] M. Huhns and L. Stephens, *Multiagent Systems and Societies of Agents.* Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence, MIT Press (2000).

[18] D. Kinny, *Measuring the Effectiveness of Situated Agents.* Technical Report 11, Australian AI Institute, Carlton, Australia (1990).

[19] P. Maes, *Modeling Adaptive Autonomous Agents.* Artificial Life Journal 1(1-2), MIT Press, Cambridge, MA (1994), 135–162.

[20] V. Parunak, *Go to the Ant: Engineering Principles from Natural Agent Systems.* Annals of Operations Research 75 (1997), 69–101.

[21] V. Parunak, A.D. Baker and S.J. Clark, *The AARIA Agent Architecture: From Manufacturing Requirements to Agent-Based System Design.* Workshop on Agent-Based Manufacturing, ICAA98, Minneapolis, MN (1998).

[22] M. Pollack and M. Ringuette, *Introducing the Tileworld: experimentally evaluating agent architectures*, Eighth National Conference on Artificial Intelligence, CA (1990).

[23] K. Rosenblatt and D. Payton, *A fine grained alternative to the subsumbtion architecture for mobile robot control.* IEEE Joint Conference on Neural Networks, (1989).

[24] J.A. Sauter and V. Parunak, *ANTS in the Supply Chain.* Workshop on Agent based Decision Support for Managing the Internet-Enabled Supply Chain, WA (1999).

[25] L. Steels, *Cooperation between distributed agents through self-organization.* First European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Elsevier Science Publishers, Holland (1990), 175–196.

[26] T. Tyrrell, *Computational Mechanisms for Action Selection.* PhD dissertation, University of Edinburgh (1993).

[27] B. Vandeweerdt, *A Model for the Environment in Reactive Multi-Agent Systems.* Master thesis Katholieke Universiteit Leuven (2003), only available in Dutch.

[28] D. Weyns and T. Holvoet, *Regional Synchronization for Simultaneous Actions in Situated Multi-Agent Systems.* 3rd International/Central and Eastern European Conference on Multi-Agent Systems, Czech Republic, LNCS 2691 Springer (2003).

[29] D. Weyns and T. Holvoet, *A Model for Simultaneous Actions in Situated Multi-Agent Systems.* 1st International German Conference on Multi-Agent System Technologies, Germany, LNCS 2831 (2003).

[30] D. Weyns and T. Holvoet, *A Formal Model for Situated Multi-agent Systems.* Fundamenta Informaticae 63(2-3) (2004), 125-158.

[31] D. Weyns, E. Steegmans and T. Holvoet, *Combining Adaptive Behavior and Role Modeling with Statecharts.* 3th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, ICSE, Scotland (2004).

[32] D. Weyns, E. Steegmans and T. Holvoet, *Protocol Based Communication for Situated Multi-Agent Systems.* 3th International Joint Conference on Autonomous Agents and Multi-Agent Systems, New York (2004), 118–127.

[33] D. Weyns, E. Steegmans and T. Holvoet, *Towards Active Percption in Situated Multi-Agent Systems.* Journal on Applied Artificial Intelligence 18(8-9) (2004), 867–883.

[34] D. Weyns, K. Schelfthout and T. Holvoet, *Decentralized Control of E'GV Transportation Systems.* 4th Joint Conference on Autonomous Agents and Multi-Agent Systems, Industry Track, Utrecht, (2005).

[35] M. Wooldridge, *An Introduction to MultiAgent Systems.* John Wiley and Sons, England (2002).

Information about the Software

The software of the Packet-World is available on the Internet as full fledged software (under GNU General Public License), version no.: 1.0. The Internet address is: https://sourceforge.net/projects/packet-world/
The Packet-World is a 100% Java test bed for investigating situated multiagent systems. The tool allows users to experiment with various aspects of multiagent systems, incl. agent architectures, stigmergy, communication, etc. For questions about the software, please email : danny.weyns@cs.kuleuven.ac.be

Danny Weyns
Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A, 3001 Leuven
Belgium
e-mail: danny.weyns@cs.kuleuven.ac.be

Alexander Helleboogh
Katholieke Universiteit Leuven, Belgium
e-mail: alexander.helleboogh@cs.kuleuven.ac.be

Tom Holvoet
Katholieke Universiteit Leuven, Belgium
e-mail: tom.holvoet@cs.kuleuven.ac.be