# From Reactive Robotics to Situated Multiagent Systems

## A Historical Perspective on the Role of Environment in Multiagent Systems

Danny Weyns and Tom Holvoet

AgentWise, DistriNet, Katholieke Universiteit Leuven,
Celestijnenlaan 200 A, B-3001 Leuven, Belgium
{danny.weyns, tom.holvoet}@cs.kuleuven.be

**Abstract.** Historically, the idea of situated multiagent systems—in which the environment gets a prominent role—originates from the domain of reactive robotics. In this paper, we give a historical perspective of research on agency that devotes pertinent attention to the environment, and show how the role of the environment evolved along with subsequent evolutions of agent systems. Today, it is quite obvious that the environment offers opportunities and challenges for all types of agency. We discuss recent research in this area, which advocates that the environment is not only an essential part of every multiagent system, but also provides an exploitable design abstraction to build multiagent systems. The notion of environment exceeds specific types of agency, and as such offers opportunities for synergetic research in the interest of multiagent systems in general.

## 1 Introduction

Recently, the environment became subject of active research in multiagent system [1, 2, 3, 4]. Research on environments, however, is not new. In situated multiagent systems the environment has always been a central part of the system. Historically, the idea of situated multiagent systems originates from the domain of reactive robotics. Throughout the different stages in the evolution, from single robotic systems to situated multiagent systems, the role of the environment evolved along with subsequent evolutions of agent systems. Whereas the environment was initially considered as "the external world" in which agents were situated, gradually researchers became aware that the environment provides a medium that could be exploited for building multiagent systems. Today, it is quite obvious that the environment offers opportunities and challenges for all types of agency.

This paper provides a background on the role of the environment in multiagent systems, aiming to help researchers to improve their understanding of the notion of environment in multiagent systems. We give a historical overview of research on agency that devotes pertinent attention to the environment. We show how the role of the environment evolved along with subsequent evolutions of agent systems, and we discuss recent developments in research on environments. The notion of environment exceeds specific types of agency, and as such offers opportunities for synergetic research in the interest of multiagent systems in general.

This paper is structured as follows. In Sect. 2, we give an overview of single agent systems that originate from the principles of reactivity. Section 3 discusses the evolution of multiagent systems, starting from collective reactive behavior to today's situated

O. Dikenelli, M.-P. Gleizes, and A. Ricci (Eds.): ESAW 2005, LNAI 3963, pp. 63–88, 2006.

multiagent systems. In Sect. 4, we discuss recent developments in research on environments and we point to a number of challenging domains for future research.

## 2 Single Agent Systems

Around 1985, several researchers pointed to fundamental problems with deliberative approaches to build agent systems [5, 6, 7]. Reasoning on internal symbolic models and action planning turned out to be insufficient for agents that have to operate in a dynamic and unpredictable environment. These researchers proposed radical new architectures for building agents. Whereas deliberative approaches emphasize explicit knowledge and rational choice, the emphasis of these new architectures was on direct coupling of perception to action, modularization of behavior, and dynamic interaction with the environment. Initially, the focus of this research was on single agent systems. In this section, we give an overview of the subsequent evolutions of single agent architectures and we discuss the role of the environment in this evolution.

### 2.1 Reactive Robotics

In the mid 1980s, researchers were faced with the problem of how to build autonomous robots that are able to generate robust behavior in the face of uncertain sensors, an unpredicted environment, and a changing world [8]. Attempts to build such robots with traditional techniques from artificial intelligence showed deficiencies such as brittleness, inflexibility, and no real-time reaction [9]. Besides, these systems suffered from several theoretical problems, such as the frame problem and the problem of non-monotonic reasoning within realistic time constraints [10]. This brought a number of researches to the conclusion that reasoning on symbolic internal models, and planning the sequence of actions to achieve the goals is unfeasible for agents with many—often conflicting—goals that have to operate in complex, dynamic environments. This conclusion led to the development of a radically new approach to build autonomous agents. The key characteristics of this approach are described by Brooks in [8]:

- *Situatedness.* The robots are situated in the world, they do not deal with abstract descriptions, but with the here and now of the world directly influencing the behavior of the system.
- *Embodiment.* The robots have bodies and experience the world directly, their actions are part of a dynamic with the world.
- *Intelligence.* Robots are observed to be intelligent. The source of intelligence is not limited to the agents internal system, it also comes from physical coupling of the robot with the world.
- *Emergence.* The intelligence of the system emerges from the system's interactions with the world and from indirect interactions between its components.

Architectures for these robots emphasize a direct coupling of perception to action and the dynamic interaction with the environment. The environment is not only taken into account dynamically, but its characteristics are exploited to serve the functioning of the system. The internal machinery of the robots typically consists of combinatorial circuits completed with a timing circuitry. Each circuit represents a simple behavior of the

agent. These circuits are hard-wired or pre-compiled from specifications. The resulting structure allows robots to *react* in real-time to the changing conditions of the world in which they are embedded. Representative examples of approaches for reactive agents are Pengi [6] and Situated Automata [7]. In Pengi, the penguin's situated actions are coded in the form of simple rules. The expressions of the rules use so called indexical-functional representations of the environment. Pengi does not associate symbols with individual objects in the world, but uses expressions that describe causal relationships between the agent and indexically or functionally entities in the world. An example of a situated action is "if there is an ice-cube-besides-me then push ice-cube-besides-me". In Situated Automata, an agent is specified declaratively in the Gapps language [11]. From this specification a runtime program is generated, which satisfies the declarative specification. This program achieves real-time performance, it acts reactively without doing any symbol manipulation.

As an illustration of reactive robots, we discuss the Subsumption Architecture developed by Brooks [5]. The Subsumption Architecture is organized as a series of parallel working layers, each layer is responsible for a specific behavior of the agent. The priority of layers—behaviors—increases from bottom to top. Higher layers are able to inhibit lower layers, giving priority to more important behavior. Fig. 1 depicts an example of a Subsumption Architecture for a simple robot that has to collect packets and deliver them at a destination. On its way, the robot must avoid obstacles in the environment.

A layer in the architecture directly connects perception to action by means of a finite state machine augmented with timing elements. Each layer collects its own sensor data that is written in registers. The arrival of specific data, or the expiration of a timer, can trigger a change of state in the interior finite state machine and possibly produce output commands to actuators. Inhibition and suppression mechanisms resolve conflicts between actuator commands from different layers. In the original version of the Subsumption Architecture, finite state machines could not share any state, each layer encapsulated its registers and clock. Later this restriction was relaxed, allowing clusters of finite state machines to share state and clocks. The Subsumption Architecture has successfully been used in many practical robots.

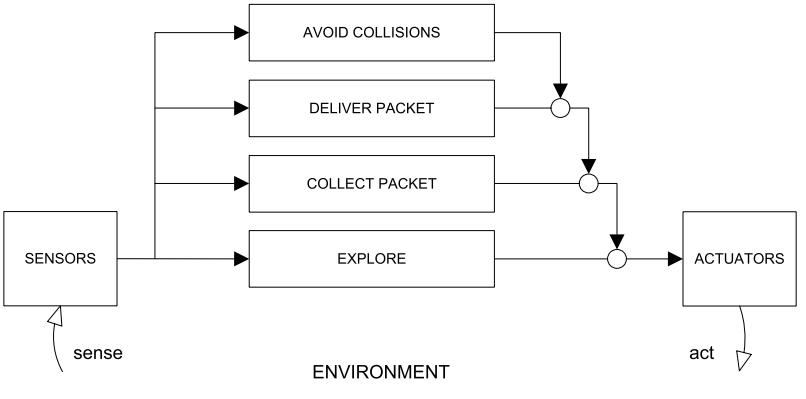


**Fig. 1.** Subsumption Architecture for a Simple Robot

### 2.2 Behavior-Based Agents

In the early 1990s, researchers raised important limitations of the initial reactive approaches. In [9], Maes points to a number of problems with the wired or pre-compiled action selection structures of reactive architectures. Although these approaches demonstrate very good performance, they are typically very specific solutions, leaving little room for reuse. For complex agents in complex environments, the architectures are very hard to build. Another important shortcoming is the lack of explicit goals and goal-handling. The designer must anticipate what the best action is to take in all occurring situations. However, for complex systems much of the necessary information will only be available at runtime. Goals may vary over time and now goals may come into play.

Different approaches that support run-time decision making have been developed, usually referred to as behavior-based or situated agents. Prominent examples are Motor Schemas [12], Distributed Architecture for Mobile Navigation [13] (DAMN), and Free-Flow Architectures [14, 15]. Motor schemas is based on schema theory that explains a robot's motor behavior in terms of the concurrent control of different activities [16]. A schema-based robot consists of a number of parallel executing motor schemas, each schema providing a behavior. Schemas can be added or removed at runtime. Each motor schema has as output an action vector that defines the way the robot should move in response to the perceived stimuli. The sum of output vectors determines the behavior of the robot. In DAMN different behaviors generate outputs as a collection of votes. Behavior arbitration is a winner-take-all strategy in which the largest number of votes for an action is selected for execution. Multiple parallel arbiters for different control functions can be combined, e.g. for speed, turning, etc. A free-flow architecture consists of a hierarchy of nodes which receive information from internal and external stimuli in the form of activity. The nodes feed their activity down through the hierarchy until the activity arrives at the action nodes (i.e. the leaf nodes of the tree) where a winner-take-all process decides which action is selected. A free-flow architectures allows an agent to take into account different preferences simultaneously.

As an illustration of behavior-based agents, we discuss Maes' Agent Network Architecture [9] (ANA). ANA combines the robot-oriented principles of reactivity such as decomposition along tasks, de-emphasizing of internal world models and emergent functionality with goal-handling at runtime, and puts this approach in a broader context of software agent systems. An ANA consists of a network of competence modules. A competence module is a node in the network with its own specific competence. A competence module has a list of preconditions which have to be true before the competence module becomes executable. In addition, each competence module has a level of activity. When the activation level of an executable competence module reaches a certain threshold, it may be selected for execution, resulting in some actions. Fig. 2 shows a simple example of an agent network architecture.

Competence modules are linked through different types of links. Modules use these links to activate and inhibit each other, so that after some time the activation energy accumulates in the modules that represent the best actions to take, given the current situation and goals. The spreading of activation among modules, as well as the input of

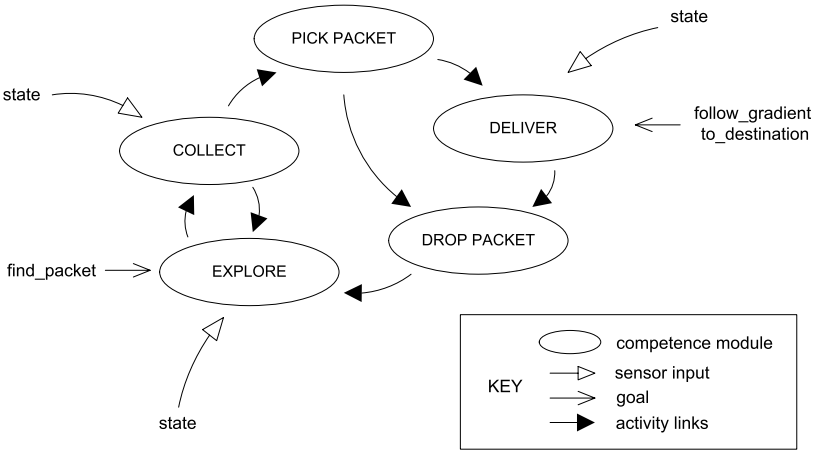


**Fig. 2.** Agent Network Architecture for a Simple Robot [17]

new activation energy into the network is determined by the current observations and the goals of the agent. Note that goals may change at runtime. Through the cumulative effect of forward and backward spreading of activation energy along sequences of competence modules, the network exhibits implicit "planning" capabilities. The continuous re-evaluation of environmental input ensures that the action selection easily adapts with changing situations. However, ANA suffers also from a number of limitations, a detailed discussion is given by Tyrrell in [15]. One problem is the loss of information because the approach assumes binary sensor data. However, many properties of realistic environments are continuous. Tyrrell has demonstrated that ANA suffers from an inherent unbalance of competition among competence modules, resulting in inefficient behavior. Another problem with ANA is the lack of compromise actions, i.e. ANA does not consider preferences of more then one competence module at a time. From our experiences [17], we learned that it is very difficult to design an agent network architecture for a non-trivial agent. ANA offers little support for structuring the behavior of complex agents. Moreover, adding a competence module to an existing network is almost impossible without affecting the existing structure.

### 2.3 Explicit World Models and Hybrid Agent Architectures

The use of explicit world models in reactive-based agent architectures has been subject of debate from the early start of reactive agents. Brooks argued against the need for any kind of world model or cognitive level at all [5]. Other researchers showed how knowledge may be compiled into non-symbolic implementations, see e.g. [18]. In [19], Steels states that "autonomous agents without internal models will always be severely limited". He proposes to use analogical instead of symbolic representations, and demonstrates his approach for a simple robot that has to acquire a map of the environment by wandering around. Another argumentation for the necessity of knowledge representation was given by Arkin in [20]. Arkin states that "despite the assumptions of

early work in reactive control, representational knowledge *is important* for robot navigation", and he demonstrates how a priori and dynamically acquired world knowledge can be exploited to increase flexibility and efficiency of reactive navigation.

Related to the issue of explicit world models is the position of plans. In [21], Agre and Chapman elaborate on the use of plans in agents' decision making. The authors contrast two views on plans: plans as a resource to the agent versus plans for actions. In the view of plans as a resource, agents use plans as a resource among others in continually re-deciding what to do. In the view of plans for action, agents execute plans to achieve goals, i.e. a plan is a prescription of subsequent actions to achieve a goal. The analysis of Agre and Chapman laid the foundation for the work on reactive planning [22, 23].

In [24], Malcolm and Smithers introduced the notion of hybrid architecture. A hybrid architecture combines a deliberative subsystem with a behavior-based subsystem. The deliberative subsystem permits representational knowledge to be used for planning purposes in advance of execution, while the behavior-based subsystem maintains the responsiveness, robustness, and flexibility of purely reactive systems. Over the years, many hybrid behavior-based architectures have been developed. Today, the approach is common in the domain of robotics, for an overview see [25]. A key function in hybrid architectures is the interface between deliberation and reactivity since it links rapid reaction and long-range planning. A common approach to balance reactivity with planning is to introduce an explicit third layer that coordinates among the reactive and deliberative layer. In general however, coordination of deliberation and reactivity is not yet well understood and is subject of active research.

### 2.4 Reflection

Starting from the initial principles of reactivity, a wide range of architectural approaches have been developed. Three classes of approaches are identified:

1. *Reactive robots* emphasize the dynamic interaction with the environment. The internal machinery of the robots directly couples perception to action, enabling real-time reaction.
2. *Behavior-based agents* stress the need for dynamic and flexible action selection, aiming to cope with complex environments. Architectures for behavior-based agents support runtime arbitration among parallel executing behaviors and allow goals to vary dynamically over time.
3. *Hybrid agents* exploit representational knowledge of static aspects of the environment. Architectures for hybrid agents integrate cognition (reasoning over internal representations of the world and planning) with reactivity (real-time reaction to stimuli) aiming to combine the advantages of planning and quick responsiveness.

These approaches share two properties:

1. The focus is on the *architecture* of *single* agents. Architectures differ in the way they solve the problem of *action selection*. Architectures do not support social interaction.
2. The approaches stress the importance of *environmental dynamics*. However, the environment itself is considered as *external* to the system, i.e. the environment is not an explicit part of the models or architectures.

# 3 From Collective Reactive Behavior to Situated Multiagent Systems

Since the early 1990s, researchers which devote pertinent attention to the environment have been investigating systems in which multiple agents work together to realize the system's functionality. In these systems, the agents *exploit* the environment to share information and coordinate their behavior. In this section, we take a look at a number of relevant approaches that have been developed.

## 3.1 Collective Reactive Behavior

In [26], Reynolds demonstrated flocking behavior between a set of agents. The aggregate behavior of the multiagent system emerged from the interaction of multiple agents that each follows a set of simple behavioral rules. Mataric adopted these techniques to real robots [27], showing how a set of robots produced pack behavior. Each robot was 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. In [28], Zeghal demonstrated another form of reactive coordination. Zeghal used 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 repulsing forces resulting from goals and obstacles (including other agents) respectively. An advanced example of behavior-based coordination among unmanned guided vehicles is demonstrated in the DARPA UGV programme.[1] In this case, a DAMN arbiter was used to coordinate the vehicle's behavior given its position in the formation. Although very attractive, several researchers have pointed to the complexity of designing collective reactive behavior, see e.g. [30, 29].

## 3.2 Stigmergic Agent Systems

In [31], Grassé introduced the term *stigmergy* to explain nest construction in termite colonies. The concept indicates that individual entities interact indirectly through a shared environment: one individual modifies the environment and others respond to the modification, and modify it in turn. Deneubourgh [32] and Steels [33] demonstrated how explorer robots can improve the search of target objects by putting marks in the environment. When a robot finds a source of target objects, it puts a trail of marks in the environment from the source of objects toward the robot base, while returning home with an object. This trail allows other exploring robots to find the source of objects efficiently, similar to ants that inform each other about sources of food by depositing pheromone trails in the environment. To ensure that the robots are not mislead when the source becomes exhausted, the marks must be dynamical elements that vanish over time. This mechanism of indirect coordination through the environment combines positive feedback (reinforcement of the trail) with negative feedback (decay of the trail over time).

Stigmergy has been a source of inspiration for many researcher in the multiagent systems. In [34], Parunak describes how principles of different natural agent systems

[1] For a detailed discussion see [29].

(ants, wasps, wolves, etc.) can be applied to build self-organizing artificial agent systems. Example applications of stigmergy are ant colony optimization [35], routing calls through telecommunication networks [36], supply chain systems [37], manufacturing control [38], and peer to peer systems [39].

We illustrate the use of marks in the environment with two prominent examples from literature: first we look at Synthetic Ecosystem developed by Brueckner [38], after that we briefly discuss the Co-Fields approach proposed by Mamei and Zambonelli [40].

**Synthetic Ecosystem.** A synthetic ecosystem enables indirect coordination among software agents in the same way social ants coordinate, the software environment emulates the "services" provided by the real world of ants. The part of the software environment realizing the services is called the pheromone infrastructure. The pheromone infrastructure models a discrete spatial dimension. It comprises a finite set of places and a topological structure linking the places. A link connecting two places has a downstream and an upstream direction. Each agent in a synthetic ecosystem is mapped to a place, i.e. the current location of the agent, which may change over time. The pheromone infrastructure models a finite set of pheromone types. A pheromone type is a specification of a software object comprising a strength-slot (real number) and other data-slots. For each pheromone type, a propagation direction (downstream or upstream) is specified. The pheromone infrastructure handles a finite set of software pheromones for each pheromone type. Every data-slot is assigned a value of a finite domain to form one pheromone (type, direction, propagation, evaporation, etc.). The strength value (i.e. the value in the strength-slot) is interpreted as a specific amount of the pheromone. Different pheromones of a synthetic ecosystem may be stored in each place.

The pheromone infrastructure manipulates the values in the strength-slot of the pheromones at each place in three different ways:

1. External input (aggregation): Based on a request by an agent, the strength of the specified pheromone is changed by the specified value.
2. Internal propagation (propagation/diffusion): When an agent injects pheromone at a place, the input event is immediately propagated to the neighbors of that place in the direction of the pheromone. There the local strength of the pheromone is increased with the arriving pheromone value reduced by the propagation parameter. This process is recursively repeated until the remaining pheromone value crosses a minimal threshold.
3. Without taking changes caused by external input or propagation into account, the strength of each pheromone is constantly reduced in its absolute value (evaporation). The reduction is influenced by the evaporation parameter of the pheromone.

The pheromone infrastructure realizes an application-independent support for synthetic ecosystems designed according to a number of design principles, such as decentralization, locality, parallelism, indirect communication, information sharing, feedback, randomization and forgetting. In [38, 34], Brueckner and Parunak describe a set of engineering principles for designing synthetic ecosystems, including: agents are things, not functions – keep agents small – decentralize control – support agent diversity – enable information sharing – support concurrency.

The principles of synthetic ecosystems and the proposed pheromone infrastructure are applied to a manufacturing control system [38]. V. Parunak and his colleagues have applied digital pheromones in many other practical applications, for an overview we refer to [41].

**Co-fields.** Computational Fields (Co-Fields) is an approach to model and engineer the coordinated movements of a group of agents such as mobile devices (possibly carried by users), mobile robots, or sensors of a dynamic sensor network. In Co-Fields, the movements of the agents are driven by abstract (computational) force fields. By letting agents follow the shape of the fields, global coordination and self-organization can emerge.

The Co-Fields model is essentially based on the following three principles:

1. The environment is represented by fields that can be spread by agents or by the environment itself. These fields convey useful information for the agents to coordinate their behavior.
2. The coordination among agents is essentially realized by letting the agents following the waveform of these fields.
3. Environment dynamics and movements of the agents induce changes in the surface of the fields, realizing a feedback cycle that influences agents' movement. This feedback cycle enables the system (agents and environment) to auto-organize.

A field is defined as a distributed data structure composed of a unique identifier, a value that represents the field magnitude, and a propagation rule. Fields can be generated by the agents or by the environment, and are propagated through the space according to the propagation rule. The propagation rule determines the shape of the field surface. Fields can be static or dynamic. A field is static if its magnitude does not change over time, while a the magnitude of a dynamic field may change. Agents combine the values of the fields they perceive, the resulting new field is called the agents coordination field. Agents follow (deterministically or probabilistically) the shape of their coordination field. Agents can follow the coordination field downhill, uphill, or along one of the equipotential lines of the field. Complex movements are achieved by dynamically reshaping the surface of the field.

In principle, the approach can be generalized toward coordination fields spread in abstract spaces to encode coordination among agents that is related to actions differently from physical movements. In such a case, the agents follow their coordination field, not by moving from one place to another, but by making other kinds of actions.

The Co-Fields model is applied to a number of experimental applications, including a case study in urban traffic management [42] and a video game [43].

### 3.3 Situated Multiagent Systems

Stigmergic agent systems have proven their value in practice, yet, a number of comments are in order:

- Stigmergic agents are considered as "simple" entities. However, there is little or no attention for the architecture of agents.

- Stigmergic agents are not able to set up explicit collaborations to exploit contextual opportunities.
- The environment is considered as *infrastructure for coordination*, typically supporting one particular form of coordination. However, these infrastructures are not concerned with other environmental aspects such as perception, direct communication, or synchronization of actions. As for agents, there is little or no attention for the architecture of the environment.

Motivated by these considerations, researchers have extended the vision of stigmergic agents and developed architectures for a family of agent systems that is generally referred to as situated multiagent systems.

**Multilayered Multi Agent Situated System.** In the Multilayered Multi Agent Situated System [44, 45] (MMASS) agents and the environment are explicitly modelled. MMASS introduces the notion of agent type which defines agent state, perceptual capabilities and a behavior specification. Agent behavior can be specified with a behavior specification language [46] that defines a number of basic primitives, such as emit (starts the diffusion of a field), transport (defines the movement of the agent), or trigger (specifies state change when a particular condition is sensed in the environment). MMASS models the environment as a multi-layered structure, where each layer is represented as a connected graph of sites. Layers may represent abstractions of a physical environment, but can also represent logical aspects, e.g. the organizational structure of a company. Between the layers specific connections (interfaces) can be defined that are used to specify that information generated in one layer, may propagate into other layers. In MMASS, agents can (1) interact through a reaction among adjacent entities, (2) emit fields that are diffused in the environment, and (3) can be perceived by other agents.

**Influence–Reaction Model.** In [47], Ferber and Müller propose a basic architecture for situated multiagent systems. This architecture builds upon earlier work of Genesereth and Nilson [48]. Ferber and Müller distinguish between tropistic and hysteric agents. Tropistic agents are essentially reactive agents without memory, whereas hysteric agents may have complex behaviors that use past experiences for decision making. Central to the model is the way actions are modelled. The action model distinguishes between influences and reactions to influences. Influences are produced by agents and are attempts to modify the course of events in the world. Reactions, which result in state changes, are produced by the environment by combining influences of all agents, given the local state of the environment and the laws of the world. This clear distinction between the products of the agents' behavior and the reaction of the environment provides a way to handle simultaneous activity in the multiagent systems. In [49], Ferber uses the BRIC formalism (Block-like Representation of Interactive Components) to model situated multiagent systems. In BRIC, a multiagent system is modelled as a set of interconnected components that can exchange messages via links. BRIC components encapsulate their own behavior and can be composed hierarchically. An interesting model for action that extends the influence–reaction model with the notion of *activity* as first-class concept is proposed in [50].

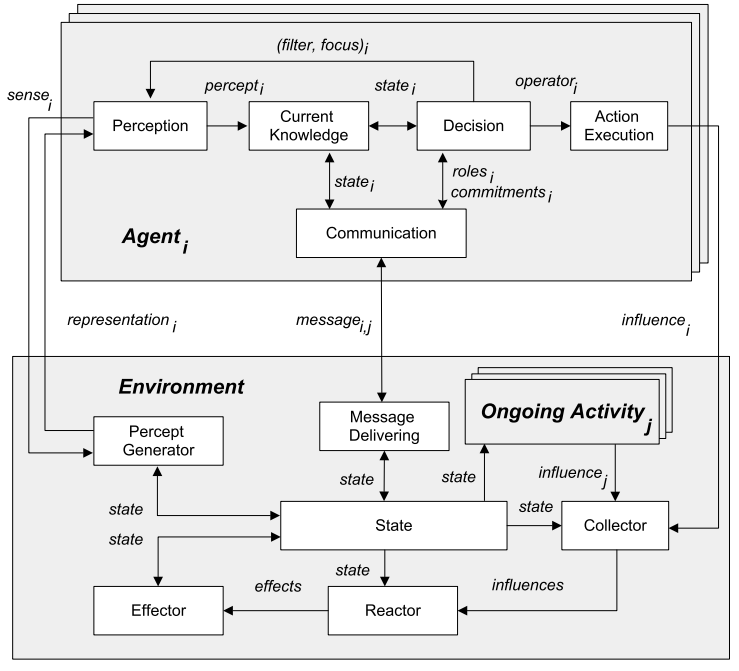


**Fig. 3.** Reference Architecture for Situated Multiagent Systems

**Reference Architecture for Situated Multiagent Systems.** Inspired by the work of Ferber and Müller, in our research we have developed a reference architecture for situated multiagent systems. This reference architecture generalizes and extracts common functions and structures from various applications we have studied and built, including the Packet-World [51], a peer-to-peer file sharing system [52], a number of simple robot applications [53], and an simulator for Automatic Guided Vehicle systems [54]. Fig. 3 shows a high-level module view of the reference architecture. The architecture integrates three primary abstractions: agents, ongoing activities and the environment. We successively look at the architecture of each abstraction.

*Agents.* The agent architecture models different concerns of the agent (perception, decision making and communication) as separate modules. The *Perception* module maps a local representation of the state of the environment to a percept for the agent. We developed a model for *selective perception* that enables an agent to direct its perception at the most relevant aspects in the environment according to its current task [55]. To sense its environment, the agent selects a set of *foci*. Sensing results in a representation of the agent's surrounding that can be interpret by the agent producing a percept. Finally, the percept is filtered by a set of selected *filters*, restricting the perceived data according to specific context relevant selection criteria.

The *CurrentKnowledge* module integrates percepts to update the current knowledge of the agent. The *Decision* module is responsible for action selection [56, 57]. We

developed the decision module as a free-flow architecture. Free-flow architectures allow flexible and adaptive action selection [15]. Since existing free-flow architectures lack explicit support for social behavior, we introduced the concepts of a *role* and a *situated commitment*. A role covers a logical functionality of the agent, while a situated commitment allows an agent to adjust its behavior towards the role in its commitment. An agent can commit to itself, e.g. when it has to fulfill a vital task. However, in a collaboration, agents commit to one another via communication. Roles and situated commitments are building blocks for explicit collective behavior. The operator selected by the decision module is passed to the *ActionExecution* module that invokes an influence in the environment. The action model is based on the influence—reaction model of Ferber and Müller [47].

The *Communication* module takes care of the communicative interactions. We developed a communication module that processes incoming messages and produces outgoing messages according to well-defined communication protocols [58]. The module consists of three functional modules: message decoding, communicating and message encoding. The message decoding module extracts the information from the received messages. The core of the model, the communicating module (1) interprets decoded messages and reacts to them in accordance with the applicable protocol, and (2) initiates or continues conversations when the conditions imposed by the applicable protocol are satisfied. Finally, the message encoding module encodes new messages and passes them to the message transport system of the environment. Communication enables agents to exchange information, and set up collaborations reflected in mutual situated commitments.

*Ongoing Activities.* Next to agents, we introduced the concept of an ongoing activity [59]. An ongoing activity provides an abstraction for an environmental process that happens independent of agents. An ongoing activity is defined by an *Operation* that produces influences in the environment according to the state of the world. Examples of ongoing activities are an evaporating pheromone, a self-managing gradient field, a moving object, or a timer. Ongoing activities are generic building blocks for indirect coordination, and as such it forms a basis for collective behavior.

*Environment.* The environment architecture decomposes the environment into different functional modules (perception, communication, action and interaction). The *PerceptGenerator* module is responsible for perception management [55]. When an agent is interested in perceiving its surroundings, it invokes a *sense* command in the environment. Such a sense command contains a set of foci that expresses the agent's current interests of perception. The PerceptGenerator then composes a representation based on the foci, the current state of the environment and a set of perceptual laws. A perceptual law constrains the composition of a representation according to the requirements of the modelled domain. An example of a perceptual law in the context of a simulation is a law that specifies how an area behind an obstacle is out of scope of a perceiving agent. However, perceptual laws can also serve as an instrument for the designer to introduce "synthetic" constraints on perception. E.g., for reasons of efficiency a designer can introduce default limits for perception in order to restrain the amount of information that has to be processed, or to limit the occupied bandwidth.

The *MessageDelivering* module is responsible for message transfer. When a message arrives, the MessageDelivering module passes the message to the list of addressees indicated in the message. It is possible to provide communication laws that are applied when messages are transferred. An examples is a communication law that specifies the maximal distance that messages can be delivered. Communication laws are interesting for simulation purposes, but can also be a useful instrument for designers, e.g. to regulate the message transfer.

The *Collector—Reactor—Effector* modules take care of action handling [59]. The Collector collects the influences of simultaneously performed activity in the system and passes them to the Reactor. Simultaneity of activity can be based on transactional semantics, or it can be determined by a synchronization mechanism, see e.g. [47, 60]. The Collector passes the influences to the Reactor that calculates, according to a set of domain specific interaction laws, the reaction, i.e. state changes in the environment. An example of an interaction law in the context RoboCup soccer is a law that determines the effects of two football players that kick the ball simultaneously. The Reactor finally passes the effects to the Effector that applies the outcome of the interaction by updating the state of the environment.

It is important to notice that the module view of the architecture as depicted in Fig. 3 abstracts from distribution. For a practical application, the state of the environment, the delivering of messages, ongoing activities, etc., have to be implemented according to the domain at hand, i.e. centralized or distributed. Another important remark is that the presented model also abstracts from physical resources, external to the multiagent system. The state of the environment may represent external resources. Support to keep the state of the representation consistent with external resources is not covered by the presented model.

The reference architecture for situated multiagent systems has been applied in an industrial system for logistics services in warehouses and manufactories. This real-world application uses a situated multiagent system to control an automated guided vehicle (AGV) transportation system [61, 62]. We briefly discuss this application in Sect. 4.2.

### 3.4 Reflection

In multiagent systems, multiple agents work together to realize system functionality. We identified three classes of systems in which the environment has a central role:

- Agents with *collective reactive behavior* follow a set of simple behavioral rules. Each agent is driven by what it perceives in the environment. The aggregate behavior of the multiagent system emerges from the local behavior of agents.
- In *stigmergic agent systems*, the environment serves as a medium for coordination. Stigmergic agents coordinate their behavior through the manipulation of marks in the environment. The environment is an active entity that maintains processes independent of the activity of the agents. Stigmergic coordination combines positive feedback (reinforcement of interesting information) with negative feedback (decay of information over time).
- *Situated multiagent systems* emphasize the importance of architecture for agents and the environment. Basic concerns of agent architecture are perception, communication, and decision making. Advanced types of situated agents support social

behavior enabling them to set up explicit collaborations. Basic concerns of the environment include perception management, message delivering, action handling, and maintenance of processes independent of agents. Laws represent domain specific constraints, but can also be used as a design instrument to impose rules in the multiagent system.

Important characteristics of these multiagent systems are:

- Agents and the environment are explicit parts of the system, each with its specific responsibilities.
- System functionality emerges from the indirect interactions of agents through the environment.

Along the evolution from collective reactive behavior to situated multiagent systems, the role of the environment evolved from (1) the context that drives the agents, to (2) an active coordination medium, to (3) an explicit abstraction with its specific concerns that differ from agent concerns.

Today's situated multiagent systems integrate the architectural perspective of the earlier reactive and behavior-based agent systems with the explicit role of environment of stigmergic agent systems. Moreover, architectures for situated agents extend the initial architectures for single agents by (1) providing support not only for action selection, but for different concerns of agents (perception, communication, etc.), and (2) providing support for explicit social behavior (roles, situated commitments, etc.). Similarly, architectures for the environment extend the role of the environment from a an infrastructure for coordination to a design abstraction that covers specific concerns that differ from agent concerns (perception management, action handling, maintenance of processes, laws, etc.).

Fig. 4 shows a time line with the introduction of subsequent agent systems, together with the main steps in the evolution of the role of the environment in the agent systems.

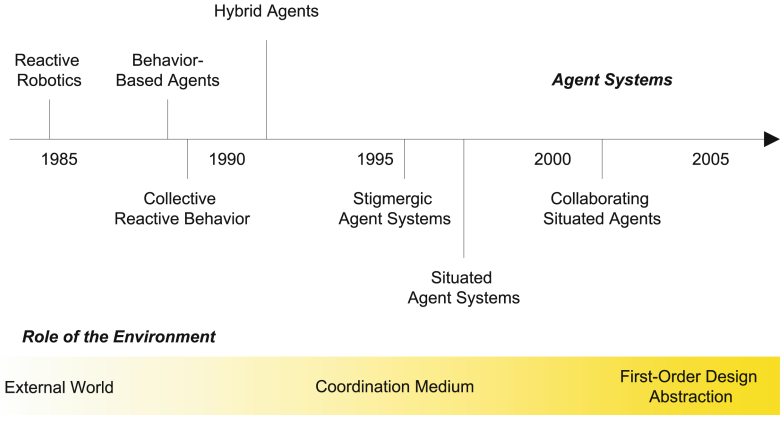


**Fig. 4.** Subsequent Agent Systems and the Evolution of the Role of Environment

# 4 Environment, a First-Order Abstraction in Multiagent Systems

Originating from research on behavior-based agent systems and multiagent systems, and stimulated by a number of recent efforts [1, 2, 3], the environment is now a focus of research in multiagent systems in general. In this section, we first zoom in on the role of the environment in multiagent systems. After that, we discuss a real-world application in which the environment is exploited for coordinating agents behavior. The section concludes with a number of pointers to interesting domains for future research on environments for multiagent systems.

## 4.1 Role of the Environment in Multiagent Systems

Today's research on environments considers a dual role of the environment in multiagent systems. On the one hand, the environment is an *essential* part of every multiagent system that encapsulates parts of a multiagent system that conceptually do not belong to agents, such as infrastructure for communication, the topology of a spatial domain, or laws of an e-institution. Basically, the environment provides the surrounding conditions for agents to exist, it offers an abstraction of the external world to agents in which they can act and interact. This abstraction bridges the conceptual gap between the agent abstraction and low-level issues, such as details of communication, or resources access. On the other hand, the environment provides an *exploitable* design abstraction to build multiagent systems. The environment can serve as a medium for agents to share information and coordinate their behavior.

Distinguishing between agents and the environment supports *separation of concerns* in multiagent systems. A clean separation of agent and environment concerns helps to manage the huge complexity of engineering complex real-world applications. To clarify the role of the environment in multiagent systems, we list a number of important functionalities of the environment:

**The Environment Structures the Multiagent System.** The environment is first of all a shared "space" for the agents, resources and services, which structures the whole system. Resources are objects with a specific state. Services are considered as reactive entities that encapsulate functionality. The agents as well as resources and services are dynamically interrelated to each other. It is the role of the environment to define the rules which these relationships have to comply to. As such the environment acts as a *structuring* entity for the multiagent system. In general, different forms of structuring can be distinguished:

- *Physical structure* refers to spatial structure, topology, and possibly distribution, see e.g. [38, 44].
- *Communication structure* refers to infrastructure for message transfer, infrastructure for stigmergy [38, 40], or support for implicit communication [63, 64].
- *Social structure* refers to the organizational structure of the environment in terms of roles, organizations, and societies, e.g. [65, 66].

Structuring is a fundamental functionality of the environment. Structures of the environment may be imposed by constraints of the domain at hand, or they may be carefully considered design choices.

**The Environment Manages Recourses, Services, and Dynamics.** The environment embeds resources and services. An important functionality of the environment is to enable and control the access to these resources and services, hiding the complexity of low-level issues to agents. In general, resources can be read/perceived, written/modified or generated/consumed by agents. Services on the other hand provide functionality to the agents on their request. The extent to which agents are able to access a particular resource or service may depend on several factors such as the nature of the resource or service, the capabilities of the agent, the (current) interrelationships with other resources, services or agents, etc.

The environment also embeds the agents. The environment may provide support for maintaining external state of agents, examples are tags for coordination or reputation mechanisms.

Besides the activity of the agents, the environment can assign particular activities to resources as well. A digital pheromone, for example, is a dynamic structure as it aggregates with additional pheromone that is dropped, it diffuses in space and it evaporates over time. Other examples of environmental activities are a self-managing field in a network, or in the context of simulation a rolling ball that moves on, or the local temperature that evolves over time. Maintaining such dynamics is an important functionality of the environment.

**The Environment is Locally Observable to Agents.** Contrary to agents, the environment must be observable. Agents must be able to inspect the different structures of the environment, as well as resources, services, and possibly external state of other agents. Observation of a structure is typically limited to the current context (spatial context, communication context and social context) in which the agent find itself. In general, agents should be able to inspect the environment according to their current tasks. Examples of selective perception are [55] where "foci" are proposed to enable agents to perceive their environment according to their current tasks, and [67, 68] where "views" are proposed as selector for perception. Perception is constrained not only by agents' capabilities, but also by environmental properties. In [55], the perceptual constraints are made explicit in the form of "perceptual laws".

Related to observability is the semantic description of the domain, which can be defined by an environment ontology, see e.g. [69]. The ontology must cover the different structures of the environment as well as the observable characteristics of resources, services and agents, and possibly the regulating laws. In an open system, it would be useful for agents to be able to understand at run-time a new environment they are discovering. For symbolically-oriented agents, an explicit ontology should be available to the agents to enable them to interpret their environment and reason about it. For non-reasoning agents, the designer/developer applies the ontology to encode the agents' internal structures. As such, these kinds of agents have an implicit ontology that enables them to make decisions.

**The Environment is Locally Accessible to Agents.** Agents must be able to access the different structures of the environment, as well as resources, services, and possibly external state of other agents. As for observability, accessing a structure is limited to

the current context in which the agent find itself. Access to spatial structure refers to support for metrics, mobility, etc. Access to communication infrastructure refers to support for direct communication (message transfer), support for indirect communication (pheromones, etc.), or support for implicit communication (over-hearing, over-sensing, etc.). Access to social structures refers to group membership, etc.

**The Environment Can Defines Rules for the Multiagent System.** The environment can define different types of rules on all entities in the multiagent system. Rules may refer to constraints imposed by the domain at hand (e.g. mobility in a network), or refer to "synthetic laws" imposed by the designer (e.g. limitation of access to neighboring nodes in a network for reasons of performance). Rules may restrict access to specific resources or services to particular types of agents, or determine the outcome of agents' interactions.

Dealing with interactions in multiagent systems in general is a very complex matter. [70] points out the difficulties to control the activities of agents operating in distributed systems and propose coordination policies to deal with control. According to the authors, coordination policies need to be formulated explicitly rather than being implicit in the code of the agents involved and they should be enforced by means of a generic, broad spectrum mechanism. The environment is the natural candidate to embed such control mechanism.

In electronic institutions [71], agents interact through agent group meetings that are called scenes. Interactions in a scene have to follow a well-defined communication protocol. Scenes can be composed in a performative structure. The specification of a performative structure contains a description of how the different roles can legally move from scene to scene. Agents within a performative structure may participate in different scenes at the same time with different roles. Agent actions in the context of an institution may have consequences that either limit or enlarge its subsequent acting possibilities. Such consequences will impose obligations to the agents and affect its possible paths within the performative structure. The environment can define and enforce the rules imposed on the movements and interactions of agents in an electronic institution.

A particular problem is the regulation of simultaneous actions in simulations. To allow multiple agents to act in the environment in parallel, explicit models are needed to deal with actions that range far beyond the scope of state changes based on simple individual manipulation of objects. [47, 59, 50] discusses models for simultaneous actions.

### 4.2 Exploiting the Environment in Practice

In this section, we illustrate how the reference architecture for situated multiagent systems discussed in Sect. 3.3 is applied to an automated transportation system for warehouse logistics. This real-world application is developed in a joint R&D project between the AgentWise research group and Egemin, a manufacturer of automating logistics services in warehouses and manufactories [61, 72].

The automated transportation system uses automatic guided vehicles (AGVs) to transport loads through a warehouse. Typical applications are distributing incoming

goods to various branches, or distributing manufactured products to storage locations. An AGV is provided with a battery as its energy source. AGVs can move through a warehouse, following fixed paths on the factory floor, typically guided by a laser navigation system, or by magnets or cables that are fixed in the floor. The low-level control of the AGVs in terms of sensors and actuators (such as staying on track on a path, turning, and determining the current position, etc.), is handled by the AGV control software. Fig. 5 depicts a high-level model of the situated multiagent system. The situated

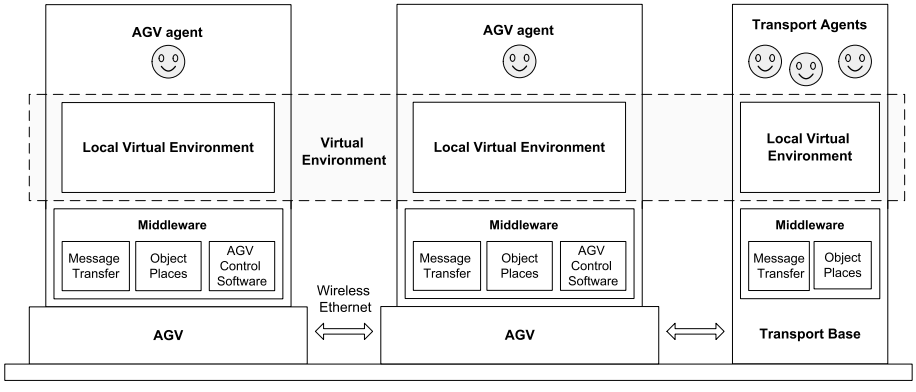


**Fig. 5.** High-level model of the AGV transportation system

multiagent system consists of two kinds of agents, *transport agents* and *AGV agents*. Transport agents are located at *transport bases*. AGV agents are located in AGVs that are situated on the factory floor. The communication infrastructure provides a wireless network that enables mobile AGVs to communicate with each other and with transport agents on transport bases.

A transport agent represents a transport that needs to be handled by an AGV. AGV agents are responsible for executing the assigned transports. AGVs are situated in a physical environment, however, this environment is very constrained: AGVs cannot manipulate the environment, except by picking and dropping loads. This restricts how AGV agents can exploit their environment. Therefore, a virtual environment was introduced for agents to live in. This virtual environment offers a medium that agents can use to exchange information and coordinate their behavior. Besides, the virtual environment serves as a suitable abstraction that shields the AGV agents form low-level issues, such as the physical control of the AGV. The AGV control software that deals with the low-level control of the AGVs is fully reused. As such, the AGV agents control the movement and actions of AGVs on a fairly high level.

In the AGV application, the only physical infrastructure available to the AGVs is a wireless network for communication. In other words, the virtual environment is necessarily distributed over the AGVs and transport bases. In effect, each AGV and each transport base maintains a *local virtual environment*, which is a local manifestation of the virtual environment. Local virtual environments are merged with other local virtual

environments opportunistically, as the need arises. In other words, *the* virtual environment as a software entity does not exist; rather, there are as many local virtual environments as there are AGVs and transport bases. Some of these local virtual environments may have been synchronized recently with each other, while others may not. From the agent perspective, the virtual environment appears as one entity. The synchronization of the state of neighboring local virtual environments is supported by the ObjectPlaces middleware [68].

We now illustrate the use of the virtual environment with a couple of examples.

*Routing.* For routing purposes, the virtual environment has a static map of the paths through the warehouse. This graph-like map corresponds to the layout used by low-level AGV control software. To allow agents to find their way through the warehouse efficiently, the virtual environment provides signs on the map that the agents use to find their way to a given destination. These signs can be compared to traffic signs by the road that provide directions to drivers. At each node in the map, a sign in the virtual environment represents the cost to a given destination for each outgoing segment. The cost of the path is the sum of the static costs of the segments in the path. The cost per segment is based on the average time it takes for an AGV to drive over the segment. The agent perceives the signs in its environment, and uses them to determine which segment it will take next.

*Traffic Information.* Besides the static routing cost associated with each segment, the cost is also dependent on dynamic factors, such as congestion of a segment. To warn other agents that certain paths are blocked or have a long waiting time, agents mark segments with a dynamic cost on a *traffic map* in the virtual environment. Agents mark the traffic map by dropping pheromones on the applicable segments. When AGVs come in each others neighborhood, the information of the traffic maps is exchanged and merged to provide up-to-date information to the AGV agents. Since pheromones evaporate over time, outdated information automatically vanishes over time. AGV agents take the information on the traffic map into account when they decide how to drive through the warehouse.

*Collision Avoidance.* AGV agents avoid collisions by coordinating with other agents through the virtual environment. AGV agents mark the path they are going to drive in their environment using *hulls*. The hull of an AGV is the physical area the AGV occupies. A series of hulls then describes the physical area an AGV occupies along a certain path. If the area is not marked by other hulls (the AGV's own hulls do not intersect with others), the AGV can move along and actually drive over the reserved path. Afterwards, the AGV removes the markings in the virtual environment. [62] discusses collision avoidance through the virtual environment in detail.

In summary, the virtual environment serves as a flexible coordination medium, which hides much of the complexity of the system (distribution, mobility, etc.) from the agents: agents coordinate by putting marks in the environment, and observing marks from other agents. The virtual environment creates opportunities beyond a physical environment that situated AGV agents can exploit.

### 4.3 Challenging for Future Research on Environments

Many issues are open for future research on environments in multiagent systems. [73] gives an extensive overview of challenges in the domain. One particular challenge we stress here is environment engineering. Environment engineering poses challenges a three levels: (1) Architectural design, (2) Detailed design, and (3) Implementation. Successively, we zoom in on each level.

**Architectural Design.** Starting from system requirements, including functional and quality requirements (robustness, flexibility, openness, etc.) as well as project and business constraints (budgets, schedules, etc.), the first step in environment engineering is defining a suitable *software architecture* [74]. Software architecture urges engineers to think first in abstract terms about the structure of the environment, distilling away low-level design and implementation details. Software elements of the software architecture provide the functionality of the environment, while the required quality requirements are primarily achieved through the structures of the software architecture. Integration with legacy systems and middleware are important issues when designing the software architecture of an environment. An important challenge for research on environments will be the development of reusable architectural approaches for architectural design of environments. Architectural patterns [75] (or architectural styles) are recurring architectural approaches with particular quality attributes that can be reused for building software architectures of environments. A reference architecture combines a set of architectural patterns and can serve as a blueprint for developing software architectures for a family of environments that share a common base of functional and quality attributes. One interesting challenge is to develop support for the architectural design of different environment structures (physical, communication, social; see Sect. 4). Interesting work on architectural design of environments is discussed in [76, 62, 77, 78].

**Detailed Design.** A software architecture constrains the concrete development of an environment, yet, it does not *define* it. Detailed design is concerned with the concrete design of the software architectures of environments. One important challenge here is the development of suitable description languages. Examples of open problems for detailed design of environments are support for indirect interaction or environmental laws. Another interesting area for research are the development of specific design and implementation patterns for environments [79, 80].

**Implementation.** Support for the implementation of environments can come from frameworks, libraries, and development platforms. Existing agent tools can be extended with explicit support for environments, or new tools can be developed that support environments within which different kinds of agents can interact. An important aspect of implementation of environments is the integration with middleware platforms. Middleware hides hardware and platform details, and offers powerful capabilities such as remote method invocation, threading, transaction, etc. Moreover, middleware provides a software platform on which distributed environments can run, hiding complex issues

such as low-level details of communication or mobility. A number of proven middleware infrastructures for multiagent systems are [81, 82, 83, 68, 84].

## 5 Concluding Remarks

In this paper, we discussed the evolution of the role of the environment in multiagent systems from an historical perspective of situated multiagent systems. We have showed how the role of the environment evolved along with subsequent types of agent systems. We identified three phases in the evolution of the role of the environment:

1. Single agent systems emphasize environmental dynamics. The environment is considered as "the external world", which is not an explicit part of models and architectures.
2. In stigmergic agent systems, the environment is considered as coordination infrastructure. Stigmergic agents coordinate their behavior through the manipulation of marks in the environment.
3. In situated multiagent systems, agents and the environment are first-order abstractions, each with its own specific responsibilities. Basic concerns of the environment include perception management, message delivering, action handling, and maintenance of processes independent of agents.

Originating from the area of situated multiagent systems, research on environments today exceeds specific types of agency. Distinguishing between agents and the environment supports *separation of concerns* in multiagent systems. Separating agent and environment concerns helps to manage the huge complexity of engineering complex real-world applications. Today's research on environments considers a dual role of the environment in multiagent systems:

1. The environment is an *essential* part of every multiagent system that provides the surrounding conditions for agents to exist.
2. The environment provides an *exploitable* design abstraction to build multiagent systems.

We illustrated how the environment is exploited in a industrial system for logistic services in warehouses. This practical application shows how a virtual environment creates opportunities for agents to share information and coordinate their behavior an a way that would be impossible in a physical environment.

Environments offers numerous opportunities for future research. Interesting challenges for environment engineering are the development of reusable architectural approaches, including architectural patterns and reference architectures for environments; the development of description languages for environment concerns such as indirect interaction or laws; and the development of frameworks and libraries to support the implementation of environments. Developing such reusable tools for environment engineering is the result of extensive practical experiences with building concrete environments in practical multiagent system applications.

We hope that this paper helps researchers to improve their understanding of the notion of environment in multiagent systems. The notion of environment provides a

challenging area for synergetic research in multiagent systems, the environment offers opportunities for all types of agency, from ant systems to rational agent systems such as BDI agents. Understanding the background of environments is essential to carry on the exploration and exploitation of environments in multiagent systems.

## References


1. Weyns, D., Parunak, V., Michel, F., eds.: Proceedings of the First International Workshop on Environments for Multi-Agent Systems, New York, 2004. Volume 3374 of Lecture Notes in Computer Science., Springer-Verlag (2005)
2. Weyns, D., Parunak, V., Michel, F., eds.: Proceedings of the Second International Workshop on Environments for Multi-Agent Systems, Utrecht, 2005. Volume 3830 of Lecture Notes in Computer Science., Springer-Verlag (to appear)
3. AgentLink III Technical Forum Group on Environments for Multiagent Systems. (http://www.cs.kuleuven.ac.be/∼distrinet/events/e4mas/tfg2005/)
4. Weyns, D., Schumacher, M., Ricci, A., Viroli, M., Holvoet, T.: Environment in Multiagent Systems. Knowledge Engineering Review **20** (2005)
5. Brooks, R.A.: Achieving Artificial Intelligence through Building Robots. AI Memo 899, MIT Lab (1986)
6. Agre, P.E., Chapman, D.: Pengi: An Implementation of a Theory of Activity. In: Proceedings of National Conference on Artificial Intelligence, Seattle, WA. (1987)
7. Rosenschein, S.J., Kaelbling, L.P.: The Synthesis of Digital Machines With Provable Epistemic Properties. In: Proceedings of the First Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, CA. (1986)
8. Brooks, R.A.: Intelligence Without Reason. In: Proceedings of 12th International Joint Conference on Artificial Intelligence, Sydney, Australia (1991)
9. Maes, P.: Situated Agents Can Have Goals. Designing Autonomous Agents, MIT Press (1990)
10. Pylyshyn, Z.: The Robot's Dilemma. The Frame Problem in Artificial Intelligence. Ablex Publishing Corp., Norwood, New Jersey (1987)
11. Kaelbling, L.P., Rosenschein, S.J.: Action and Planning in Embedded Agents. Designing Autonomous Agents, MIT Press (1990)
12. Arkin, R.C.: Motor Schema-Based Mobile Robot Navigation. International Journal of Robotics Research **8** (1989)
13. Rosenblatt, J.: DAMN: A Distributed Architecture for Mobile Navigation. In: Proceedings of the Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents, AAAI Press (1995)
14. Rosenblatt, K., Payton, D.: A Fine Grained Alternative to the Subsumption Architecture for Mobile Robot Control. Proceedings of the International Joint Conference on Neural Networks, IEEE (1989)
15. Tyrrell, T.: Computational Mechanisms for Action Selection. University of Edinburgh (1993)
16. Arbib, M.A.: Schema Theory. Encyclopedia of Artificial Intelligence (1992)
17. Custers, R.: The Agent Network Architecture Extended for Cooperating Robots. Master Thesis, Katholieke Universiteit Leuven, Belgium (2004)
18. Kaelbling, L.P.: Goals as Parallel Program Specifications. In: Proceedings of the Seventh National Conference on Artifical Intelligence, Minneapolis, Minnesota. (1988)
19. Steels, L.: Exploiting Analogicl Representations. Designing Autonomous Agents (1990)
20. Arkin, R.: Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. Designing Autonomous Agents, MIT Press (1990)

21. Agre, P.E., Chapman, D.: What are Plans for? Designing Autonomous Agents, MIT Press (1990)
22. Nilsson, N.J.: Teleo-Reactive Programs for Agent Control. Journal of Artificial Intelligence Research **1** (1994)
23. Bryson, J.J.: Intelligence by Design, Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. PhD Dissertation: MIT (2001)
24. Malcolm, C., Smithers, T.: Symbol Grounding via a Hybrid Architecture in an Autonomous Assembly System. Designing Autonomous Agents, MIT Press (1990)
25. Arkin, R.: Bahavior-Based Robotics. MIT Press (1998)
26. Reynolds, C.: Flocks, Herds and Schools: A Distributed Behavior Model. Computer Graphics **21** (1996)
27. Mataric, M.: Leaning to Behave Socially. In: From Animals to Animats, Proceedings of the 3th International Conference on Simulation of Adaptive Behavior, MIT Press (1994)
28. Zeghal, K., Ferber, J.: CRAASH: A Coordinated Collision Avoidance System. In: Proceedings of European Simulation Conference, Lyon, France. (1993)
29. Arkin, R.: Behavior-Based Robotics. Massachusetts Institute of Technology, MIT Press, Cambridge, MA, USA (1998)
30. Wavish, P.R., Connah, D.M.: Representing Multiagent Worlds in ABLE. Technical Note, TN2964, Philips Research Laboratories (1990)
31. Grassé, P.P.: La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubitermes sp. La theorie de la Stigmergie. Essai d'interpretation du Comportement des Termites Constructeurs. Insectes Sociaux **6** (1959)
32. Deneubourg, J.L., Goss, S.: Collective Patterns and Decision Making. Ecology, Ethology and Evolution **1** (1989)
33. Steels, L.: Cooperation between Distributed Agents through Self-Organization. Decentralized Artificial Intelligence (1989)
34. Parunak, V.: Go to the Ant: Engineering Principles from Natural Agent Systems. Annals of Operations Research **75** (1997)
35. Dorigo, M., Gambardella, L.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation **1** (1997)
36. Bonabeau, E., Hnaux, F., Gurin, S., Snyers, D., Kuntz, P., Theraulaz, G.: Routing in Telecommunications Networks with Ant-Like Agents. IATA (1998)
37. Sauter, J., Parunak, H.: ANTS in the Supply Chain. Agent based Decision Support for Managing the Internet-Enabled Supply Chain, Seattle, WA (1999)
38. Brueckner, S.: Return from the Ant, Synthetic Ecosystems for Manufacturing Control. Ph.D Dissertation, Humboldt University, Berlin, Germany (2000)
39. Babaoglu, O., Meling, H., Montresor, A.: Anthill: A Framework for the Development of Agent-Based Peer-to-Peer systems. In: Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, IEEE Computer Society, Digital Library (2002)
40. Mamei, M., Zambonelli, F.: Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination. IEEE Pervasive Computing **3** (2004)
41. V. Parunak, home page. (http://www.erim.org/ vparunak/)
42. Mamei, M., Zambonelli, F., Leonardi, L.: Distributed Motion Coordination with Co-Fields: A Case Study in Urban Traffic Management. In: 6th IEEE Symposium on Autonomous Decentralized Systems, Pisa, Italy, IEEE Press (2003)
43. Mamei, M., Zambonelli, F.: Motion Coordination in the Quake3 Arena Environment. In: Environments for Multiagent Systems, E4MAS. Volume 3374 of Lecture Notes in Computer Science., Springer (2005)

44. Bandini, S., Manzoni, S., Simone, C.: Dealing with Space in Multiagent Systems: A Model for Situated Multiagent Systems. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press (2002)
45. Bandini, S., Manzoni, S., Vizzari, G.: MultiAgent Approach to Localization Problems: the Case of Multilayered Multi Agent Situated System. Web Intelligence and Agent Systems **2** (2004)
46. Bandini, S., Federici, M.L., Manzoni, S., Vizarri, G.: Towards a Methodology for Situated Cellular Agent Based Crowd Simulations. In: Sixth International Workshop on Engineering Societies in the Agents World, ESAW. (2005)
47. Ferber, J., Muller, J.: Influences and Reaction: a Model of Situated Multiagent Systems. Second International Conference on Multi-agent Systems, Japan, AAAI Press (1996)
48. Genesereth, M.R., Nilsson, N.: Logical Foundations of Artificial Intelligence. Morgan Kaufmanns (1997)
49. Ferber, J.: An Introduction to Distributed Artificial Intelligence. Addison-Wesley (1999)
50. Helleboogh, A., Holvoet, T., Berbers, Y.: Simulating actions in dynamic environments. In: Conceptual Modeling and Simulation Conference, CMS2005, Track on Agent Based Modeling and Simulation in Industry and Environment. (2005)
51. Weyns, D., Helleboogh, A., Holvoet, T.: The Packet-World: A Test Bed for Investigating Situated Multiagent Systems. In: Software Agent-Based Applications, Platforms and Development Kits, Whitestein Series in Software Agent Technology (2005)
52. P2P Simulator. (http://trappie.studentenweb.org/andy/www/site mai/main.php)
53. Helsen, E., Deschacht, K.: The DELTA Framework for Situated Multiagent Systems. Master Thesis, Katholieke Universiteit Leuven, Belgium (2005)
54. AGV Simulator. (http://www.cs.kuleuven.ac.be/∼distrinet/taskforces/agentwise/agvsimulator/)
55. Weyns, D., Steegmans, E., Holvoet, T.: Towards Active Perception in Situated Multi-Agent Systems. Journal on Applied Artificial Intelligence **18** (2004)
56. Weyns, D., Steegmans, E., Holvoet, T.: Integrating Free-Flow Architectures with Role Models Based on Statecharts. In: Environments for Multiagent Systems. Volume 3374 of Lecture Notes in Computer Science., Springer-Verlag (2005)
57. Steegmans, E., Weyns, D., Holvoet, T., Berbers, Y.: A Design Process for Adaptive Behavior of Situated Agents. Agent-Oriented Software Engineering, Lecture Notes in Computer Science **3382** (2005)
58. Weyns, D., Steegmans, E., Holvoet, T.: Protocol Based Communication for Situated Multiagent Systems. 3th Joint Conference on Autonomous Agents and Multi-Agent Systems, New York (2004)
59. Weyns, D., Holvoet, T.: Formal Model for Situated Multi-Agent Systems. Fundamenta Informaticae **63** (2004)
60. Weyns, D., , Holvoet, T.: Regional Synchronization for Situated Multi-agent Systems. In: Third International Central and Eastern European Conference on Multi-Agent Systems, Prague, Czech Republic. Volume 2691 of Lecture Notes in Computer Science., Springer-Verlag (2004)
61. EMC$^2$: Egemin Modular Controls Concept. (http://emc2.egemin.com/)
62. Weyns, D., Schelfthout, K., Holvoet, T.: Exploiting a Virtual Environment in a Real-World Application. Second International Workshop on Environments for Multiagent Systems, Utrecht (2005)
63. Tummolini, L., Castelfranchi, C., Omicini, A., Ricci, A., Viroli:, M.: "Exhibitionists" and "Voyeurs" do it Better: a Shared Environment for Flexible Coordination with Tacit Messages. In: Environments for Multiagent Systems. Volume 3374 of Lecture Notes in Computer Science, Springer-Verlag (2005)

64. Platon, E., Sabouret, N., Honiden, S.: Oversensing with a Softbody in the Environment: Another Dimension of Observation. In: Proceedings of Modeling Others from Observation at International Joint Conference on Artificial Intelligence, Edinburgh, Scotland (2005)
65. Ferber, J., Michel, F., Baez, J.: AGRE: Integrating environments with organizations. In: Environments for Multiagent Systems. Volume 3374 of Lecture Notes in Computer Science, Springer-Verlag (2005)
66. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing Multiagent Systems: The Gaia Methodology. ACM Transactions on Software Engineering and Methodology **12** (2003)
67. Julien, C., Roman, G.C.: Egocentric Context-Aware Programming in Ad-Hoc Mobile Environments. In: Proceedings of the 10th Symposium on Foundations of Software Engineering, Charleston, South Carolina, USA, ACM Press, New York, NY, USA (2002)
68. Schelfthout, K., Holvoet, T.: Views: Customizable Abstractions for Context-Aware Applications in MANETs. Software Engineering for Large-Scale Multi-Agent Systems, St. Louis, USA (2005)
69. Chang, P., Chen, K., Chien, Y., Kao, E., Soo, V.: From Reality to Mind: A Cognitive Middle Layer of Environment Concepts for Believable Agents. In: Environments for Multiagent Systems. Volume 3374 of Lecture Notes in Computer Science., Springer-Verlag (2005)
70. Minsky, N., Ungureanu, V.: Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. ACM Transactions on Software Engineering Methodologies **9** (2000)
71. Noriega, P., Sierra, C.: Electronic Institutions: Future Trends and Challenges. In: Proceedings of the 6th International Workshop on Cooperative Information Agents. Volume 2446 of Lecture Notes in Computer Science., Springer-Verlag, London, UK (2002) 14–17
72. Weyns, D., Schelfthout, K., Holvoet, T., Lefever, T.: Decentralized control of E'GV transportation systems. In: 4th Joint Conference on Autonomous Agents and Multiagent Systems, Industry Track, Utrecht, The Netherlands, ACM Press, New York, NY, USA (2005)
73. Weyns, D., Parunak, V., Michel, F., Holvoet, T., Ferber, J.: Environments for Multiagent Systems, State-of-the-Art and Research Challenges. In: Environments for Multiagent Systems. Volume 3374 of Lecture Notes in Computer Science., Springer-Verlag (2005)
74. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison Wesley Publishing Comp. (2003)
75. Shaw, M., Garlan, D.: Software architecture: perspectives on an emerging discipline. Prentice-Hall (1996)
76. Valckenaers, P., Van Brussel, H.: Holonic Manufacturing Execution Systems. CIRP Annals-Manufacturing Technology **54** (2005) 427–432
77. Viroli, M., A.Omicini, Ricci, A.: Engineering MAS Environment with Artifacts. In Weyns, D., Parunak, V., Michel, F., eds.: 2nd International Workshop Environments for Multi-Agent Systems, AAMAS 2005, Utrecht, The Netherlands (2005)
78. Molesini, A., Omicini, A., Denti, E., Ricci, A.: SODA: A Roadmap to Artifacts. In: Sixth International Workshop on Engineering Societies in the Agents World, ESAW. (2005)
79. Kendall, E., Jiang, C.: Multiagent System Design Based on Object Oriented Patterns. Journal of Object Oriented Programming (1997)
80. Schelfthout, K., Coninx, T., Helleboogh, A., Holvoet, T., Steegmans, E., Weyns, D.: Agent Implementation Patterns. In: OOPSLA Workshop on Agent-oriented Methodologies, Seattle, WA USA. (2002)
81. Murphy, A., Picco, G., Roman, G.: LIME: a Middleware for Physical and Logical Mobility. 21th International Conference on Distributed Computing Systems (2001)

82. Omicini, A., Ossowski, S., Ricci, A.: Coordination infrastructures in the engineering of multiagent systems. In Bergenti, F., Gleizes, M.P., Zambonelli, F., eds.: Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Volume 11 of Multiagent Systems, Artificial Societies, and Simulated Organizations. Kluwer Academic Publishers (2004) 273–296
83. Mamei, M., Zambonelli, F.: Programming pervasive and mobile computing applications with the tota middleware. 2nd IEEE International Conference on Pervasive Computing and Communication (2004)
84. Schelfthout, K., Weyns, D., Holvoet, T.: Middleware for Protocol-based Coordination in Dynamic Networks. In: Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing, Grenoble, France, ACM Press (2005)