2. Multi-Agent Systems

2.1 Introduction

The interest for multi-agent systems (MASs) has grown increasingly in the last years. These systems are beginning to be used in a great variety of applications such as air traffic control, process control, manufacturing, electronic commerce, patient monitoring, or games. This appeal is due to the fact that multi-agent systems present very attractive means of more naturally understanding, designing and implementing several classes of complex distributed and concurrent software. This growing attention to multi-agent technology has been even more accentuated with the increase of internet computing, which integrates an underlying infrastructure presenting a space organization in which autonomous agents roam and interact with one another.

Nevertheless, as a consequence of their popularity, the terms "*autonomous agents*" and "*multi-agent systems*" are often misused. It is indeed not rare to find software described in these terms, although it is not necessarily correlated with multi-agent technology. Therefore, we have to clarify what autonomous agents and multi-agent systems really are, how they can be modeled, designed and implemented.

This chapter is organized as follows. In sections 2.2 and 2.3, we present what are autonomous agents and multi-agent systems. On this basis, we discuss in section 2.4 MAS modeling by proposing an explicit integration of the interaction setup of a MAS and by sustaining a distinction between subjective and objective coordination. Then, section 2.5 exposes our target class of systems. Finally section 2.6 discusses how practical MASs can be built and concludes by maintaining the use of coordination models and languages for the design and the implementation of MASs.

2.2 What Is an Autonomous Agent?

The debate on what constitutes an autonomous agent is still under way. As a large range of agent types exist, it is obvious that a lot of descriptions have been proposed, without, however, reaching a commonly accepted definition. This section nevertheless sketches the fundamental characteristics of

M. Schumacher: Objective Coordination in MAS Engineering, LNAI 2039, pp. 9–32, 2001.

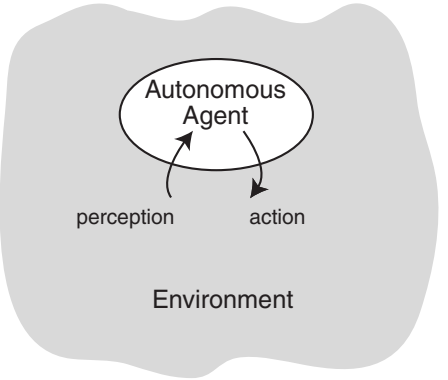

Fig. 2.1. An autonomous agent and its environment.

what most research understands with an autonomous agent by presenting the definition we adhere to and by discussing a wide-spread definition.

2.2.1 Definitions

Wishing to avoid the formulation of a new definition and being conscious of the difficulty of the task due to the wide variety of existing agent paradigms (see for instance [Mae95]. [HR95]. [BT96]. [JSW98]), this section proposes to adopt a proposal of Franklin and Graesser [FG96].

After having analyzed several agent proposals, Franklin and Graesser give a definition, which is similar to that of Beer [Bee95]. They explain the essence of an autonomous agent as follows:

> *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*

This definition is further explained:

> *Each agent is situated in, and is part of some environment. Each senses its environment and acts autonomously upon it. No other entity is required to feed its input, or to interpret and use its output. Each acts in pursuit of it's own agenda. (...) Each acts so that its current actions may effect its later sensing, that is its actions effect its environment. Finally, each acts continually over some period of time.*

This definition shows that the notion of *situatedness* within an environment is an essential concept: the agent can sense its surrounding (receive sensory input) and act upon it so as to change it. Furthermore, the response

of the agent to the environment is done in a timely fashion. Figure 2.1 gives a schematic representation of those ideas.

In the literature, it is not rare to find descriptions of what an autonomous agent is, as proposed by Wooldridge and Jennings in [WJ95]. They define an autonomous agent as:

> *a hardware or (more usually) software-based computer system that enjoys the following properties:*
> – *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;*
> – *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;*
> – *reactivity: agents perceive their environment (...), and respond in timely fashion to changes that occur in it;*
> – *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.*

This definition is not as general as the precedent, because it requires agents to interact using an agent communication language (ACL) [LFP99]. But there exist several types of autonomous agents that do not need such ACLs and only interact through influences in the environment (see section 2.5). Nevertheless, this definition is considered by Wooldridge and Jennings as a *weak* notion of agency, in contrast with a *strong* notion of agency that uses mentalistic notions applied to humans (see section 2.2.3) and that represents one of several existing generic agent architectures. Before presenting these architectures in section 2.2.3, we discuss more in detail two characteristics of autonomous agents, namely autonomy and embodiment.

2.2.2 Autonomy and Embodiment

The definitions discussed in section 2.2.1 clearly refer to a basic notion of autonomy: an agent is considered autonomous if it acts for its proper agenda by choosing, by itself, its actions. But, intuitively, we see that there may exist different degrees of autonomy, including stronger ones. In fact Ziemke distinguishes two families of autonomy [Zie97]: *operational* and *behavioral* autonomy. Operational autonomy is the capacity to operate *without human intervention* [Pfe96]. This is the simplest and basic meaning. The behavioral autonomy, however, has a stronger definition: it can be considered as having its *own capacity to form and adapt its principles of behaviour* [Ste95]. Note that this second autonomy is not a necessary characteristics for the agent definition adopted.

But autonomy also means that each agent possesses its proper temporality in the sense that it runs concurrently to other agents without being synchro-

nized to them. This requirement should especially be taken in consideration when implementing agent-based systems.

Strongly coupled to the notion of autonomy is the fact that an autonomous agent is an embodied system, i.e. the fact that an autonomous agent has a "body" that delimits and isolates the agent from its environment. Different embodiments yield to different kinds of agents [Zie97]:

- *Physically embodied agents* subsist in a physical environment: their interaction with the environment goes through sensory perception and motor control. They are typically robots, or even living agents.
- Several agents use a *simulated embodiment*: simulated physical agents are roaming and acting in a simulated world, thus physical embodied agents and a realistic environment are only simulated.
  It is indeed very useful to realize simulations of cost and time intensive experiments before one completes a final real (physical) implementation. One has, for instance, the possibility to experiment several different control algorithms for evaluating them. Realizing statistical studies is also much easier with simulated embodied autonomous agents, because implementations are more efficient and faster. In certain cases, a lack of physical resources (physically embodied) also motivates the implementation of simulated embodied agents.
- Finally, *software agents* [Nwa96], [Sho99] directly interact with software environments. They lack of a body, sensors and motor, but are nevertheless situated within an often complex software environment. Famous examples are the *interface agents*, which are personal assistants of a user, and the *information agents*, which find on the WWW information specifically interesting a user [Mae94].

The second and third class of agents are obviously computational agents. The presented taxonomy is summarized in figure 2.2. As we shall see in section 2.5, our target class of MASs comprises simulated embodied agents.

2.2.3 Generic Agent Architectures

Autonomous agents are systems in continuous long-term interaction with an environment in which their are situated. This constitutes a common basis for all autonomous agents. But based on so-called agent theories [WJ95], different *generic agent architectures* have appeared, differing from one another in their view of the intra-agent aspects. They can be grouped in *mental*, *reactive* and *hybrid agents* (for an overview, see for instance [WJ95] or [Oss99]).

Mental Agents. Mental agents, also named *rational* or *deliberative agents*, are viewed as intentional systems [Den87], using mentalistic notions applied to humans. The agent has an explicit symbolic model of the world in which it lives. It uses sensor data in order to update its model of the environment. A planner reasons on the world model and decides which actions to realize.

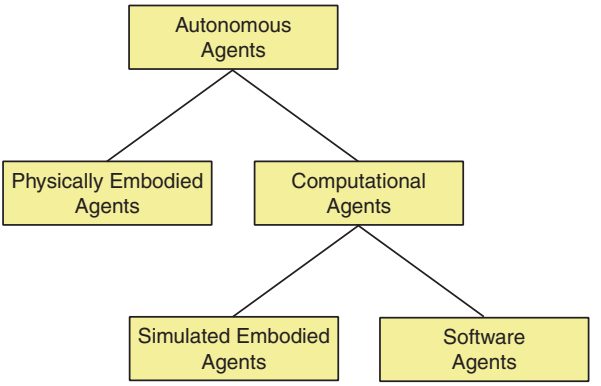

Fig. 2.2. A basic agent taxonomy in relation to the notion of embodiment.

In contrast to the *weak* notion (see section 2.2), Wooldridge and Jennings consider mental agency as a *strong* notion of agency [WJ95]. Note that it is obvious that mental agency actually inherits experience from the symbolic AI. This constitutes thus a compromise between the main streams of AI (see section 2.1).

Of the best known mental agents are the BDI agents, based on the BDI theory (*belief*, *desire* and *intention*) of Rao and Georgeff [GR95].

Reactive Agents. Reactive agents are based on the idea that *the World is the best model* [Bro94]. Intended to handle basic behaviors, they base their architecture on simple routines schemes, excluding abstract reasoning. These simple actions seem indeed easier to perform basic behaviors. The methodology is bottom-up: agents react to changes in the environment in a stimulus-response fashion by executing the simple routines corresponding to a specific sensor stimulation.

A famous reactive agent architecture is Brooks' subsumption architecture [Bro86]. Brooks' robot possesses a layered architecture, where each layer represents a simple behavior reacting to specific stimuli. The high layers can control the most primitive layers by filtering their input data and preventing their output data from influencing the effectors.

Hybrid Agents. Some work tries to unify mental and reactive agencies in order to surmount their respective weaknesses. The idea is the following: basically, these agents are steered by their simple routines reacting to basic stimuli. However, the deliberative module controls the reactive one when it wants to perform stimulus-free actions (like reasoning) or to change long-term goals.

Important examples are Müller's InterRAP architecture [Mue96] and Ferguson's Touring Machines [Fer95].

2.3 Characteristics of MASs

As indicated by its name, a *Multi-Agent System*[1] (MAS) is a macro-system comprising multiple agents, each of which is considered as a micro-system. MASs result therefore from the organization of multiple agents within an environment. This organization is done with a specific hypothesis and goal. Actually MAS research expects that an ensemble resulting from the interactions of individual agents possesses a value-added towards each simple agent capability [NN99]. This is the basic motivation. But what are the characteristics of MASs? The following can be cited [JSW98]:

- In a MAS, every agent has a subjective view; it can only have incomplete information of the system, because its viewpoint is limited.
- As a consequence of the first characteristics, no global control is applied. Each agent has its proper state that is not accessible by other participants in the system. This state changes individually as a result of the behavior rules of the agent. These rules are themselves influenced by data sensed in the environment. On its turn, an agent influences the environment by acting on it.
- The data is fully decentralized, distributed in the participating agents and the environment.

At a modeling level, agents in a MAS are concurrent from one another: each agent is conceived to execute independently from the other agents. In many cases, however, this theoretical concurrency is not guaranteed by an underlying implementation that realizes a serial simulation of a system (see for instance [LBDL99], [Mic99]). Still at a modeling level, a MAS integrates a specific organization of its environment in which agents evolve. This organization can be imposed by either the model or the underlying physical layer.

Although sometimes it is possible to conceive a system with a single agent, this would simply reduce a problem to wrapping a program so that it can interact with a user. This has been, for instance, realized by wrapping expert systems for assisting users in making decisions. Hence, MAS applications have several advantages in comparison to single-agent systems:

- The implementation of problems that ask for distributed data and control over them are more naturally realized using MASs.
- They tend to robustness, because the failure of an agent can be overcome if another agent takes in charge the uncompleted work.
- Scalability: a MAS should easily be extended by adding new agents.
- Reusability: thanks to their modularity, agents should be easily re-integrated in a new MAS.

[1] Several reviews on research in multi-agent systems are available; see for instance [Bra97], [Zie97], [HS98], or [Wei98].

If MASs present real benefits, their construction shows several inherent problems that challenge the research. Some of these problems are applicable to MASs of every type; others are related to specific views. Among the basic questions that should be solved are the following: Which communication schemes should be chosen? What are the means of interaction within a MAS? How should communication and computation be balanced? When agents tend to a common goal, how should they coordinate with one another? How is conflict between agents handled?

The list of questions is by far incomplete. However the main problem remains a software engineering one: how can practical MASs be modeled, designed and implemented? This book contributes to an answer to this question.

We do this in the following way:

- We sustain, in the next section, that the modeling of a MAS should explicitly integrate the interaction setup of the MAS by clearly identifying what we call objective and subjective dependencies, and we insist upon the management of objective dependencies, leading to what we denominate objective coordination.
- We present, in section 2.5, a specific class of MASs as our target and show how it can be modeled.
- We discuss, in section 2.6, actual proposals for helping the implementation of MASs, and, as we promote a focus on objective coordination, we advocate in section 2.6.3 the use of coordination models and languages for designing and implementing objective coordination in a MAS.
- We present in the subsequent chapters a concrete coordination model named ECM and a corresponding coordination language named STL++ that we have tailored towards our target MAS.
- Finally, an implementation of an application of our target MASs is presented using STL++.

2.4 Modeling MASs

Interaction between agents is absolutely essential in a MAS, because it is the "raison d'être" of the MAS. If agents are not able to interact with one another, no global behaviour in the MAS is possible. One has therefore to model the interaction setup of the multitude of agents participating in the MAS. If this is not done, this typically leads to an *agent-oriented* view of a MAS [COZ99b]. This agent-oriented view models a MAS by describing the intra-agent aspects, such as the agent's representation of the world, its beliefs, desires, intentions, and by neglecting the description of the agent interactions and of the space where these interactions take place.

This necessity for a clear identification of the interaction setup in a MAS naturally calls for a separation between the design of the individual tasks

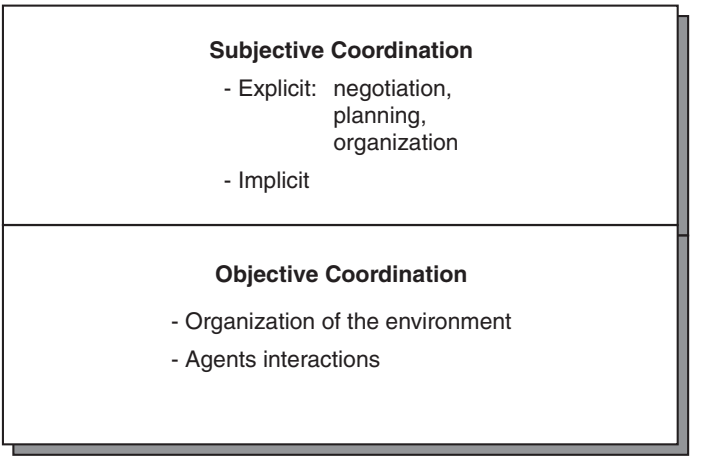

Fig. 2.3. Coordination in a MAS.

of each agent and the design of their interactions. Several researchers have recognized this point, by proposing to explicitly integrate the interaction setup and the handling of the resulting dependencies in the analysis phase of a MAS [Bur96], [Sin98b], [WJK99], [COZ99b], [COZ99a]. We maintain that this can be done at two different levels, according to the types of dependencies.

Indeed the modeling of the setup of multiple agents into a MAS leads to the detection of many dependencies of different nature. On one side, these dependencies rely on the result of the external composition of multiple agents into an ensemble; on the other side, they result from the individual or peculiar point of view of each agent interacting with other agents. We thus distinguish between two types of dependencies [SCH99] and, as coordination is defined as *managing dependencies between activities* [MC94a] (see section 3.1), two corresponding types or levels of coordination (see figure 2.3):

– A MAS is built by *objective dependencies* which refer to inter-agent dependencies, namely the configuration of the system in terms of the basic interaction means, agent generation/destruction and organization of the environment. We refer to the management of these dependencies as *objective coordination*, because these dependencies are external to the agents. Thus, objective coordination is essentially concerned with inter-agent aspects.
– Agents have *subjective dependencies* which refer to intra-agent dependencies towards other agents. The management of these subjective dependencies refers to what we call *subjective coordination*. Thus, subjective coordination is essentially concerned with intra-agent aspects.

Subjective coordination is dependent from objective coordination, because the first is based on and supposes the existence of the second. The

mechanisms that are engaged to ensure subjective coordination must indeed have access to the mechanisms for objective coordination. If this is not the case, no subjective coordination is possible at all. This does not mean that objective coordination belongs to the intra-agent view, but only that the access mechanisms have a subjective expression in the agent. This is essentially the case for mechanisms like sending or receiving information.

We thus maintain that, when modeling a MAS, the first step is to identify which objective dependencies are present and how they can be handled. It is only after this identification that all subjective dependencies can in turn be identified and described. Not differentiating the two levels of coordination leads to MASs that resolve objective coordination with subjective coordination means, i.e. by using intra-agent aspects for describing system configurations. For instance, a MAS intended at modeling the hierarchy in an organization would model this hierarchy internally in each agent by means of knowledge representation of the hierarchy, and would not describe it by establishing the communication flows that represent it.

This mixture of subjective and objective aspects is typically present in MASs composed of mental agents that use agent communication languages (ACL) [LFP99] in order to communicate. The principal aim of ACLs, such as KQML [FFMM94], [LF97] and the FIPA's proposal [FIP99b], [FIP99a], is to offer means for exchanging information and knowledge. But they do this by integrating in a message simultaneously two types of information: (i) lower-level communication information (such as the identities of the participating agents, a unique communication identifier, or the used protocol), and (ii) meta-information about the content of the message, such as the general intention or the type of a message in term of speech acts [Sea69], [Sin98a] (examples are *request*, *deny*, *propose*, *confirm*), and a common understanding, possibly described with ontologies [Gru93] identifying a common semantics between the agents.

In the next sections, we discuss more in detail both objective and subjective coordination. Actually, we classify existing general MAS issues into the two proposed dimensions.

2.4.1 Objective Coordination

Objective coordination is mainly concerned with the organization of the world of a MAS. This is achieved in two ways: i) by describing how the environment is organized, and ii) by handling the agent interactions. We address these two points.

Organization of the Environment. The organization of the environment varies according to the space the MAS wants to integrate. We thus propose to distinguish between *implicit* and *explicit environment organizations*.

An implicit organization does not explicitly model the environment, because it is given or imposed by the underlying logical structure on which a

MAS evolves. This is, for instance, the case for network-aware agents roaming on the World Wide Web, where the environment is structured by the nodes of the network.

An explicit organization, however, establishes a model of an environment that does not necessarily reflect the intended logical structure. This can be done by realizing an approximation of the target. When, for instance, one wants to simulate a continuous physical space, he will not be able to keep the continuity and will have to render the space discrete.

But, more importantly, the environment allows the arising of the interactions between the agents. We discuss hereafter how this can be completed.

Agent Interactions. Handling agent interactions asks for the description of the interactions between an agent and its environment, and the interactions between the agent themselves.

As the environment also have a container function, it can be used to communicate. Indeed, an agent has a relation with its environment by means of its perception. Furthermore, it can influence the state of its milieu with specific actions. The interaction with the environment can then be understood and used to communicate: all information is transmitted within the environment. Communication thus becomes an action that influences the environment. Consider, for instance, the case of an insect-like robot dispersing in the environment information similar to pheromone. This information can be sensed by another robot that notices it as a trace of the roaming of the first agent.

But the interactions between agents are usually based on specific communication means. From the perspective of the direct concerned recipient of the communicated data, we classify these communication means between agents in four basic paradigms (figure 2.4 pictures them):

- *Peer-to-peer* communication: messages are sent directly to specific agents. This is usually done by identifying the partners, for instance with an email-like address (message-passing-like communication). It is also possible that an intermediate channel takes in charge the transmission of the data, and that the partners of communication do not know from each other.
- *Broadcast* communication: a message is sent to everybody in the MAS. Interested agents can evaluate the received data or ignore it.
- *Group* communication: A message is sent to a specific group of agents.
- *Generative* communication: communication is realized through a blackboard [HR88]: agents generate persistent objects (messages) on the blackboard, which are read by other agents. The reading can be done independently of the time of the message generation; thus the communication is fully uncoupled.

Furthermore, it is possible to distinguish *identified* and *anonymous* communication. Identified communication requires an identity of its partner. In

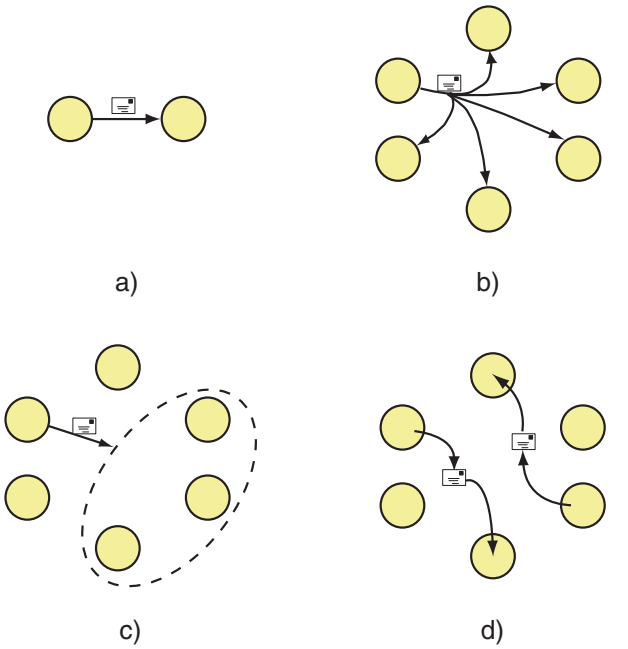

Fig. 2.4. Basic communication paradigms: a) peer-to-peer, b) broadcast, c) multicast and d) generative communication.

anonymous communication, the agent producing the message ignores its recipient, and vice-versa. This can be achieved, for instance, by creating in the agent ports having the role of communication interfaces; these ports, on which an agent puts and reads messages, are connected to other ports belonging to other agents.

2.4.2 Subjective Coordination

We distinguish two types of subjective coordination: *explicit* and *implicit* subjective coordination. Their differentiate themselves in the explicit or implicit treatment of the management of subjective dependencies.

The research in distributed artificial intelligence (DAI) has proposed several coordination techniques [Jen96], [NLJ97], [Oss99] that deal with explicit subjective coordination. These techniques typically consider coordination as *the process by which an agent reasons about its local action and the (anticipated) actions of others to try and ensure the community acts in a coherent manner* [Jen96]. Thus, these techniques are qualified as subjective coordination, because they try to resolve subjective dependencies by means of intra-agent structures that often involve high-level mentalistic notions and appropriate protocols. We characterize these techniques as explicit, because they explicitly handle coordination.

In [Oss99], Ossowski classifies them in three families: *multi-agent planning*, *negotiation*, and *organization*. We briefly explain these families:

Multi-agent Planning
: With multi-agent planning techniques, agents build a plan and commit to behave in accordance with it. This plan describes all actions that are needed to achieve the respective goals of the agents. Note also that planning can be centralized or decentralized.
: An overview on multi-agent planning techniques can be found in [Dur96].

Negotiation
: Negotiation has constituted the most significant part of DAI research in coordination. It can be defined as *the communication process of a group of agents in order to reach a mutually accepted agreement on some matter* [BM92]. Often starting from contradictory demands, the negotiation process generates agreements that can be later re-negotiated.
: A review on negotiation techniques can be found in [NLJ97].

Organization
: Organization techniques support a priori organization patterns by defining *roles* for each agent. A role determines the expectations about the agent's individual behaviour by describing the agent's responsibilities, capabilities and authority inside the MAS. An agent then consults its organizational knowledge of its role in the MAS and acts in accordance with it. Although the purpose of organization belongs to objective coordination, the notion of role typically refers to a subjective dependency towards other roles in the MAS. This is the reason why we consider these techniques as belonging to subjective coordination.
: It is worthy to note that these methods generally suffer from lack of dynamicity in the structure of the organization, because the roles are thought as long-term relationships.
: A valuable review on organization theory applied to DAI can be found in [CG98].

Agents may also coordinate themselves implicitly, without having explicit mechanisms of coordination. This may be, for instance, the case in the framework of collective robotics [BHD94], [MM95], [Mat95], in which robots act on the base of the result of the work of other robots. Thus, this result, which is locally perceived through the sensors, allows to resolve a subjective dependency, namely the necessity to sense a specific information in order to act. This kind of coordination, which is also named *stigmergic* coordination, literally means an *incitement to work by the product of the work* [BHD94]. Our target class of MASs typically allows such a coordination (see section 2.5).

2.4.3 Emergence

We tackle now an aspect that does not belong to the modeling of a MAS, but that constitutes its goal: emergence.

Indeed, MASs are interested in an interaction-centered perspective. Dealing with agent interactions leads naturally to the concept of *emergence* of behavior and functionality [Ste94], [ECJS94]. According to Forrest [For90], emergence can be defined as follows: a system's behavior is considered emergent if it can only be specified using descriptive categories which are not to be used to describe the behavior of the constituent components. Two levels of behaviors are thus differentiated: i) the behavior of the constituents, ii) and the global behavior risen from the conjunction of the constituents. The definition shows that it is not possible to specify emergence, because it is only the result of the interactions of multiple agents.

Emergence offers a bridge between the necessity of complex and adaptive behavior at a macro level and the mechanisms of multiple competences at a micro level [For90]. Although emergence keeps a subjective aspect by directly depending on the cognitive properties of an observer [CLC99a], it has the advantage of basing on simple components and at achieving a global behavior by letting agents interact.

2.5 Our Target Class of MASs

The preceding sections have discussed the modeling of MASs. Before discussing implementation issues in section 2.6, we describe a specific class of MASs; this class is attempted to be implemented on distributed architectures. In chapter 8, we will present an implementation of an application of this class of MASs using our coordination language STL++.

The work presented in this book has been developed in the framework of research aimed at solutions for applications in the fields of robotics and distributed systems. For this reason, we are mainly interested in simulated embodied autonomous agents [CKS+98], [CDSH98]. More precisely, we design multi-agent simulations of collective robotics [BHD94], [MM95], [Mat95], [CH99], whereby global tasks are achieved thanks to emergence. Our simulated robots are operationally autonomous agents: there is no centralized control of the behaviour of the agents, because each agent acts by itself on the basis of its perception and in accordance with its rules (sensing and acting are strictly local to the agent). Furthermore no explicit subjective coordination is engaged, because we follow a stigmergic subjective coordination [BHD94] (see section 2.4.2): an implicit coordination is achieved on the basis of the result of the work of each agent. Thus the only interactions are realized through the influences in the environment.

We organize this section as follows. In 2.5.1, we present a generic model [CKS+98] for our intended autonomous agents' system. Then in 2.5.2, we

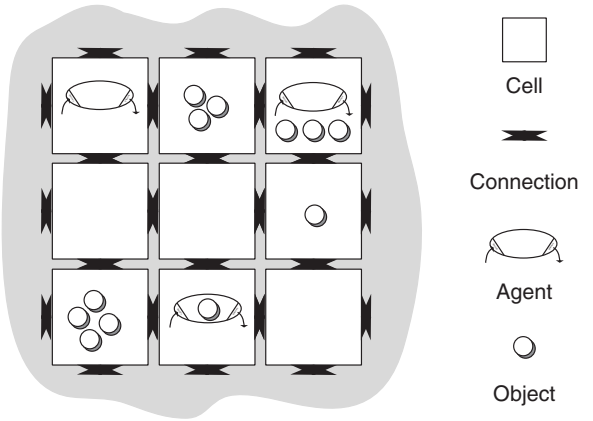

Fig. 2.5. Generic MAS Model.

present a peculiar application belonging to our target class of MASs. This model and its application, which have been defined in the PAI group[2], have been the subject of several studies on the emergence of behaviour [CDS96], [CDSH96], [DCH97], [CLC99b], [CLC99a].

2.5.1 A Generic Model for an Autonomous Agents' System

We first tackle the organization of our generic model by actually describing the objective coordination. The model is composed of an *Environment* and a set of *Agents*. The *Environment* is composed of a set of *Cells*. Each cell includes i) a set of on-cell available *Objects* at a given time, which can be manipulated by agents, and ii) a list of connections to other cells. This list of connections defines a *Neighborhood* which implicitly determines the topology. As this neighborhood can be defined separately for each cell, any type of topology may be achieved. Figure 2.5 sketches the model presented for a four connectivity. An agent possesses some sensors to perceive the environment within which it moves, and some effectors to act in this environment (embodiment). The perception is local, because, at a given time, an agent perceives only on one cell or on a restricted collection of cells. Therefore, interactions between agents are limited to their influences on the environment.

Concerning the intra-agent aspects of the model, a general agent architecture is defined, complying the autonomous agent definition we have adopted in section 2.2. Figure 2.6 sketches the proposed architecture. The modules *Perception*, *State*, *Actions* and *Control Algorithm* depend on each application; therefore, they are under the user's responsibility. The degree of autonomy of an agent is dependent upon the control algorithm module, because this

[2] Parallelism and Artificial Intelligence Group of the Computer Science department at the University of Fribourg.

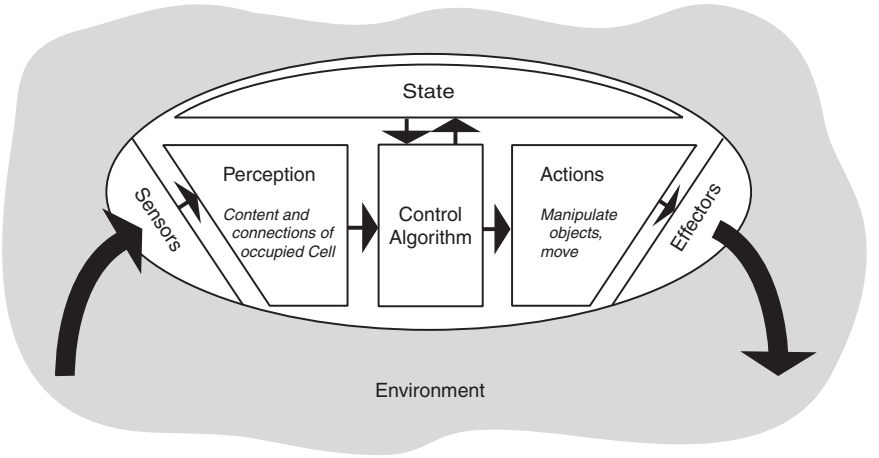

Fig. 2.6. Our target agent architecture.

module determines if an agent is *operationally* or *behaviorally* autonomous [Zie97] (see section 2.2.2). For instance, a very basic operational autonomy would consist of randomly determining which actions to perform; a more sophisticated behavioral autonomy would integrate learning capabilities using, for instance, a neural network.

2.5.2 A Typical Application: *Gathering Agents*

The model presented above can support numerous applications. Among others, it can be used for a simulation of mobile collective robotics [BHD94], [MM95], [Mat95], [CH99]. In fact, we present such a simulation, in which agents simulate the behavior of a real robot seeking objects spread in their environment. As a result of the individual behaviors of each agent, a global behavior is achieved, namely that of having agents stack all the objects, as displayed in figure 2.7.

In the simulation, the agents are operationally autonomous, because each agent in the system acts freely on a cell: the agents choose an action depending on their control algorithm and local perception. The agents do not explicitly negotiate, nor explicitly identify and handle antagonism situations (conflicts between different agents' goals). Every interaction is done in an indirect way through influences upon the environment. Thus, there is no explicit subjective coordination.

The environment is made of a discrete two dimensional L cell-sided grid, a set of No objects and Na agents roaming from one cell to the other. An object is either situated on a cell, or carried by an agent. Several control algorithms are possible (see for instance [CDSH96] and [DCH97] for various proposals).

Fig. 2.7. Collective robotics application: stacking objects.

A simple control algorithm can be explained as follows. Agents roam randomly from one cell to a connected one. As long as an agent does not sense any object in its cell, it moves to another neighbor cell. When it arrives at a cell with *no* objects, two cases are possible: i) if the agent does not carry any object, it will take one object with a probability given by $1/\mathtt{no}^{\alpha}$, where $\alpha \geq 0$ is a constant; ii) if the agent is carrying an object, it will always put down the object.

The simulation presented has been serially implemented [CDS96], [CDSH96], [DCH97], [CLC99a], [CLC99b], showing the emergence of global properties in the whole system: the agents cooperate to achieve a task without being aware of it. In chapter 8, we will present a distributed implementation of the simulation using our coordination language STL++.

2.6 Implementing MAS Applications

Once the model of a MAS realized, one has to lead this model to a concrete implementation. Actually, a very important issue in agent technology has been the elaboration of *languages* for facilitating the implementation of MASs, although a lot of realizations have used ad hoc solutions, without trying to present an abstracted layer on which to build. These languages should fill the gap, often encountered, between the model of a MAS and its implementation.

In this section, we review several important proposals of languages for MASs. We then discuss design methodologies that are emerging from the experience acquired with these languages. Finally we show why these approaches are inadequate and conclude by advocating the use of coordination models and languages for designing and implementing MASs.

2.6.1 Languages for MAS Applications

As most MAS implementations have been built in an ad hoc manner, several authors have proposed *agent languages* that offer a layer of abstraction on

which to develop MAS applications. We distinguish two main trends. On one side, many proposals follow the approach of the agent-oriented programming paradigm [Sho93]. On the other side, concurrent object-oriented programming languages [AWY93], [BGL98] are presented as a base for implementing MASs [GB92], [KB98], [Bri98], [GB99].

Agent-Oriented Languages. Shoham has proposed an agent-oriented programming (AOP) paradigm [Sho93] for programming MASs. This view suggests that agents be directly implemented with mentalistic notions.

In an AOP system, a *formal language* describes the agent mental states. It uses modalities that are more basic than those of BDI[3] [GR95]; they encompass *beliefs*, *decisions* and *capabilities*. The agent's actions are determined by its *decisions* (possibly in the past). On their turn, the decisions depend upon the agent's *beliefs*, which refer to the state of the environment (including other agents' mental states) and the proper mental state, as well as the *capabilities* of itself and others. Furthermore, Shoham slightly modifies the decision notion by replacing it by *commitments*, treating decisions as obligations.

The second element of an AOP system is an *interpreted programming language* named AGENT-0 that is based on the formal language and that concretely programs the agents. AGENT-0 defines an agent in term of its capabilities (what it can do), its initial beliefs and commitments, and a set of *commitment rules*, which describe the agent actions. A commitment is composed of a *mental condition*, a *message condition*, and an *action*. A mental condition is a combination of modal statements referring to the mental state of the agent. A message condition is a combination of *message patterns*. A message pattern is defined by the identification of its sender, the type of message, and a content as a fact or action statement. In order to trigger the rule, the conditions must be fulfilled: the mental condition is matched against the beliefs of the agent, and the message condition is matched against the received messages. An action can be *private* by executing a subroutine or *communicative* by sending a message.

Shoham proposes a last AOP component: an *agentifier* allowing the translation of neutral devices into programmable agents. It is a translation process that applies the reverse method of situated automata [Ros85]. Starting from a low-level description of the device (using a process language), the agentifier has the task of producing an intentional description of the device; it is therefore an "agentification" process. This should permit the integration of existing devices in an AOP system.

A disadvantage of AGENT-0 is that the correspondence between the formal language and the interpreted programming language is loosely defined. CONCURRENT METATEM [Fis94], [Fis97] enhances this drawback by providing an agent language that faithfully executes its associated formal language. In fact, the behaviour of an agent is defined with a set of temporal logics specifications [Eme90] represented by logical rules. And this specification is

[3] Belief, desire, and intention.

straightly executed in order to produce the behaviour of the agent. As the execution of an agent is described by temporal logics which models time as a sequence of discrete states, this execution depends at each moment on the history of the agent.

AGENT-0 and its agent-oriented programming paradigm have inspired and influenced a lot of other work. Thomas has developed PLACA [Tho93] which improves AGENT-0 with planning and communication requests for action via high-level goals. AGENTSPEAK [WRR94] allows agent programs to be written and interpreted in a manner similar to that of horn-clause logic programs, integrating aspects of concurrent-object technology.

Concurrent Object Oriented Languages. A lot of applications implementing MASs use an existing programming language as a basis for their realization and do not employ a language tailored toward their implementation. In a discussion about the relationships between agent languages and other programming paradigms [Mey98], Clark promotes the idea of basing agent programming languages on Concurrent Object Oriented Languages (COOLs) [AWY93], [BGL98]. This point of view is also argued, for instance, in [GB92], [KB98], [Bri98] and [GB99], where the authors are convinced that COOLs are especially useful as a basis for implementing MASs, although these languages are not directly agent languages. This approach is precisely taken by APRIL [MC94b], as we shall see later in this section.

COOLs combine concurrency with object-orientation, leading to the notion of *active object*. An active object adds to the object design the capacity to have its own activity. This leads to programs with inter-object concurrency. Certain languages also allow degrees of intra-object concurrency, i.e. several threads of execution inside the same object[4].

While the passive object model follows a reactivity principle (methods of an object are invoked synchronously by reception of a message), active objects are active since their creation. Some COOLs keep reactivity, like ACT++ [KL90]. But most of them respect more autonomy, by requiring the active objects to accept incoming messages and handling them explicitly. COOLs contain, however, an important drawback to their basic aim of being object-oriented, particularly in what concerns code reuse by inheritance. This drawback is called the *inheritance anomaly* [MY93], [McH94]. Essentially it is originated in the interference between inheritance and synchronization constraints, which often causes re-implementation of synchronized code.

The property of encapsulation of a private state and the support for a proper thread of execution make active objects an attractive layer on which to build autonomous agents. This has led to the use of several COOLs to implement agent-based applications. The COOLs that implement the ACTOR model [Hew77], [Agh86] have a strong influence in this respect [KB98]. As showed in figure 2.8, an actor communicates asynchronously by sending messages to other actors whose addresses it knows. At the receiver actor, the

[4] For a discussion on the levels of object concurrency, see [Weg90].

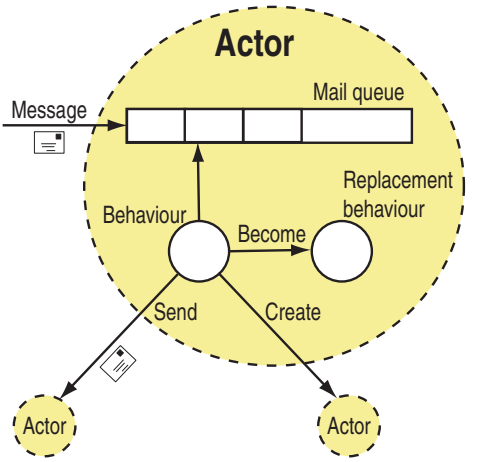

Fig. 2.8. The ACTOR model.

reaction to an incoming message is determined by its behaviour defined by a script. This behaviour allows an actor to send new messages, create new actors, and determine its own subsequent behaviour by the next message arrival. The ACTOR model supports both inter-actor and intra-actor concurrency. Several implementations of the ACTOR model exist, see for instance [Bri89] and [KML93].

A well-known example of object-oriented concurrency for MASs is APRIL [MC94b]. This language is a strongly typed object-based concurrent language with active objects as processes. It is oriented towards the implementation of multi-agent applications. Thus, it is not, as such, strictly an agent language, but it can be applied to implement further agent-oriented languages. Actually, APRIL has been used to realize MAIL [HSM94], a high level language that can be used for implementing MASs. Fundamentally, APRIL defines means for process creation and inter-process communication in a message-passing style, possesses higher-order features that allow program structuring with modules (records of functions and procedures), and permits the sending of code in messages.

We have identified two main groups of existing languages for implementing MASs. On one hand, agent-oriented languages such as AGENT-0 have a typical intra-oriented view of MASs, because they implement a MAS by specifying the mental structures of the agents. They do not support an emphasis of objective coordination in the development of a MAS. On the other hand, COOLs are only considered as a basis for implementing MASs, because the active object model seems adequate for supporting the implementation of agents. But originally COOLs do not directly tackle the space organization of the active objects and their coordination. For that reason, some

enhancements around the basic ACTOR model have been proposed, for instance by structuring the actors in *Actor Spaces* [CA94], which are passive containers, or by organizing them in *Casts* [VA99], which are groups handled by a director. Another approach uses *Synchronizers* [FA95], which allow declarative constructs of coordination activity between several actors. These enhancement enter the framework of the research on *coordination models and languages* [CG92a], [PA98c], which precisely give necessary abstractions for handling objective coordination. But before discussing coordination models and languages in the next chapter, we review methodologies for the implementation of MASs and discuss the appropriateness of coordination models and language for MASs.

2.6.2 Methodologies for MAS Applications

The development of numerous agent languages had the advantage of bringing the theories of agenthood on a practical ground by offering concrete means to program MASs. This has supported the beginning of an engineering view of multi-agent system construction. Nevertheless, the need for methodologies has recently arisen as a help for the design of MAS applications [FMS+97], [IGG98].

Several methodologies have been, to date, proposed, all remaining at an experimental stage [Oss99]. They concern mostly the design of mental agents systems. Furthermore, none has yet gained a large acceptance. [IGG98] reviews several approaches to agent-oriented methodologies, all mostly extending other existing methodologies. Two main groups have been proposed, which are respectively based on knowledge engineering or object-oriented methodologies.

Research extending knowledge engineering methodologies is intended for mental agents. They center their design in the *knowledge acquisition process*. They present, however, an important drawback, by conceiving a knowledge based system as centralized. Two important works are the CoMoMAS [Gla96] and MAS-CommonKADS methodologies [IGGV97].

Object-oriented languages are often used as a natural framework for the construction of agent-based applications. Furthermore, the relation of agents to active objects in the COOLs has already been discussed in section 2.6.1. Therefore, it was natural that methodologies such as the *Object Modeling Technique* (OMT) [RBP+91] have been proposed as a base for agent-oriented methodologies. In [Bur96], for instance, Burmeister proposes a methodology based on OMT. The author defines three models and appropriate process steps for the design of multi-agent applications, all conceived in a mental perspective. First, in the *agent model*, agents are described by their beliefs, motivations, plans and cooperative attributes. Second, the *organization model* uses an OMT notation for stating the inheritance hierarchy and the roles of each agent. Third, a *cooperation model* identifies partners, messages and used

protocols, all considered in a mental dimension. Another proposal based on object-orientation is, for instance, the method for BDI agents [KGR96].

The presented methodologies concentrate therefore on intra-agent aspects and do not really tackle objective coordination as a design dimension. In the next section, we motivate the use of coordination models and language for designing and implementing MASs.

2.6.3 Using Coordination Models and Languages for Designing and Implementing MASs

The research has lead to solutions for facilitating the design and the implementation of MASs. These proposals try to fill the gap, often encountered, between agent theory and agent applications. However, as noted in [COZ99a] and [COZ99b], the proposed solutions are mostly concentrated on intra-agent aspects. They generally design MASs in terms of the internal agent structures, based on mentalistic notions such as the BDI model [GR95], and concentrate on the resolution of subjective dependencies using several (often useful) coordination techniques. Generally, however, they do not explicitly design and treat objective coordination in the MAS, or they do it implicitly by considering, most of the time, only peer-to-peer communication schemes.

However, we have seen that, when modeling a MAS, several objective dependencies clearly appear. The modeling and the resulting design of a MAS is therefore facilitated by separating what concerns inter-agent aspects (objective coordination, i.e. the description of the environment and the agent interactions) from intra-agent aspects (agent tasks and subjective coordination). This idea analogously follows the proposal of Carriero and Gelernter [CG92a] which advocates that programming can be understood as the combination of computation and coordination (see section 3.2.1). Actually, objective coordination is orthogonal not only to the agents tasks, but also to subjective coordination, because this latter handles intra-agent aspects that are in relation with other agents.

Thus, we sustain that it is only on the basis of this separation that one can appropriately design and implement a MAS. And, like a few recent work such as [COZ99a] and [COZ99b], we maintain that this design and implementation can be done with an appropriate coordination model and language.

Indeed, research from concurrent and distributed computing, programming languages and software engineering has developed several *coordination models and languages* [CG92a], [PA98c] (discussed thoroughly in the next chapter). They are suitable precisely for supporting the design and the implementation of objective coordination in MASs, because they define abstractions dealing with the management of the interactions between activities and the organization of the activity space. We therefore propose the use of a coordination model to sustain the design phase of a MAS, and the use of a

corresponding coordination language (the linguistic embodiment of the coordination model) to implement the MAS.

The use of a coordination model for designing a MAS is also promoted in [COZ99a] and [COZ99b], but the authors go even further by following a stronger societal view of MASs [ST95], [Sin98a], [HOY+99]. They maintain that a MAS design should define social laws that regulate the interactions between the agents. These laws should be extensible by having the possibility to specify new rules regulating the communication means. This may be achieved by defining in the interaction space new rules reacting to and ruling communication events. Actually, the authors propose a coordination model that integrates a so-called *programmable coordination medium* [DNO97] (see section 3.5) enabling the definition of new rules detecting and reacting to communication events.

Employing a coordination model and language becomes even more crucial if a MAS runs on a distributed and concurrent architecture, which should be the natural substrate for its execution. In fact, coordination models and languages are utilized for implementing applications executing on such architectures, and have therefore a strong expertise in concurrent and distributed computing. We explain this point.

At a conceptual level, agents in a MAS run concurrently and are organized in several activity spaces. It seems thus obvious that concurrent and distributed architectures should constitute the perfect substrate for MASs. However, the projection of a MAS onto such architectures is not trivial [DC97]. This is especially true for the class of MASs we are interested in, namely simulated-embodied autonomous agent systems, because they possess an explicit organization of an environment (see section 2.4.1) that simulates a continuous space, and this organization does not comply with the structure of our underlying architecture. This may be the main reason why most implementations of this class of MASs are based on sequential architectures with round-robin mechanisms. This is, for instance, the case of important toolkits such as SWARM [LBDL99], SIM_AGENT [SP95], [Slo99], and the KHEPERA Simulator [Mic99].

At an implementation stage, we thus have to deal with the organization of the actual processes, the active pieces of software which represent the agents, along with the inherent problems of shared resources and data, synchronization and consistency concerns. Coordination models and languages are precisely aimed at providing solutions for these problems: they use coordination media and tools so as to deal with the consequences of the concurrency and the spatial distribution of the supporting processes.

Therefore, to realize a MAS on a concurrent and distributed architecture in the most natural way, an appropriate coordination model and corresponding language are needed. On one hand, the coordination model should reflect the image of the MAS model (objective coordination at the modeling level) with the utmost fidelity, while preserving the structure of the agents

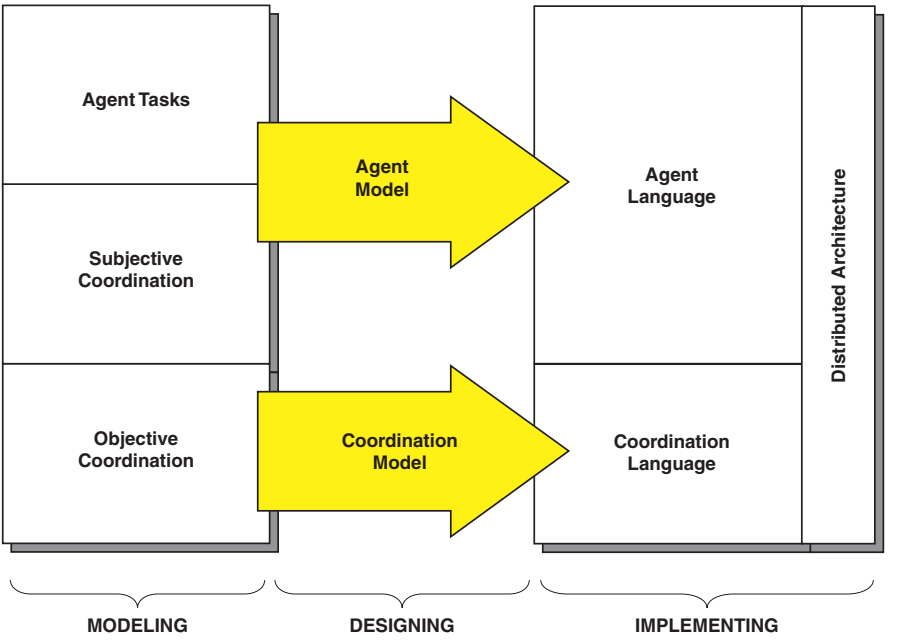

Fig. 2.9. Implementing MASs.

and their behavior, such as, for instance, their degree of autonomy. On the other hand, the coordination language must enable a mapping of the MAS on a distributed and concurrent architecture by managing the creation and destruction of the processes that support the agents and by insuring their interactions and organization within structured and hierarchical spaces.

In conclusion, our approach, which is sketched in figure 2.9, can be summarized in three steps:

Modeling: One first analyzes which objective coordination is needed in the MAS. It is only on this basis that one should identify two remaining crucial issues, namely the agent tasks, as well as the arisen subjective dependencies and, if necessary, the handling of these dependencies.

Designing: An appropriate coordination model is applied for designing the objective coordination in the MAS, and an agent model is also applied for designing agent tasks and subjective coordination.

Implementing: The realized design is implemented using a corresponding coordination language and, respectively, an agent language, which can be, in some cases, a common computation language.

In the next chapter, we thoroughly review what are coordination models and languages. Subsequently, we present the ECM coordination model and its instance STL++.