A Bio-inspired Representation Model for Engineering Self-Organizing Emergent Systems

Maíra Athanázio de Cerqueira Gatti, Carlos José Pereira de Lucena

Departamento de Informática
Rua Marques de São Vicente 225, 4º andar RDC
Rio de Janeiro – RJ – Brasil

{mgatti, lucena}@inf.puc-rio.br

Abstract. *Self-organization is a dynamic and adaptive process where components of a system acquire and maintain information about their environment and neighbors without external control. Agent-Oriented Software Engineering (AOSE) can provide methods and technologies that help building self-organizing systems. A fundamental engineering issue when designing self-organizing emergent multi-agent systems (MASs) is to achieve required macroscopic properties by manipulating the microscopic behavior of locally interacting agents. In this paper, we propose a bio-inspired approach consisting of a representation model that allows a systematic design of the desirable emergent macroscopic properties from a macro scale to the micro scale.*

Resumo*. Auto-organização é um processo adaptativo e dinâmico onde componentes de um sistema adquirem e mantém informação sobre o ambiente e seus vizinhos sem controle externo. A Engenharia de Software Orientada a Agente pode fornecer métodos e tecnologias que apóiam a construção de sistemas auto-organizáveis. Uma questão de engenharia fundamental durante o projeto de sistemas multiagentes organizáveis é alcançar propriedades macroscópicas desejadas para manipular o comportamento microscópico das interações locais dos agentes. Neste artigo é proposta uma abordagem inspirada na biologia que consiste de um modelo de representação que permite um projeto sistemático das propriedades emergentes macroscópicas desejadas a partir do nível macro para o micro.*

1. Introduction

Self-organization is a very powerful approach to the engineering of complex systems because of its many interesting properties, including adaptation and robustness [Mameia et al., 2006]. Unfortunately, self-organization provides difficult design challenges as we are using apparent microscopic behavior of system components and their interactions to achieve a deterministic result for the entire system.

However, biologists and computer scientists are collaborating closely as they seek to understand and to influence the natural information processes studied in computing [Denning, 2007]. Computing interacts constantly with other fields. The other fields teach us more about computing and we help them find better ways to model and perhaps understand the world. Thus, from the analysis of natural systems, it may be possible to identify underlying mechanisms and structures to support a software engineering methodology for creating systems with complex behavior.

Self-organization is a dynamic and adaptive process where components of a system acquire and maintain information about their environment and neighbors without external control. Agent-Oriented Software Engineering (AOSE) can provide methods and technologies that help in the building self-organizing systems. A fundamental engineer-

ing issue when designing self-organizing emergent multi-agent systems (MASs) is to achieve required macroscopic properties by manipulating the microscopic behavior of locally interacting agents.

Current agent-oriented methodologies are mainly focused on engineering such microscopic issues as the agents, the rules, the protocols and their interaction without explicit support for engineering the required outcome of the system. As a consequence, the required macroscopic behavior is achieved in an ad-hoc manner. We propose a bio-inspired approach consisting of a representation model that allows a systematic specification of desirable emergent macroscopic properties, which can be mapped into the behavior of individual agents, going from the macro to the micro scale. It is bio-inspired, since we derived it from earlier experiences in modeling and simulating stem cell behavior using self-organizing multi-agent systems [Gatti et al., 2007], [Gatti, 2008a], [Gatti et al., 2008].

First the paper presents some fundamental concepts to indicate why agent-oriented software engineering and agent-based simulation fit well in the domain of self-organizing systems and why a bio-inspired representation model is a good choice. The paper then presents the meta-model underlined in the representation model and both the static and dynamics models. Then, we also demonstrate how the representation model can be applied by considering a self-organized autonomic and emergent application networking case study. In the final sub-section we show why the related work falls short on designing macro properties. We present the conclusions and future research in last section.

2. Self-Organization: Definitions and Mechanisms

A fascinating self-organizing mechanism can be observed by the straight line of ants found in our gardens stretching from food sources to anthills. Taking a closer look, we found that this straight line was made from hundreds of individual industrious ants, each behaving energetically. If we next focus on the micro view (from a modeling perspective looking at the individual elements of a system), of an individual ant's behavior, it is very easy to interpret the behavior of the individual as unfocussed and chaotic. It is certainly very difficult to interpret its behavior as being purposeful when taken in isolation. It is only when we take a step back, and look at the behavior of the entire group (called the macro view), that we can observe a purposeful global system behavior. The purpose of bringing food back to the ant hill (an emergent function) emerges from the cumulative view of the behaviors and interactions of the apparently undirected individual. Somehow the sum of the local interactions of each individual, each responding only to their local environmental, produces a stable, surviving system, even though individual ants get lost or die.

Here we define

self-organization of a system as a dynamic and adaptive process where each component of a system acquires and maintains information about its environment and neighbors without external control and where the emergent system behavior may evolve or change over time.

Hence, self-organization has three main properties: evolution, emergence and adaptation. Typically, people describe "emergence" as the phenomenon where global behavior arises from the interactions between the components of the system. Examples of emergence include: global pheromone paths that arise from local path following and phero-

mone-dropping ants, the swarming movements of a flock of birds, and a traffic jam from the interactions of cars. Because of the complexity imposed by decentralization and the highly dynamic nature of the problem domain it is usually impossible to impose an initial macroscopic structure. The macroscopic behavior arises and organizes autonomously producing self-organizing emergent behavior.

An adaptive system will self-modify into different system-states in order to navigate, function and succeed within different environments. The development of self-adaptive systems can be viewed from two perspectives, either top-down when considering an individual system, or bottom-up when considering self-organizing systems. A self-organizing system is expected to cope with the presence of perturbations and change autonomously to maintain its organization.

Finally, evolution is a consequence of emergence and adaptation in self-organizing systems. Constituent parts or components and behaviors can appear and disappear when needed.

In multi-scale self-organization architecture, emergent function or properties emerge from micro scale agent interactions in several layers resulting in a complex self-organized system. The emergent function might be a desirable behavior or an undesirable behavior (or misbehavior) in that the self-organization might emerge to a robust state or a failure state. For instance, in the ants foraging, the emergent property is the pheromone path. The ants are not aware of this property although they produce it.

A multi-agent approach contains the definition of entities or components composing a system and of the interaction among them. This approach has proven to be appropriate to simulate systems characterized by emergent properties for which basic component behaviors are known while analyzing overall phenomena derived from the interactions among them, under particular starting conditions. From this architecture we can depict the heterogeneity property where agents can interact with emergent functions being represented by emergent components composed of agents which occur in biological systems as cells. In such systems each cell is a self-organized autonomous component that emerges from molecular interactions and also interacts not only with other cells but with other molecules at the environment level.

The complexity of self-organizing emergent systems usually arises from our inability to reduce the global properties to a combination of local behaviors. Hierarchies are present in a system when multiple nested self-organized levels can be observed. Living systems accomplish self-organization repeatedly across a vast range of length scales through hierarchical organization.

Successful biological systems have been able to utilize the available (self-assembled) biological components at any particular length scale, together with the available forces and transport mechanisms, to develop new and distinct self-organization processes at a larger length scale. This hierarchical organization is evident in the animal kingdom through the assembly of: proteins from amino acids; cells from proteins and other macromolecules; tissues and organs from cells; organisms from tissues and organs; social communities of organisms from individual organisms.

3. A Bio-inspired Emergent-based Representation Model

Not only are biological systems an excellent application area for multi-agent systems concepts and development technologies; they also inspire models for new software phenomena (for instance, see [Kephart, 2003], [Lin et al., et al., 2005], [Bandini et al., et

al., 2004]). By inspiring we mean that it is possible to apply the knowledge obtained from the study of biological systems to contribute to innovations in the engineering of multi-agent systems.

We derived the representation model proposed here from earlier experiences on the development of stem cell computational modeling using self-organizing multi-agent systems [Gatti et al., 2007], [Gatti, 2008a], [Gatti et al., 2008]. During the research, we noticed a lack of design support in the literature for modeling the micro scale from the macro scale and for the design of dynamic adaptive behaviors. From the bio-inspired point of view, each cell is a self-organized autonomous component that emerges from molecular interactions and also interacts not only with other cells but with other molecules at the environment level.

The inspiration comes from two main characteristics concluded from the stem cell self-organization: the environment role in the process and the relation between states and action while the emergent properties appear. The former seems to show us that the environment behavior is explicitly coordinated with the local parts during the self-organization. And the latter shows us the interplay between (agent and environment) behaviors states and actions that allows designing emergent properties and focusing the creative effort on the enabler of self-organization. Those design issues were specifically concluded from the cell life cycle and signaling pathways. Those intracellular processes, which are emergent properties, drive the cellular proliferation and differentiation in a multi-scale architecture.

4. The Representation Model

Meta-model

The design consists of a meta-model to structure entities and a representation model which adapts UML 2 diagrams [UML, 2008], [Fowler, 2004], [Booch et al., et al., 2005] to support the mapping between macro and micro scale.

There are two main early activities that must be done before using the representation model: the textual definition of the self-organizing global behavior which might be composed of self-organizing mechanisms patterns [Gardelli, 2007], [Mameia et al., 2006], [DeWolf, 2007b] and the description of the desired emergent properties that compose the global behavior and associated patterns. During the design, the modeling consists of: defining the structural entities defined on top of the proposed meta-model, and designing the local behaviors relative to the self-organizing patterns.

The goal of the meta-model proposed in this paper is to provide a foundation for self-organizing agent-based software engineering on top of a known and trusted meta-model for software development. Therefore our meta-model was based on the UML 2 meta-model. The Unified Modeling Language (UML) has a long history and is the result of a standardization effort based on different modeling languages (such as Entity-Relationship-Diagrams, the Booch-Notation, OMT, OOSE). The most popular versions of UML are UML 1.x, but during the last four years UML 2.0 has been gradually replacing UML 1.x. The representation model proposed reuses all the notation and graphical notation in UML and extends the UML meta-model in order to provide the foundations.

We consider the environment as an explicit part of multi-agent systems, considering both the environment and the agents as *first-order abstractions*. The rationale for making the environment a first-order abstraction in multi-agent systems is presented in

[Weyns et al., 2007]. Although this discussion is not directed to self-organizing systems, we have seen in Section 2 and 3 the importance of the environment role in self-organizing systems. In this situation, the environment has a dual role: it provides the surrounding conditions for agents to exist, which implies that the environment is an essential part of every multi-agent system, and the environment provides an exploitable design abstraction to build multi-agent system applications [Weyns et al., 2007]. This second role will be explicitly addressed in the dynamic representation model presented in the next subsection.

An important issue in self-organizing systems is the global state. The global state is composed of the environment state and agent state. The environment state in turn is composed of all agent states, since if one agent leaves the environment or moves itself, the environment state changes. If we want to design the global state, we must consider explicitly the environment state in our modeling.

Moreover, the environment provides the conditions under which agents exist and it mediates both the interaction among agents and their access to resources. Moreover, the environment is locally observable to agents and if multiple environments exist, an agent can only exist in one environment at a time. In self-organizing systems, the environment acts autonomously with adaptive behavior just like agents and interacts by means of reaction or through the propagation of events.

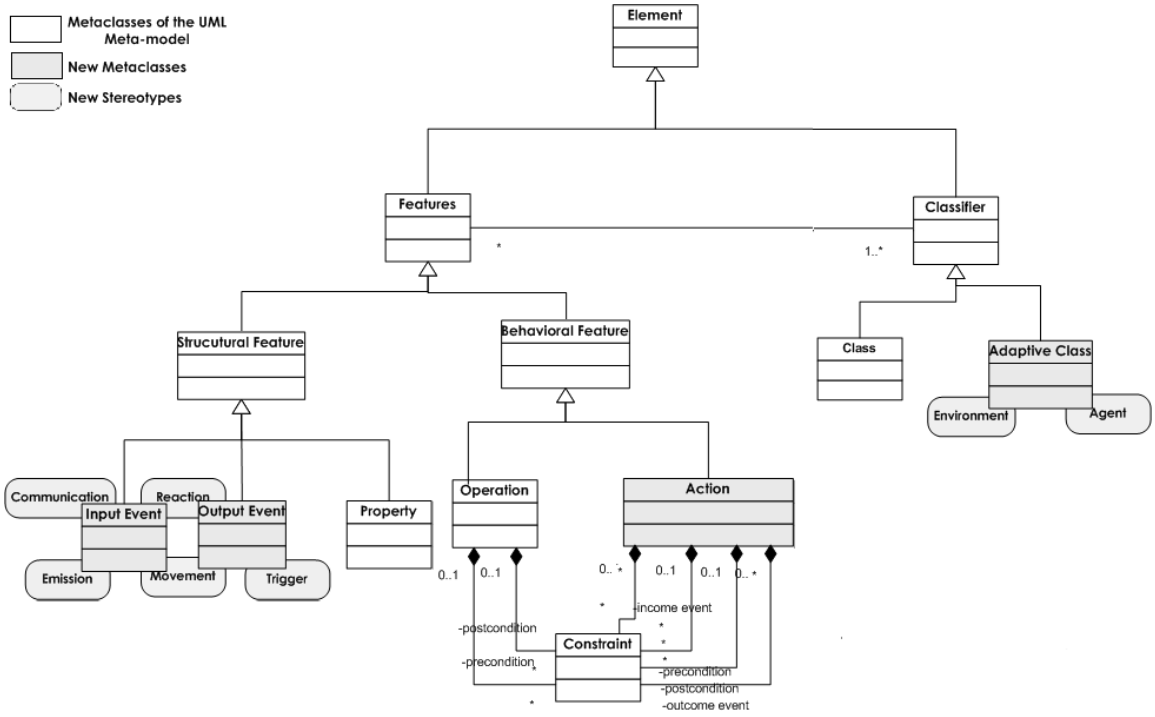

Figure 1. The meta-model proposed

We classify the events as: (i) emission: signal an asynchronous interaction among agents and their environment. Broadcasting can be performed through emissions; (ii) trigger: signal a change of agent state as a consequence of a perceived event. For instance, an agent can raise a trigger event when perceiving an emission event which changed its state; (iii) movement: signal an agent movement across the environment; (iv) reaction: signal a synchronous interaction among agents, however without an explicit receiver. It can be a neighbor of the agent or the environment; and (v) communication: signal a message exchange between agents with explicit receivers (one or more). Note that when an event is an emission the agent does not wait for a response, while in a

communication it may wait. Each of those events may be raised by actions performed by agents or by the environment and updates their states. Furthermore, the messages exchanged through communication events are interaction protocols and should be FIPA [FIPA, 2008] compliant so they can be understood by the receiver agents.

Fig. 1 shows the new meta-classes and the new stereotypes that have been proposed. The icons that represent the stereotypes are associated with the meta-classes on which the stereotypes are based. In UML, a class may be designated as active (i.e., each of its instances having its own thread of control) or passive (i.e., each of its instances executing within the context of some other object) through the Boolean attribute *isActive*. If true, then the owning class is referred to as an active class. If false, then such a class is referred to as a passive class. Default value is false. Since an active object has a different meaning than an agent [Silva, 2004], [Silva, 2007] we have defined the *Adaptive* class for classifying agents and environments. The next sub-section provides more detail about the meta-classes with the proposed representation model.

Static Model: Class Diagram

We propose to reuse the UML Class Diagram to represent the static model presented in this paper. A *Class* Diagram describes both a data model, i.e. collection of declarative (static) model elements, like classes and types, and its contents and relationships. Moreover, the static structure of the system to be developed and all relevant structure dependencies and data types can be modeled with class diagrams.

We defined the meta-class *Adaptive,* which must be stereotyped either as *agent* or *environment*. The Adaptive class holds for all agent common definitions (for instance, see [Jennings, 2000], MAS-ML [Silva, 2004], [Silva, 2007], AUML [Bauer, 2001], Anote [Choren, 2005]). It can be either a proactive or reactive agent with sensors and effectors. An agent can execute several actions regarding its goals or perceptions. As well, the environment has the same features as its autonomous behavior, mentioned before in this paper. However, it must be clear that an emergent property or macro property must not be defined as agents goals: they must emerge from the several agents and environment interactions.

We are not addressing the agent local interactions through protocols and messages in this work since there has been a huge effort from the agent research community about inter-agent communication. For instance, MAS-ML or AUML protocols diagrams can be combined with the work proposed here for designing interaction protocols.

We have two new structural features, the *input* and *output event*, and one new behavior feature, the *action*. The former define the events that the agent or environment perceives (input event) and that is a condition for activating an action, and the events that they generate (output event) after executing the action. They vary according to the stereotypes and can be stereotyped according to our previous classification (last page, last paragraph).

An *action* is executed during agent or environment execution without explicitly being called by other objects or agents. Agents interact with one another and the environment sending and receiving messages or sending and receiving events. Regarding objects, an operation can be implemented by a method that can be called either by itself or by another object. In our model, an operation can be implemented through a method that can be called through actions execution by the action owner. Moreover, the UML Action meta-class was not used to represent agents and environment actions, since it does not extend *BehavioralFeature*, so cannot be described as a Classifier characteristic.

Every action is described specifying preconditions and effects. Like the *Operation* meta-class, the *Action* meta-class is associated with the *Constraints* meta-class in order to define the *pre* and *post conditions* and the *in* and *output events*. Constraints are conditions expressed in natural language text or in a machine legible language in order to declare an entity's semantics [UML, 2008].

Dynamic Model: State-Action-based Behavior Communication

Our dynamic model is based on the UML State Machine diagram. It combines state charts [Harel, 1988]) with action and interaction overview diagram [UML, 2008], [Fowler, 2004], [Booch et al., 2005]. We did not use activity modeling from Activity diagrams. Activity modeling emphasizes the sequence and conditions for coordinating lower-level behaviors. These behaviors are commonly called control flow and object flow models. The actions coordinated by activity models can be initiated because other actions finish executing, as objects and data become available, or because events occur external to the flow. They certainly play a key role in modeling self-organizing systems because they allow the information flow modeling to achieve coordination [DeWolf, 2007]; they can also be used as a complementary design activity. However we need to model the global state, the environment state. Recall that the macro properties are defined through emergent properties which materialize from a set of local and global states during local behaviors. Thus, we need to adapt the State Machine Diagram so it can be useful for our need.

A State Machine Diagram describes discrete behavior modeled through finite state-transition systems. UML 2.0 distinguishes behavioral state machines, i.e. state machines can be used to specify behavior of various model elements. For example, they can be used to model the behavior of individual entities (e.g., class instances). The state machine formalism described is an object based variant of Harel state charts; and Protocol State machines, i.e. Protocol state machines are used to express usage protocols. Protocol state machines express the legal transitions that a classifier can trigger. The state machine notation is a convenient way to define a lifecycle for objects, or an order of the invocation of its operation. Protocol state machines do not preclude any specific behavioral implementation and enforce legal usage scenarios of classifiers.

More specifically, our dynamic model reuses the UML 2 behavioral state machine. Each agent and environment behavior is designed using behavioral state machine diagrams. Each behavioral state machine diagram can communicate with all the other diagrams through a communication channel and the desired emergent property may appear as a result of those communications. Moreover, behaviors are composed of actions. Actions are executed through input events and pre-conditions and raise output events.

We need to compose behaviors in parallel. The most noticeable concepts of statecharts are *hierarchy* and *orthogonality* [Harel, 1988]. The hierarchy concept allows hierarchical state decomposition, i.e., nesting of states within states. The outer enclosing state is called a superstate, and the inner states are called substates. The problem is that the parent state is always in a single child state. Another important concept, called orthogonality, allows a state on one statechart to be decomposed into two or more concurrent and independent orthogonal regions. Each of the orthogonal regions is named and operates independently of the other regions, and the state of the entire machine or enclosing superstate is represented by a combination of active states of the orthogonal regions. For instance, if a statechart consists of two, X and Y, orthogonal regions. When an event occurs, it is transferred to both orthogonal regions X and Y simultaneously,

resulting in the two final states for each region. It provides little concurrency support when dealing with concurrent real-time tasks, in this case, agent behaviors.

Within the context of state machines, we define a behavior as a particular instance of the agent or environment in a scenario that represents a typical path through the state space within a single state machine, i.e., an ordered sequence of state transitions triggered by events and accompanied by actions. A notable problem with a traditional state machine is that it allows only a single behavior within itself at a time, thus executing concurrent behaviors (for each agent/ environment) only in a sequential manner on a behavior-by-behavior basis, not on an event-by-event basis. The result can be poor real-time performance and underutilization of system resources such as CPU and network bandwidth.

The rationale behind our approach is that it can interleave the execution of concurrent behaviors by switching among behaviors on an event-by-event basis. As well, the self-organizing mechanism may reuse all or part of local behaviors. Thus, the new dynamic model representation must be able to encapsulate behaviors in such a way that they can be reused. To characterize the macro properties, we need to represent the communication of the agents with the environment. Thus we need local behaviors communicating with environment behaviors in order to achieve a macro behavior or emergent property. Hence the semantics of state-actions models proposed by this paper to accomplish those issues is defined by the semantics of state diagrams, dynamic overview diagram and descriptions in natural languages.

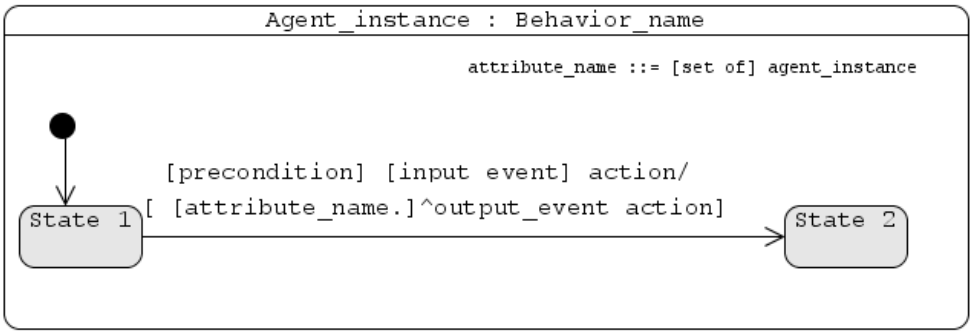

Figure 2. The abstract state-action-based behavior representation model for an agent's behavior

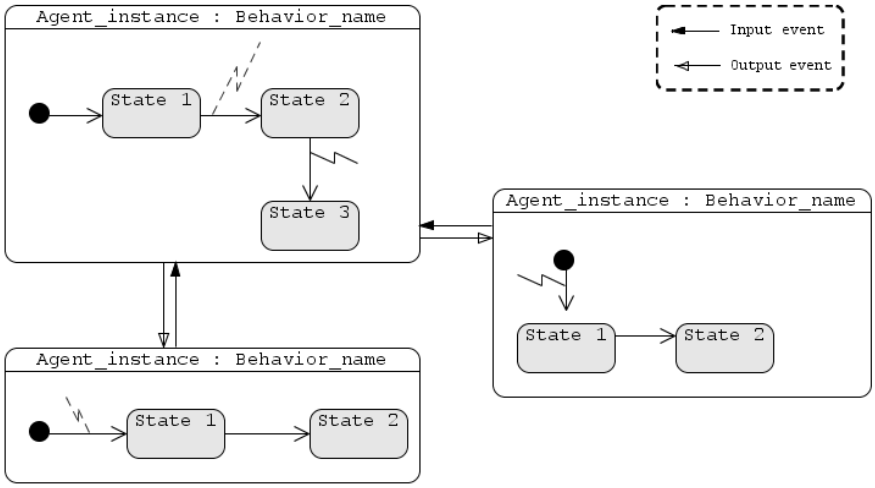

Figure 3. Abstract view of behavior communication channels and input versus output event perceptions.

Fig. 2 illustrates the abstract state-action behavior representation model for an agent's behavior. From the diagram top you have to define the instance of the agent for that behavior state. If it was an environment behavior, then the agent instance name has to be replaced by the environment instance name. Each behavior state diagram must start with a transition. The behavior state of the agent (or environment) instance may change according to a transition firing (action execution); the transition will only be fired if the agent executing the specified behavior is in State 1 and according to the input event received and the evaluated pre-condition, if specified.

The action executed might also fire an output event and might affect the agent state resulting in a post condition definition. In order to identify which entity would perceive the output event in a complex composition behavior, attributes can be defined and specified in the transition before the ^ symbol and output event to be perceived by the entity (ies).

Fig. 3 shows how behaviors can communicate through our abstract state-action representation model. The Figure shows the abstract view of behavioral communication channels and input versus output event perceptions. For instance, an output event from the upper left model from State 1 to State 2 (dashed line) may be perceived as an input event that starts at the bottom left model and takes it to State 1. The arrows between behaviors' state models show how they communicate among themselves and whether they perceive an input event or an output event. In [Gatti, 2008b] you can find instantiated descriptions about how the behavior communications works and how the parallel behavior model is accomplished.

While developing the models we realized how the emergent properties or self-organization patterns can be represented in a meso-scale through the agent and environment behavior communications. Models can be encapsulated as emergent properties achieving modularity and cohesion at that level of abstraction (Fig. 4).

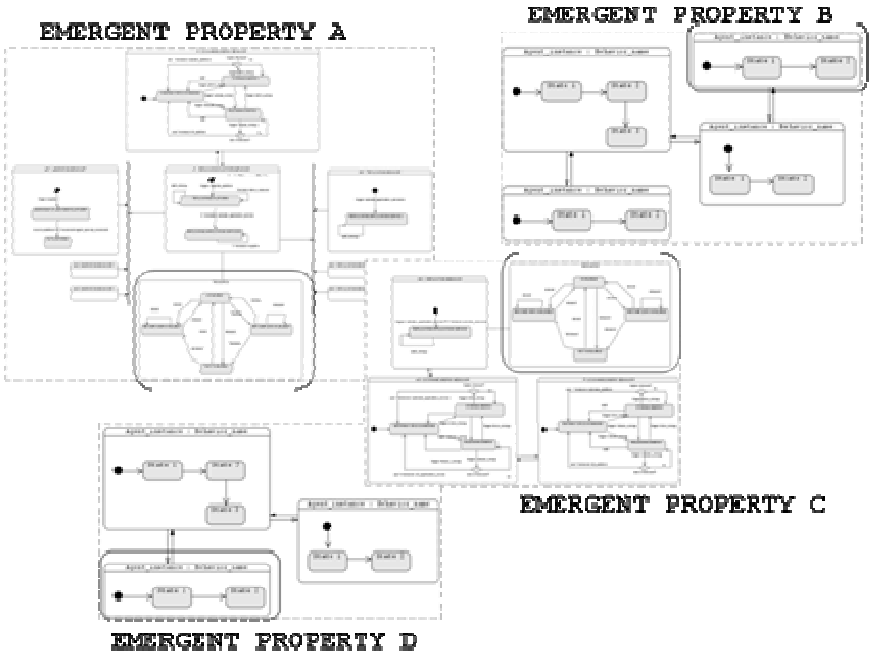

Figure 4. Emergent Properties decomposed in meso scale behaviors

In addition, there are several behavior state intersections between emergent properties. Through this modeling representation we can easily reuse them and determine how they are related to other emergent properties. For instance, in our case study we reused sev-

eral agent and environment behaviors in different emergent property models. We also noticed the presence of multiple environment static entities, as shared resources in those models. Fig. 4 illustrates these phenomena and more details can be found in [Gatti, 2008b].

Once we have defined the meso scale models, we need to model the micro scale. I.e., we need to define the local behaviors of those models. As already mentioned, if necessary our models can combine state diagrams with overview interaction diagram. The interaction overview diagram plays a key role in this process since we can use it to model for each state the local interaction between the state owner and other entities from the system, as objects, agents or environment. Hence, we use interaction overview diagrams to define interactions through a variant of static-action behavior diagrams in a way that promotes an overview of the control flow where the nodes are interactions and a link between the meso scales to the micro scale.

Case Study: The Self-Organized Autonomic and Emergent Application Networking

We present the representation model for emergent-based self-organizing multi-agent systems through a self-organized autonomic and emergent application network case study (Fig. 5). Autonomic computing is an important research area in which self-organizing emergent solutions address a specific need. Decentralized autonomic computing in [DeWolf, 2007] is achieved when a system is constructed as a group of locally interacting autonomous entities that cooperate in order to maintain the desired system-wide self-star requirements adaptively [Kephart, 2003], [IBM, 2001] without any external or central control.

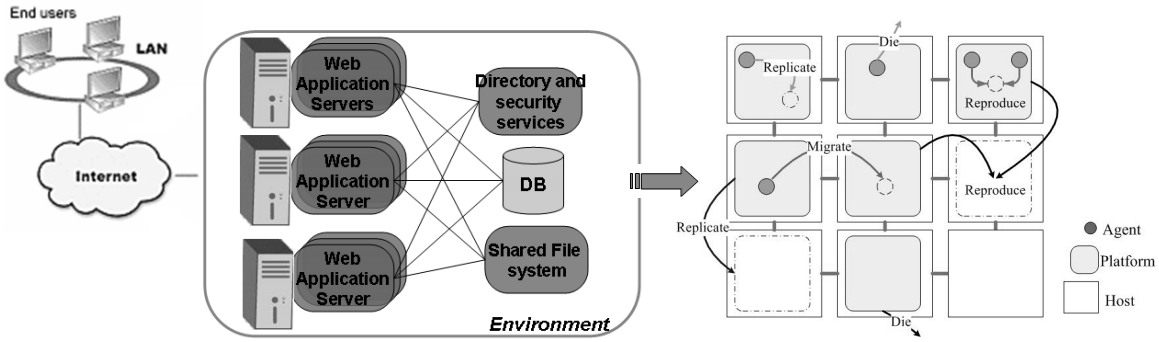

Figure 5. The autonomic application network case study, adapted from [Suzuki, 2005]

In order to apply the design approach the **first step** is to define the macro properties to be achieved. In the self-organized autonomic application networking [Suzuki, 2005], we want to achieve **two macro properties**: scalability and adaptation.

The **second step** consists of defining the underlined behavior which leads to the self-organization pattern. So, we might have each application service and middleware platform modeled as a biological entity, analogous to an individual ant in an ant colony. An application service is designed as an autonomous and distributed software agent, which implements a functional service and follows simple biological behaviors such as replication, death, migration and energy exchange (see Fig 5). In that way, agents may implement a grid application or Internet data center application on a wired network.

A middleware platform is the environment. It runs on a network host and operates agents (application services). Each platform implements a set of runtime services that

agents use to perform their services and behaviors, and follows biological behaviors such as replication, death and energy exchange. Similar to biological entities, agents and platforms in our case study store and expend energy for living. Each agent gains energy in exchange for performing its service to other agents or human users, and expends energy to use network and computing resources. Each platform gains energy in exchange for providing resources to agents, and continuously evaporates energy to the network environments.

Model agents and platforms follow several rules to determine how much energy agents/platforms expend at a time and how often they expend energy. Agents expend more energy more often when receiving more energy from users. Platforms expend more energy more often when receiving more energy from agents.

The abundance or scarcity of stored energy affects behaviors of an agent/platform. For example, an abundance of stored energy indicates higher demand for the agent/platform; thus the agent/platform may be designed to favor replication in response to higher energy level. A scarcity of stored energy (an indication of lack of demand) may cause death of the agent/platform.

Similar to biological systems, the application networking exhibits emerging desirable system characteristics such as scalability and adaptation. These characteristics emerge from collective behaviors and interactions of agents and platforms, rather than being present in any single agent/platform. We do not describe all the case study specification for space constraints, and it can be found in [Gatti, 2008b].

The **third step** was to identify the system environment(s) and agents. Figure 6 exemplifies our meta-model by showing the partial class diagram from the autonomic network case study. There you can see the Platform being classified as an *environment adaptive class*, the Application Service being classified as an *agent adaptive class* and their relationships with the resources and hosts.

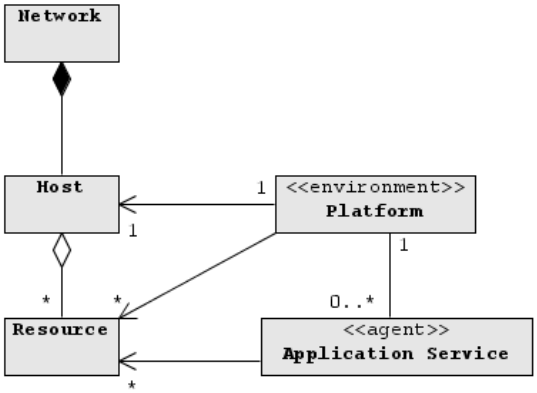

Fig. 6. The partial class diagram overview

The **fourth step** consists of designing the state-action-based behavior diagrams. We started with the exchange energy behavior since it drives all other behaviors. Figure 7 shows how the exchange energy behavior of the application service *AS1* is related to the exchange energy behavior of the platform *P1*: whenever the *AS1* stores or releases energy, *P1* also perceives and stores or releases energy. And whenever one of them is in a higher demand, they fire an event of type *emission* which will activate the respective replication behaviors.

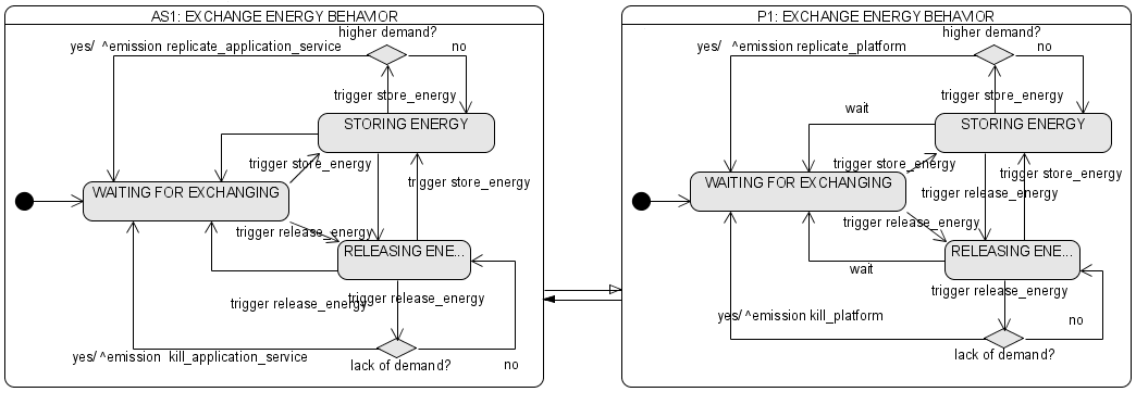

Figure 7. Exchange energy behaviors communication

Once the behavior diagrams are being built, then the **fifth step** is to update the static model. Figure 8 provides details of the input and output events and actions from the Application Service class. For instance, the *Store_ernergy()* action is executed if the agent trigger *store_energy* input event is raised by a different action and if this application service is not in a high demand state. The high demand state is defined as a pre-condition. The *Store_ernergy()* action will also emit the *store_energy* output event so it can be triggered by the Platform occupied by the agent. The post-condition for this action is that the energy of the application service must be increased by ten percent.

<<agent>>
Application Service

Input event

store_energy	<<trigger>>
release_energy	<<trigger>>
kill_application_service	<<emission>>
replicate_application_service	<<emission>>
...	...

Output event

store_energy	<<emission>>
release_energy	<<emission>>
kill_application_service	<<trigger>>
replicate_application_service	<<trigger>>
...	...

Action

kill_application_service	service_in_execution := false	Kill_application_service()	Application service uninstalled from platform	release_energy, release_resource
store_energy	higher_demand := false	Store_energy()	current_energy > previus_energy + 10%*previous_energy	store energy
...	...	...	...	...

Figure 8. The Application Service Agent Class

5. Representations and Related Work for Self-Organizing Systems

This section presents the state of the art regarding agent-oriented methodologies not-related to self-organization and the ones directly related to.

Agent-based not-related to self-organization

Current agent-oriented methodologies focus on engineering microscopic issues [Bergenti et al., 2004] e.g. the agents, their goals, their rules, action-selection, knowledge-representation, how they interact, protocols, organizations, norms, etc. As extensively argued in [Zambonelli, 2004], most current approaches to agent-oriented software engineering mainly disregard the macro scale issues and focus on development of small-size MASs (micro), and on the definition of suitable models, methodologies and tools for these kinds of systems. Examples include: Gaia v.2 [Zambonelli, 2003], Anote [Choren, 2005], MaSE [Wood, 2001], Tropos [Giorgini, 2003], MAS-ML [Silva, 2004], [Silva, 2007], AUML [Bauer, 2001], and ADELFE [Bernon et al., 2003].

ADELPHE, in particular, argues that they provide a methodology for adaptive multi-agents and self-organization. However, it simplified the concept of self-organization to cooperation and it does not address macro properties. They address non-cooperative behaviors designed at local or agent level and the agents are aware of the emergent behavior which does not really happen as already discussed in Section 2.

Related self-organizing system research

Van Parunak and Bruckner proposed a design guide for swarming systems engineering [Parunak, 2004] consisting of ten design principles that are mainly based on interactions through the exchange of information between coupled agents or processes), autocatalysis and functional adjustment (the self-organizing system must produce functions that are useful to the system's stakeholders). The ten given principles are very general and no associated tools exist. No representation model is given which helps to map the design from micro scale to macro scale.

Luca Gardelli [Viroli et Al, 2006], [Gardelli et al, 2005a], [Gardelli et al, 2005b], [Gardelli et al, 2006] proposed a meta-model and a methodological approach for engineering self-organizing multi-agent systems. The meta-model is based on stigmergy (indirect communication where individual parts communicate with one another only by modifying their local environment). He developed the engineering framework on top of the TuCSoN agent coordination infrastructure and used Pi-Calculus for specifying and verifying the system. Although the use of formal tools allows us to gain a deeper insight in emergence and self-organization, there is a general belief and proof that emergent systems cannot be specified formally [Wegner, 1997], [DeWolf, 2007d]. There is also a gap between the emergent (macro) properties defined through self-organizing design patterns from the model itself. The meta-model allows the design of only one scale.

De Wolf [DeWolf, 2007a SASO] has proposed "Information flow" as a design abstraction (by using the UML 2.0 Activity Diagram) for designing a solution independent of the coordination mechanism. We find two main problems as a sufficient design abstraction for this type of system. First, a very complex system would require very complex information flows at the macro scale. DeWolf does not propose any way to modularize, compose or even reuse the models, which makes the design approach hard to understand. As well the macroscopic information flows are based on information, not states. However, the macro properties are usually related to an optimum or desired system state or to avoid undesired system misbehavior.

6. Conclusions and Future Work

In contrast with current agent-oriented methodologies, which mainly focus on engineering such microscopic issues, our focus is on explicit support for engineering the re-

quired outcome of the system and the adjustment of the emergence properties towards convergence.

We proposed a bio-inspired approach consisting of a representation model that allows a systematic design of desirable macroscopic properties. The inspiration comes from the fact that in biological systems the environment behavior is explicitly coordinated with the local parts during the self-organization. And the relation between states and action while the emergent properties appear shows us the interplay between (agent and environment) behaviors states and actions that allows designing emergent properties and focusing the creative effort on the enabler of self-organization.

Regarding the representation model, we proposed a multiple scale design from macro to micro properties. To accomplish this design we proposed a UML-based meta-model on which we put forward the environment as an explicit part of the multi-agent system, considering both the environment and the agents as a first-order abstractions. We believe that the global state is composed of the environment state and agent state together. We added new meta-classes and the new stereotypes to the UML meta-model.

We also described the static and dynamic representation model. Our dynamic model reuses the UML 2 behavioral state machines. We represented the emergent properties or self-organization patterns at a meso-scale through the agent and environment behavior communications. In our approach, the models can be encapsulated as emergent properties achieving modularity and cohesion at that level of abstraction.

In addition, behaviors' state intersections between emergent properties can be easily identified through our approach. The modeling representation can help support further research on how to modularize aspects of emergent properties, as well as on how to reuse or combine them. In addition, we proposed to use interaction overview diagrams to define interactions through a variant of static-action behavior diagrams in a way that provides an overview of the control flow where the nodes are interactions and a link between the meso-scale and the micro scale.

For both case studies (the stem cell project and the autonomic network) the representation model proved its adequacy. We are currently also applying the proposed work in this paper in two more self-organizing case studies in industrial and market-based applications.

References

[Bandini, 2007] Bandini, S., Celada, F., Manzoni, S., Vizzari, G.; Modelling the immune system: the case of situated cellular agents. Natural Computing 6(1): 19-32 (2007).

[Bergenti, 2004] Bergenti, F., Gleizes, M.-P., Zambonelli, F. (Eds.). Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, volume 11. Springer-Verlag, 2004.

[Bauer, 2001] Bauer, B., Müller, J. P., Odell, J.; Agent UML: A Formalism for Specifying Multiagent Interaction. Agent-Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103, 2001.

[Bernon, 2003] Bernon, C. et al.; `ADELFE: A methodology for adaptive multiagent systems engineering'. In P. Petta, R. Tolksdorf, & F. Zambonelli (eds.), Engineering Societies in the Agents World III, vol. 2577 of Lecture Notes in Computer Science, pp. 156--169, Madrid, Spain. Springer, Berlin, Heidelberg, Germany, 2003.

[Booch et al., 2005] Booch, G., Rumbaugh, J., Jacobson, I.; Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Obj. Tech. Series). 2005.

[Choren, 2005] Choren, R., Lucena, C. Modeling Multi-agent Systems with ANote. Software Systems Modeling Journal 4, 2005, p. 199-208.
[Denning, 2007] Denning, P. J. and Denning, P. J. 2007. Computing is a natural science. Commun. ACM 50, 7 (Jul. 2007), 13-18.
[DeWolf, 2007a] De Wolf, T. and Holvoet, T.; Using UML 2 Activity Diagrams to Design Information Flows and Feedback-loops in Self-Organising Emergent Systems, Proc. of the Sec. Int. Workshop on Eng. Emergence in Decentralised Autonomic Systems (EEDAS 2007), pp 52-61, Editors: Tom De Wolf et al., Publisher: CMS Press, University of Greenwich, UK, Co-located with ICAC 2007, Florida, USA, 2007.
[DeWolf, 2007b] De Wolf, T. and Holvoet, T.; Design Patterns for Decentralised Coordination in Self-organising Emergent Systems, Editors: Sven Brueckner, Salima Hassas, Màrk Jelasity and Daniel Yamins, Eng. Self-Organising Systems: Fourth Int. Workshop, ESOA 2006, Future University-Hakodate, Japan, 2006, Revised Selected Papers, Lecture Notes in Computer Science, Volume 4335, 2007, pp. 28–49, Springer Verlag
[DeWolf, 2007c] De Wolf, T. and Holvoet, T.; Designing Self-Organising Emergent Systems based on Information Flows and Feedback-loops. Proc. of the First IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems (SASO), Editors: Di Marzo Serugendo et al., MIT, Boston, USA, pp 295-298, ISBN 0-7695-2906-2, July 9-11, 2007.
[DeWolf, 2007d] De Wolf, T.; Analysing and engineering self-organising emergent applications, Ph.D. Thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, May, 2007, 183
[FIPA, 2008] FIPA - Foundation for Intelligent Physical Agents. http://www.fipa.org.
[Fowler, 2004] Fowler, M.; UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition) (The Addison-Wesley Object Technology Series), 2004.
[Gardelli et al., 2005] Gardelli, L., Viroli, M., Omicini, A.; On the Role of Simulations in Engineering Self-Organizing MAS: the Case of na Intrusion Detection System in TuCSoN. Lecture Notes in Computer Science. LNAI 3910. pp 153-166. Springer, 2006 3rd Int. Workshop "Eng. Self- Organising Applications"(ESOA 2005), Utrecht, The Netherlands (NL), July 2005.
[Gardelli et al., 2006a] Gardelli, L., Viroli, M., Casadei, M.; Engineering the Environment of Self-OrganisingMulti-Agent Systems Exploiting Formal Analysis Tools. AICA 2006 – Assoc. Italianaper l'Informaticae ilCalcolo Automatico. Sep. 2006, Cesena-Italy
[Gardelli et al., 2006b] Gardelli, L., Viroli, M., Casadei, M.; On Engineering Self-Organizing Environments: Stochastic Methods for Dynamic Resource Allocation. The Third Int. Workshop on Env. for Multiagent Systems (E4MAS 06). In: The Fifth Int. Joint Conf. on AAMAS, 2006 – Future University-Hakodate, Japan, May 8, 2006.
[Gardelli, 2007] Gardelli, L., Viroli, M., Omicini; A.; Design Patterns for Self-Organizing Multiagent Systems. 2nd Int. Workshop on Eng. Emergence in Decentralised Autonomic Systems (EEDAS 2007). To be held at the 4th IEEE Int. Conf. on Autonomic Computing (ICAC 2007). June 11th, 2007, Jacksonville, Florida, USA.
[Gatti et al., 2007] Gatti, M., de Vasconcellos, J.E., Lucena, C.J.P.; An Agent Oriented Software Engineering Approach for the Adult Stem-Cell Modeling, Simulation and Visualization. Workshop SEAS, João Pessoa, 2007.
[Gatti, 2008a] Gatti, M.A. de C., Lucena, C.J.P.; Cell Simulation: An Agent-based Software Engineering Approach. *Submitted to* The IEEE Transactions on Software Engineering Journal, February, 2008.
[Gatti, 2008b]. Gatti, M.A. de C., Lucena, C.J.P.; Self-Organization and Emergent Behavior in Multi-Agents Systems: a Bio-inspired Method and Representation Model. *Submitted to* The Science of Computing Programming, March, 2008.
[Gatti et al., 2008] Gatti, M.A. de C., Faustino, G.M., Bispo, D., de Vasconcellos, J.E., Lucena, C.J.P.; Agent-Oriented Stem Cell Computational Modeling. SEMISH 2008 -

XXXV Seminário Integrado de Software e Hardware. Grandes Desafios Da Computação No Brasil, SBC 2008 - XXVIII Congresso da Sociedade Brasileira de Computação, Belém, 2008.

[Giorgini, 2003] Giorgini, P., Kolp, M., Mylopoulos, J. & Pistore, M.; The Tropos Methodology: An Overview. In F. Bergenti,M.-P. Gleizes & F. Zambonelli, editeurs, Methodologies And Software Engineering For Agent Systems. Kluwer Academic Publishing, New York, December 2003.

[Harel, 1988] Harel, D.; On visual formalisms. Communications of the ACM, V31 I5 pp514-530, 1988.

[IBM, 2001] Autonomic Computing: IBMs Perspective on the State of Information Technology. Rapport technique, IBM research, 2001.

[Jennings, 2000] Jennings, N., R. (2000). On Agent-Based Software Engineering. Artificial Intelligence Journal, 117 (2) 277-296, 2000.

[Kephart, 2003] Kephart, J. and Chess, D. M.; The Vision of Autonomic Computing. IEEE Computer Magazine 36(1): 41-50, 2003.

[Lin, 2005] Lin, P., MacArthur, A., Leaney, J.; Defining Autonomic Computing: A Software Engineering Perspective. IEEE. Proc. of the 2005 Australian Soft. Eng. Conf. (ASWEC'05).

[Mameia et al., 2006] Mameia, M., Menezesb, R., Tolksdorfc, R. and Zambonelli, F.; Case Studies for Self-Organization in Computer Science. Case studies for self-organization in computer science. J. Syst. Archit. 52, 8 (Aug. 2006), 443-460, 2006.

[Parunak, 2004] Parunak, H. V. D. & Brueckner, S.. Engineering Swarming Systems. In F. Bergenti, M.-P. Gleizes & F. Zambonelli, editeurs, Methodologies and Software Engineering for Agent Systems, pages 341–376. Kluwer, 2004.

[Silva, 2004] Silva, V. and Lucena, C. (2004). From a conceptual framework for agents and objects to a multi-agent system modeling language. Journal of AAMAS, 9, 1--2, 145—189.

[Silva, 2007] Silva, V. T.; Lucena, C. J. P. De; Modeling Multi-Agent System. In Communication of ACM (CACM), vol. 50, no. 5, Maio 2007, pp 103-108.

[Suzuki, 2005] Suzuki, J.; Biologically-inspired Adaptation of Autonomic Network Applications. In International Journal of Parallel, Emergent and Distributed Computing, Vol. 20, No. 2, pp. 127-146, Taylor & Francis, June 2005.

[UML, 2008] UML 2.x OMG Specification. http://www.omg.org/

[Viroli, 2006] Viroli, M., Casadei, M., Gardelli L.; A Case of Self-Organising Environment for MAS: the Collective Sort Problem. 4th European Workshop on Multi-Agent Systems (EUMAS 2006). December 2006 -Lisbon –Portugal.

[Wegner, 1997] Wegner, P.; Why Interaction is More Powerful than Algorithms. Communications of the ACM, vol. 40, no. 5, pages 80–91, May 1997.

[Weyns et al., 2007] Weyns, D., Omicini, A. Odell, J.; Environment, First-Order Abstraction in Multiagent Systems. Autonomous Agents and Multi-Agent Systems, v.14 n.1, p.5-30, February 2007.

[Wood, 2001] Wood, M. F. & DeLoach, S. A.; An Overview of the Multiagent Systems Eng. Methodology. In Agent-Oriented Soft. Eng., vol. 1957 of LNCS. Springer, 2001.

[Zambonelli, 2004] Zambonelli, F. & Omicini, A.; Challenges and Research Directions in Agent-Oriented Software Engineering. Autonomous Agents and Multi-Agent Systems, vol. 9, no. 3, pages 253–283, 2004.

[Zambonelli et al., 2003] Zambonelli, F., Jennings, N. & Wooldridge, M.. Developing multiagent systems: The Gaia methodology. ACM Trans. Software Eng. Meth., vol. 12, no. 3, pages 417–470, 2003.