ISSN 0103-9741

Monografias em Ciência da Computação
n° 19/08

Self-Organization and Emergent Behavior in Multi-Agents Systems: a Bio-inspired Method and Representation Model

Maíra Athanázio de Cerqueira Gatti
Carlos José Pereira de Lucena
Paulo S. C. Alencar
Donald D. Cowan

Departamento de Informática

Monografias em Ciência da Computação, No. 19/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena April, 2008

Self-Organization and Emergent Behavior in Multi-Agents Systems: a Bio-inspired Method and Representation Model

Maíra Athanázio de Cerqueira Gatti[1], Carlos José Pereira de Lucena[1], Paulo S. C. Alencar[2], and Donald D. Cowan[2]

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

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

[2] Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

{palencar,dcowang}@csg.uwaterloo.ca

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. 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. 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. The novel emergence property adjustment requires a discrete agent-based model to interface with the continuous simulation tools and interact with them as they move towards convergence. In contrast with current approaches, we propose a bio-inspired approach consisting of a method that allows a systematic specification (i.e., a representation model) of desirable macroscopic properties, which can be mapped into the behavior of individual agents, followed by development of a system, and the interactive adjustment of the required emergent macroscopic properties that need to be achieved.

Keywords: Self-Organization, Multi-agent Systems, Autonomic Computing, Software Engineering, Verification.

Resumo. Auto-organização é um processo adaptativo e dinâmico onde componentes de um sistema adquire e mantém informação sobre o ambiente e seus vizinhos sem controle externo. 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. As metodologias orientadas a agentes propostas até o momento são principalmente focadas na engenharia de tais propriedades microscópicas tais como os agentes, as regras, os protocolos de interação sem um apoio específico para a engenharia do sistema resultante. O ajuste de uma propriedade emergente requer um modelo discreto baseado em agentes para fazer a interface entre ferramentas de simulação contínua e para interagir com as mesmas a fim de obter convergência. Em contraste com as abordagens existentes, propomos uma abordagem inspirada na biologia que consiste de um método que prove uma especificação sistemática (i.e., um modelo de representação) das propriedades macroscópicas desejadas – que pode ser mapeada no comportamento individuais dos agentes, seguido do desenvolvimento do sistema, e de um ajuste interativo das propriedades macroscópicas emergentes desejadas que precisem ser alcançadas.

Palavras-chave: Auto-Organização, Sistemas Multiagentes, Computação Autonômica, Engenharia de Software, Verificação.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br

Table of Contents

I. Introduction

Self-organization is a very powerful approach to the engineering of complex systems because of its many interesting properties, including adaptation and robustness. 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 [42]. 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. 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.

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. The emergence property adjustment requires a discrete agent-based model to interface with the continuous simulation tools and interact with them as they move from towards convergence.

In contrast to the current state-of-the-art, we propose a bio-inspired approach consisting of a method that allows a systematic specification (i.e., a representation model) of desirable macroscopic properties, which can be mapped into the behavior of individual agents, followed by development of a system, and the interactive adjustment of the required emergent macroscopic properties that need to be achieved. The method is bio-inspired since we derived it from earlier experiences in modeling and simulating stem cell behavior using self-organizing multi-agent systems [30][43][44]. The method advances the state of the art in self-organizing emergent agent systems in three ways. 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 method and representation model is a good choice. The paper then presents the method for guiding the design of self-organizing emergent agent systems, a meta-model, a representation model and a framework to support the verification phase. Then, we also demonstrate how the method can be applied by considering a self-organized autonomic and emergent application networking case study [41]. We present the conclusions and potential future research in section 7.

II. Self-Organization: Definitions and Mechanisms

During the last 5 years, there has been significant research in the field of self-organization in computer science. Several definitions and mechanisms have been examined in order to understand how computing can model self-organizing systems and how self-organizing systems can empower the computer science.

Visser et al., 2004 [45] defines self-organization as the evolution of a system into an organized form in the absence of an external supervisor, where "*A system can be defined as a group of interacting agents that is functioning as a whole and distinguishable from its surroundings by its behavior*" and "*An organization is an arrangement of selected parts so as to promote a specific function*".

Serugendo et al., 2005 [35] states that "*Self-organization is defined as the mechanism or the process enabling a system to change its organization without explicit external command during its execution time.*" Further he defines *"Strong self-organizing systems are those systems where there is no explicit central control either internal or external"*; and *"Weak self-organizing systems are those systems where, from an internal point of view, there is re-organization maybe under an internal (central) control or planning.*"

Another definition from Di Marzo Serugendo et al., 2007 is: "*Self-organisation is the mechanism or the process enabling a system to change its organization without explicit external command during its execution time*".

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.

Self-organization has three main properties: evolution, emergence and adaptation.

- Emergence: 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 pheromone-dropping ants, the swarming movements of a flock of birds, and traffic jam from the interactions of cars.

The essence of emergence is the existence of a global behavior that is novel with respect to the constituent parts of the system. The essence of self-organization is an adaptable behavior that autonomously acquires and maintains an increased order for the system (i.e. statistical complexity, structure, ...). In most systems that are considered in the literature, emergence and self-organization occur together. We call such systems "self-organizing emergent" systems. In inherently decentralized and highly dynamic problem domains the combination of emergence and self-organization is recommended [23].

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.

- Adaptation: 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.

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

Figure 1 presents the multi-scale self-organization architecture from the micro scale local component interactions to the macro scale emergent self-organization. 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.

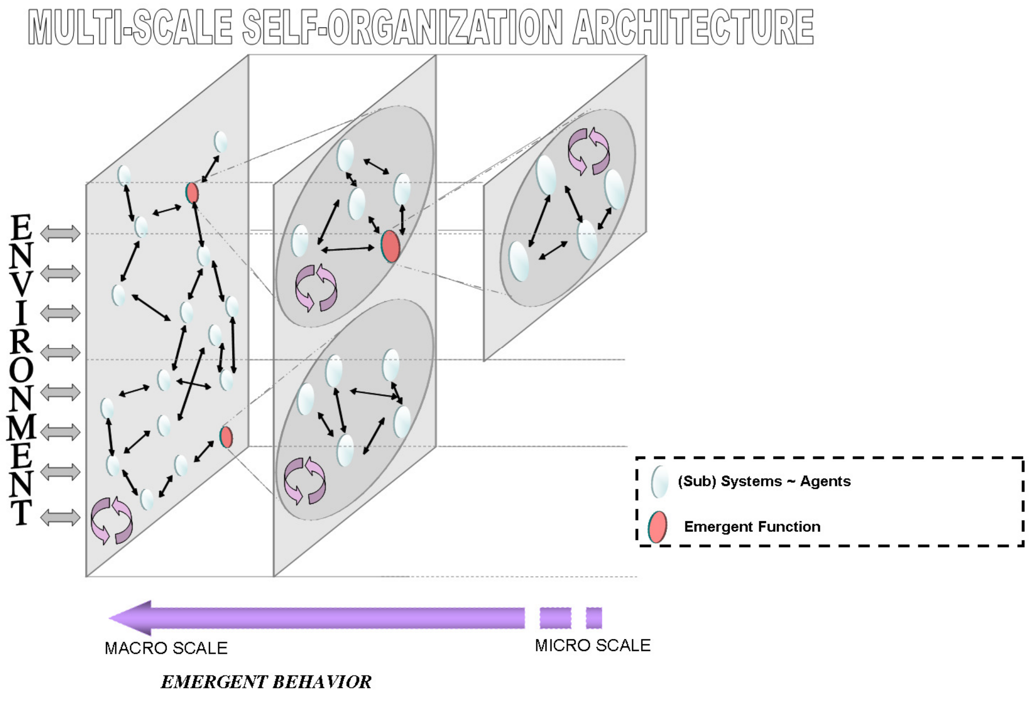


Figure 1. The agent-based multilayer self-organization architecture shows the emergent fuctions and the self-organization emerged from agents interactions through the micro scale to the macro scale.

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.

Here we summarize the assumptions related to self-organizing principles for this work:

- Order is emergent rather than pre-determined.
- A system's future is, in general, unpredictable.
- The basic entities of a complex system are agents;
- The individual agents are not aware either of the larger organization, or its goals and needs. Clearly any single agent within the system cannot know the state and current behavior of every other agent, and as a result cannot determine its behavior based on such complete global system information. Instead the behaviors of agents are governed by rules based on the local environment.
- Agents are typically comprised of reactive rules that are usually of the form of if condition then fire action. For example, a possible rule for an agent might be if there's a space next to me and I am currently too hot then move into an empty space.
- An agent can perceive aspects of its environment and can act so as to change the state of the environment. Critically, in a complex system agents must affect the environment in such a way that the change in the environment can: 1) be perceived by others; and 2) affect the behavior of others. That is the agents must have a reasonable degree of interaction beyond, for instance, simple obstacle avoidance in robot vehicle simulations.
- Agents may be equipped with an ability to adapt and learn rules so as to have a more effective way of maximizing their usefulness in given situation. New rules would try, for example, to make the agent more able to act effectively in a wider variety of environmental situations.

- Rules may compete for survival. The more a rule is used in determining behavior the greater its chance of surviving in the future. Rules that are seldom or never used will have less chance of survival. Rules may change randomly or intentionally and may be integrated for more sophisticated action.
- Agents are resource-bounded and can only perceive (or experience) their local environment;
- A few individuals or agents will not make a sustainable complex system. It is only when the number of agents reaches a certain critical threshold that the system will exhibit global, meaningful behavior.
- The information is used though two main mechanisms: trigger-based - event notification; and follow-through - the information being communicated drives the agent actions step-by-step.

III. Self-Organizing System Design Issues and Main Challenges

As software systems grow in complexity, interconnectedness, and geographic distribution, we will increasingly face unwanted emergent behavior. Unpredictable software systems are hard to debug and hard to manage. We need better tools and methods for anticipating, detecting, diagnosing, and ameliorating emergent misbehavior. These tools and methods will require research into the causes and nature of emergent misbehavior in software systems.

Emergent behavior can be beneficial, for instance, individual ants are less smart than the collection of ants known as an ant colony. But it is not always beneficial. For example, stock market panics are a form of unwanted emergent behavior in which the irrational behavior of many individual investors makes things worse for everyone.

The environment can play a key role by constraining a system, and a system needs to be able to adapt to these environmental constraints. Hence, we require a better understanding of the relationship between micro and macro levels in order to build a system able to adapt to environmental dynamics.

Thus, the first main challenge is how to make emergent behavior of a system converge to a pattern or trend, which is the self-organization goal or sub-goals? Understanding how such self-organizing components are constrained to carry out the appropriate actions at the right places and times is a key challenge.

Another main challenge is the need for tools, models and guides to develop such systems. What would be an appropriate representation model and what tools are needed to support this representation? Are existing ones sufficient? We need a tool to observe unpredictable behavior in the face of small perturbation and also a mechanism of self-adaptation in a presence of misbehavior. Others have certainly looked at the issue of emergent behavior in enterprise systems. For example, the emphasis on self-management in IBM's autonomic computing vision clearly leads to emergent behavior, as pointed out by Kephart and Chess [22], although they focused more on how to encourage emergent good behavior, rather than to detect, diagnose, or prevent emergent misbehavior.

As well self-organizing emergent MAS will only be acceptable in an industrial application if one can provide guarantees about MAS macroscopic behavior. Thus an important issue is to measure the self-organization and to validate these systems. A self-organizing system can be studied from a local or global perspective and more perspectives can be considered if self-organization spans multiple nested hierarchical levels. In all cases measures of self-organization [35] mean observed organizational structure, a process that produces and maintains that structure, or a certain function or global goal that the system aims to fulfill using self-organization.

Structure-based measurement focuses on assessing the system structure after it has stabilized following a series of changes in self-organization. Measures focusing on the self-organization process are related to the system's dynamics and its evolution over time. Finally, measurements focusing on the function that must be delivered by self-organization are related to how well the self-organizing system is able to fulfill its purpose. Therefore, measures in this case concern the characteristics of the problem the system is dedicated to solve and are similar to those used for performance assessment in classical systems.

Examples of typical self-organization measures include: capacity to reach an organization able to fulfill the goal of the system as a whole once the system is started (success/failure/time required, convergence); capacity to reach a re-organization after a perturbing event (success/failure/time required); degree of decentralized control (central/totally decentralized/hybrid); or capacity to withstand perturbations: stability/adaptability.

The global behavior verification of the system consists in determining that the system complies with the desired function. The main question is how to determine if a certain self-organizing emergent system exhibits the required macroscopic behavior. Firm guarantees could be obtained if the system is modeled formally and the required macroscopic behavior is proven analytically. However, constructing a formal model and correctness proof of a complex interacting computing system is infeasible [9]. Interaction models are so powerful that they can be considered to be incomplete, in the mathematical sense. One cannot model all possible behavior of an interaction model and thus formally proving correctness of interactive models such as self-organizing emergent systems is not merely difficult but impossible in general [46]. Therefore, we need measurements that represent these self-organizing emergent systems exhibiting trends that are predictable; by trend we understand average (system-wide) behavior.

To sum up, a fundamental problem is the lack of a step-by-step plan that supports systematic specifying specification of desirable macroscopic properties, mapping these properties onto the behavior of individual agents, building the system, and verifying that it has the required macroscopic properties.

IV. A Bio-inspired Emergent-based Method

In this section, we motivate the need for a bio-inspired emergent-based method for self-organizing multi-agent systems. 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 such as self-adaptation, self-protection, self-healing, heterogeneity, self-organization, cooperation and coordination mechanisms (for instance, see [17][18][19]). 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.

Thus, there is a need for a coherent method addressing important self-organizing engineering objective as development of novel design concepts, measurement of self-organizing systems, development of robust self-organizing engineering systems, and effective decomposition of complex self-organizing engineering problems.

We derived the method proposed here from earlier experiences on modeling and simulating stem cell behavior using self-organizing multi-agent systems [30][43][44]. During the research on agent-based conceptual model representation 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. We noticed the environment role during the developmental process and how the self-organization emergent pattern should be observed and evaluated in order to verify the system.

V. The Method and Representation Model

We will present our method and representation model for emergent-based self-organizing multi-agent systems through a self-organized autonomic and emergent application network case study. Autonomic computing is an important research area in which self-organizing emergent solutions address a specific need. Decentralized autonomic computing in [9] 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 [17][47] without any external or central control.

For instance, self-healing emphasizes the requirement to detect, diagnose, and repair problems autonomously to make the system more reliable and available (i.e. failure dynamics). Self-configuring emphasizes the requirement to handle structural changes so that the system configuration or "set-up" occurs autonomously (i.e. setup-structural dynamics). Self-optimizing requires a system never to settle for the status quo but to look continuously for ways to optimize its efficiency and performance by autonomously monitoring and tuning the system behavior (i.e. normal dynamics that signals to optimize). Self-protection emphasizes the requirement for a system to anticipate, detect, identify, and protect against malicious attacks or cascading failures autonomously to maintain overall system security and integrity (i.e. malicious dynamics).

IBM's approach to autonomic computing typically takes one autonomic manager to achieve one self-x requirement. Self-organizing emergent solutions achieve a self-x requirement through the interactive behavior of multiple agents.

Subsection 5.1 briefly details the application case study to make the paper self-contained and to support the method and representation model rationale. Subsection 5.2 presents the method itself. Then subsection 5.3 details the meta-model proposed while the subsection 5.4 details the representation model composed of structure representation and dynamic representation adapted from UML 2.0. Finally, subsection 5.5 presents the complete proposed method.

V.i. The Self-Organized Autonomic and Emergent Application Networking

In the self-organized autonomic application networking [41], we might have each application service and middleware platform modeled as a biological entity, analogous to an individual bee in a bee colony. An application service can be 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 (Fig 2). In that way, agents may implements 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.

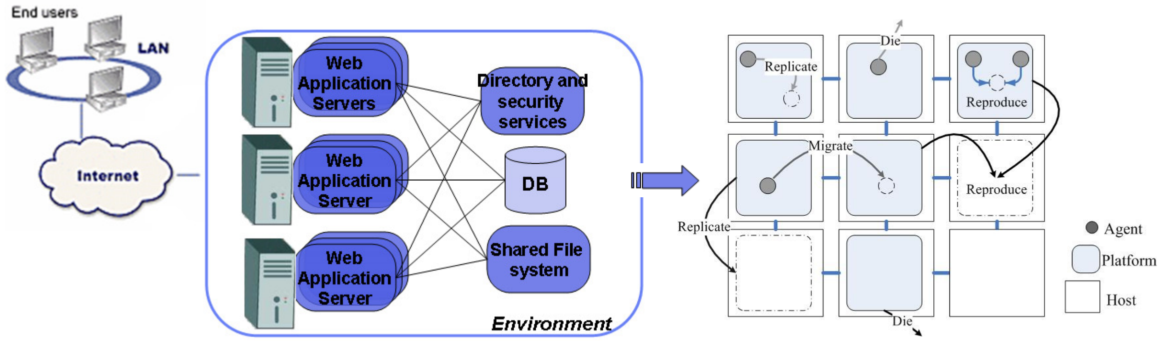


Figure 2. The bio-inspired autonomic application network case study, adapted from [41]

Simulation results show that agents and platforms autonomously scale to rapid demand changes and adapt to dynamic changes in the network (e.g. user location and resource availability). In certain circumstances, agents and platforms spontaneously cooperate in a symbiotic manner to pursue mutual benefits (i.e. to increase their scalability and adaptability), although each of them is not designed with that purpose in mind.

Agents and platforms are *decentralized*. There are no central entities to control and coordinate agents/platforms (i.e. no directory servers and no resource managers). Decentralization allows agents/platforms to be scalable and simple by avoiding a single point of performance bottleneck and by avoiding any central coordination in developing deploying agents/platforms. Agents and platforms are *autonomous***.** They monitor their local network environments, and based on the monitored environmental conditions, they autonomously behave, and interact without any intervention from/to other agents, platforms and human users. They are *adaptive* to dynamically changing environment conditions or constraints (e.g. user demands, user locations and resource availability). Adaptation is achieved through designing agent/platform behavior constraints to consider local environment conditions based on global states. For example, agents may implement a migration constraint of moving towards a platform that forwards a large number of request messages for their services. This results in the adaptation of agent locations, and agents concentrate around the users who request their services. Also, platforms may invoke replication and death behaviors when their energy levels become over and below thresholds. This results in the adaptation of platform population, and platforms adjust resource availability on them against the demands for resources.

Application service behaviors may include: migration, agents may move from one platform to another; replication, agents may make a copy of themselves as a result of abundance of energy. A replicated (child) agent is placed on the platform, on which its parent agent resides, and it receives half the amount of the parent's energy level; death, agents may die because of energy starvation. When an agent dies, an underlying platform removes the agent from the network and releases all resources allocated to that agent.

Each platform runs on a network host and operates agents. The health level is defined as a function of the age and resource availability on a network host on which the platform runs. The age indicates how long a network host remains alive (i.e. the stability of a network host). Resource availability indicates the resources (e.g. memory space) available for agents and platforms on a network host. Health level affects the behaviors of both a platform and agent. For example, a higher health level indicates higher stability and higher resource availability on a network host on which the platform resides. Thus, the platform may be designed to replicate itself on a healthier neighboring host than on the current local host. This results in the adaptation of platform locations. Platforms work on stable and resource rich network hosts.

Network platform behaviors may include: replication, platform copying as a result of energy abundance (i.e. higher demand for resources available on the platforms), providing half the amount of the parent's energy level to child platform; and death where platforms may die because of the lack of energy. A dying platform disconnects itself from the network and releases all resources the in use by the platform. Despite the death of a platform, an underlying network host remains active so that other platforms can operate on it in the future.

Runtime services are middleware services that agents and platforms use to perform their functions. In order to maximize decentralization and autonomy of agents/platforms, agents only use their local runtime services. Agents are not allowed to invoke any runtime services running on a remote platform.

The factors that constrain the agent migration behavior include:

• Service request ratio is the ratio of number of service requests on a remote platform to the number of service requests on a local platform; this ratio is used to encourage agents to move towards the local platform.

• Health level ratio is the ratio of the health level of a remote host to the health level on a local host; this ratio encourages agents to move to platforms running on healthier hosts.

• Migration interval is the interval from the time of a previous migration, which is used to discourage agents from migrating too often. If there are multiple neighboring platforms to which an agent can migrate, the agent calculates a weighted sum of the values if the previous factors for each of the platforms, and migrates to a platform that generates the highest weighted sum.

Agent replication and death behaviors use a factor that evaluates the current energy level of an agent.

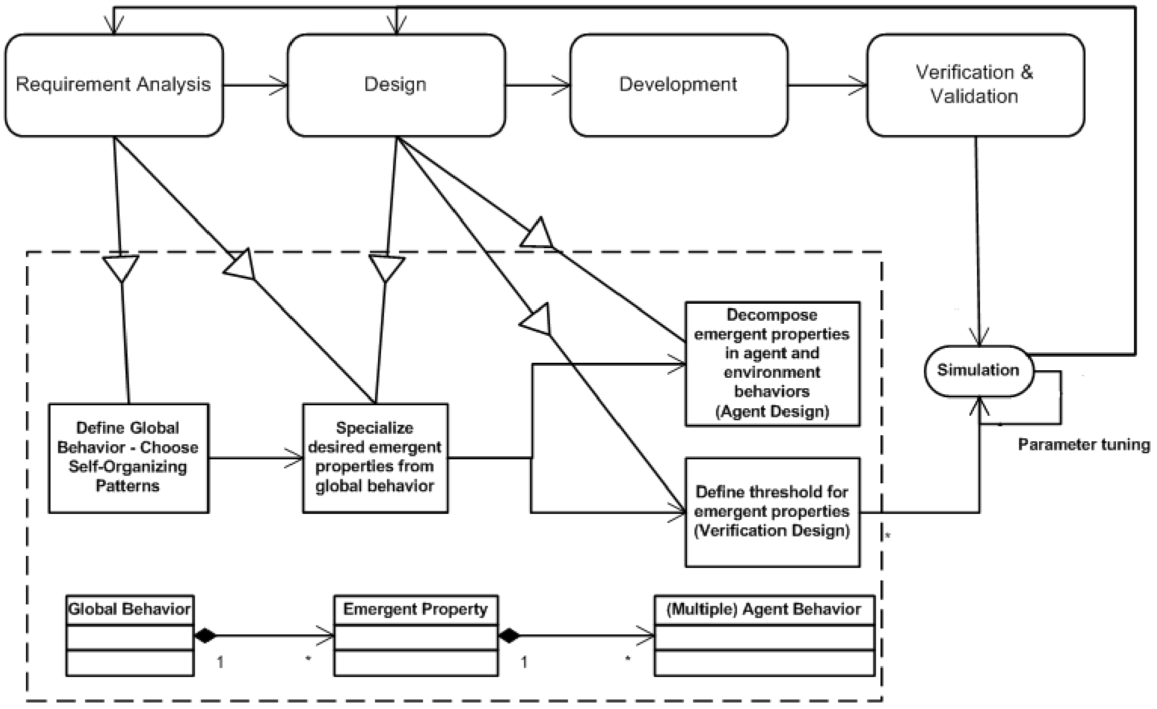
The factors affecting platform replication behavior include:
• Health Level Ratio which is ratio of the health level on a remote host to the health level on a local host; this ratio encourages platforms to replicate themselves on a healthier host.
A replicated (child) platform is placed on a host whose health level is highest among neighboring hosts.

Platform death behavior is a factor that evaluates the current energy level of platform. Each platform does not perform death behavior while an agent(s) runs on the platform.

Each agent/platform incurs energy loss (i.e. behavior cost) to invoke behaviors except death behavior. When the energy level of an agent/platform goes over the cost of a behavior, the agent/platform decides whether it performs the behavior by calculating a weighted sum of factor values.

V.ii. The Method

The method consists of a meta-model to structure entities, a representation model which adapts UML 2 diagrams [48][49][50] to support the mapping between macro and micro scale, and a method to support the verification phases.



Requirement Analysis
Design
Development
Verification & Validation
Define Global Behavior - Choose Self-Organizing Patterns
Specialize desired emergent properties from global behavior
Decompose emergent properties in agent and environment behaviors (Agent Design)
Define threshold for emergent properties (Verification Design)
Simulation
Parameter tuning
Global Behavior
Emergent Property
(Multiple) Agent Behavior
1
*
1
*
*

Figure 3. The Method Rationale

There are two main activities that must be incorporated into the requirement analysis phase: the textual definition of the self-organizing global behavior which might be composed of self-organizing mechanisms patterns [52][51], and the description of the desired emergent properties that compose the global behavior and associated patterns. At this point the software engineer does not know how the local interactions happen to make emergent properties arise. However, if the software engineer wants the self-organizing system to converge, and to verify and optimize the self-organization system in an iterative development, than those local activities are critical to the system development.

In the design phase the analyst models the emergent properties defined in the previous phase. This modeling consists of: defining the structural entities defined on the proposed meta-model presented in the next subsection; and decomposing the emergent properties into local behaviors relative to the self-organizing patterns. During the design phase, after the structural definitions, it is also necessary to define the thresholds for the emergent properties. Those thresholds will be based on the constraints specified for the global behavior and will be the input for the convergent method during the verification phase when the simulation will guide parameter tuning through an online search planner.

Considering the case study, the last subsection described the global behavior to be achieved – self-management, adaptation and scalability; and the self-organization patterns that help achieve these goals – replication, migration, death, and energy exchange. Next subsections will demonstrate how to perform the related steps.

V.iii. The Meta-model

A good procedure (if not the only viable one) for successful deployment of agent technology is to present the it as an incremental extension of known and trusted methods, and to provide powerful engineering tools that support industry-accepted methods of technology deployment. Accepted methods of industrial software development depend on standard representations for artifacts to support the analysis, specification, and design of agent software.

Thus, 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.

Some efforts **[53]** on applying and extending UML 2 to agent-based development have been investigated. UML consists of a notation describing:

1. the syntax of the modeling language and a graphical notation,
2. and a meta model for the static semantics of UML, but not providing operational semantics.

The method proposed in this paper reuses all the notation and graphical notation in UML and extends the UML meta-model in order to provide foundations for the structural definition of self-organizing multi-agent systems. Before describing the meta-model proposed, Fig. 4 shows the conceptual model that contains the new elements in the meta-model.

In this paper, 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 [54]. Although this discussion is not directed to self-organizing systems, we have seen in section 2 the importance of the environment role in self-organizing systems. In this situation, the environment has a dual role:

1. it provides the surrounding conditions for agents to exists, 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 **[54]**. This second role will be explicitly addressed in the dynamic representation model presented in the next subsection.

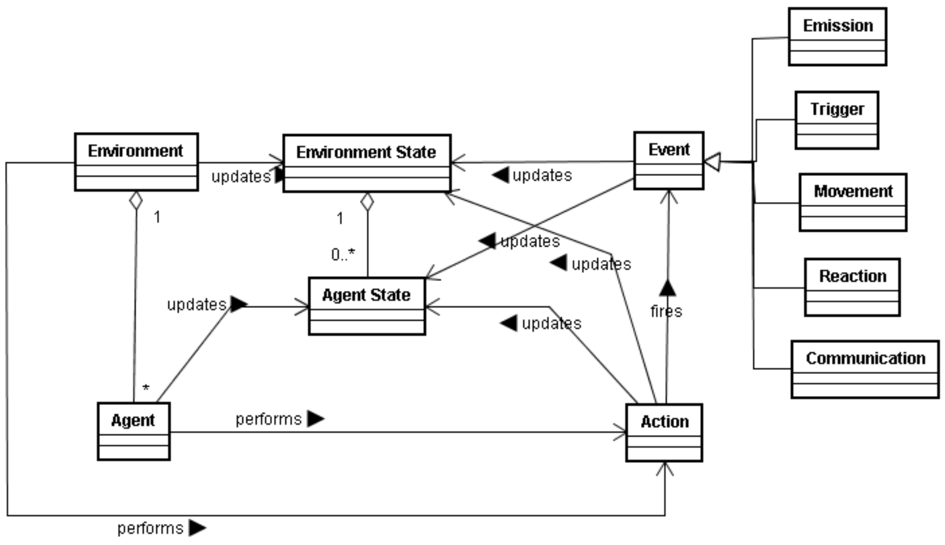


Figure 4. The meta-model conceptual model

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 by 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. As well 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. 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 explicit receiver. It can be a neighbor of the agent or the environment;
(v) *communication*: signal a message exchange between agents with explicit receivers (one or more). Each of those events may be raised by actions performed by agents or by the environment and updates their states.

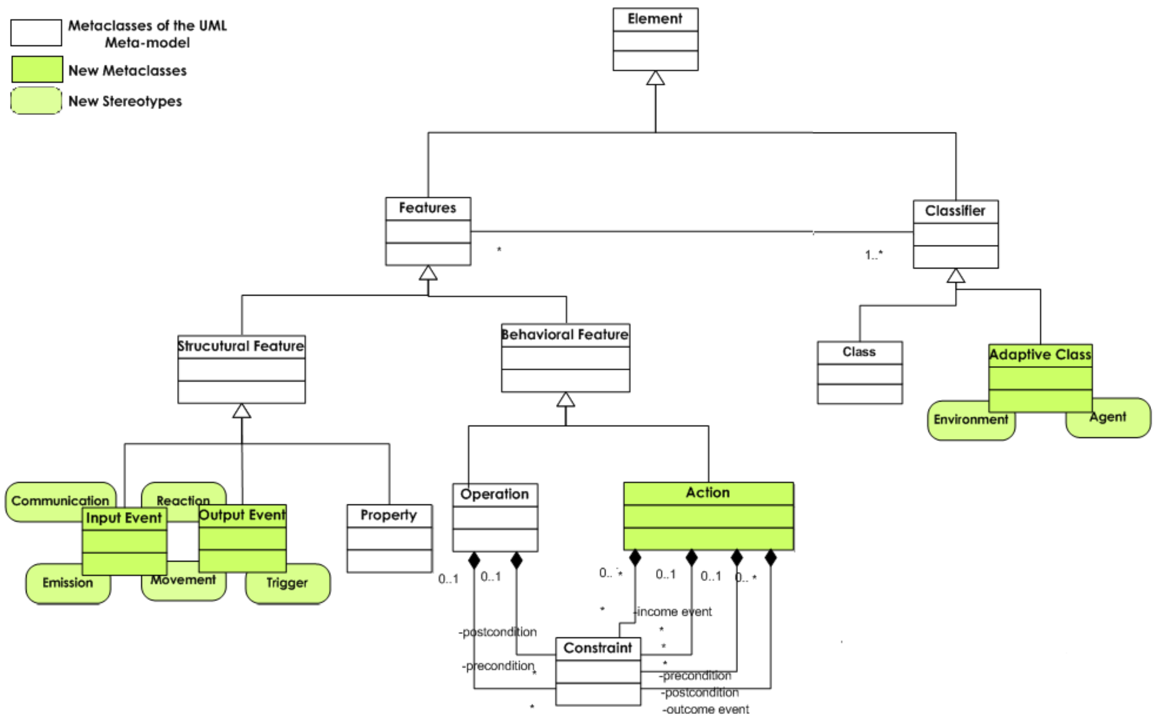


Figure 5. The meta-model proposed

Fig. 5 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. As an active object has a different meaning than an agent [28][29] we have defined the adaptive class for classifying agents and environments. The next subsection provides more detail about the meta-classes with the proposed representation model.

V.iv. The Representation Model

Agent oriented design is a technique used to separate and encapsulate agent behavior. Therefore dynamic models and the interplay that exists between the static and dynamic models are very important. A static model cannot be proven accurate without associated dynamic models. Dynamic models, on the other hand, do not adequately represent considerations of structure and dependency management. Thus, the designer must iterate between the two kinds of models to converge on an acceptable solution.

Static Model

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.

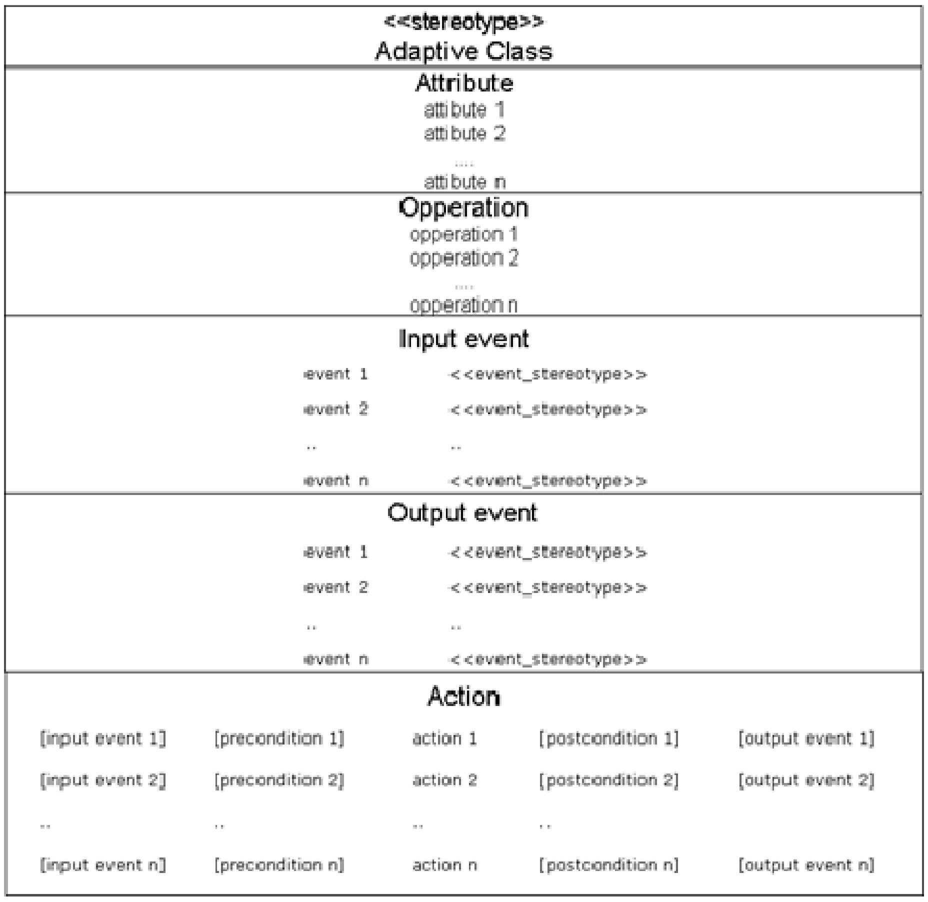

<<stereotype>>
Adaptive Class
Attribute
attibute 1
attibute 2
....
attibute n
Opperation
opperation 1
opperation 2
....
opperation n
Input event
event 1 <<event_stereotype>>
event 2 <<event_stereotype>>
event n <<event_stereotype>>
Output event
event 1 <<event_stereotype>>
event 2 <<event_stereotype>>
event n <<event_stereotype>>
Action
[input event 1] [precondition 1] action 1 [postcondition 1] [output event 1]
[input event 2] [precondition 2] action 2 [postcondition 2] [output event 2]
[input event n] [precondition n] action n [postcondition n] [output event n]

Figure 6. The Adaptive Class definition

We defined the meta-class *Adaptive* which must either be stereotyped as agent or environment. The *Adaptive* class holds for all agent common definitions (for instance, see [55], MAS-ML [28][29], AUML [27], Anote [21]). 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 since its autonomous behavior mentioned before in this paper.

We are not addressing in this work the agent local interactions through protocols and messages since there has been a huge effort from the agent research community about inter-agent communication. For instance, MAS-ML and AUML modeling languages can be easily 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.

An action is executed during agent or environment execution without explicitly being called by other objects or agents. Agents interact themselves sending and receiving messages or sending and receiving events. Regarding objects, an operation can be implemented by a method that can be called by either itself or another object. In our model, an operation can be implemented by 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 *Behavioral-Feature*, so can not 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 [48].

Input and output events vary according to the stereotypes. An output event may be the effect of a reaction as a synchronous change in state of the involved agents. The effect of the emission event is a change in the places in the environment involved in the emission. The trigger is an effect of an agent perception of a certain event in the environment. The effect of the trigger event is a change in agent or environment state according to what is perceived. The movement event is raised when an action causes a change in the agent position in the environment. And a communication event is the effect of interaction protocols. The messages exchanged through interaction protocols should be FIPA [36] compliant so they can be understood by the receiver agents.

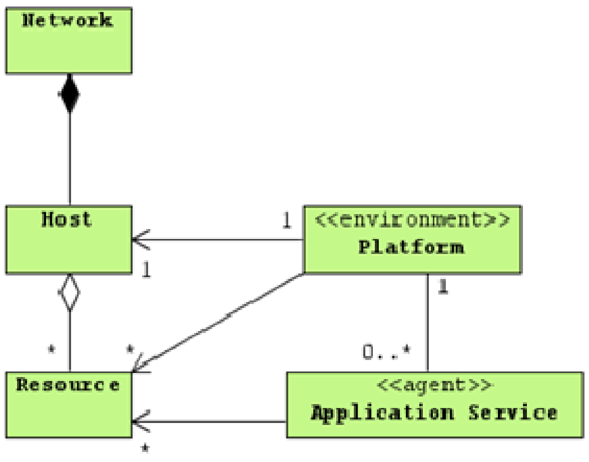


Figure 7. Example: the partial class diagram from case study

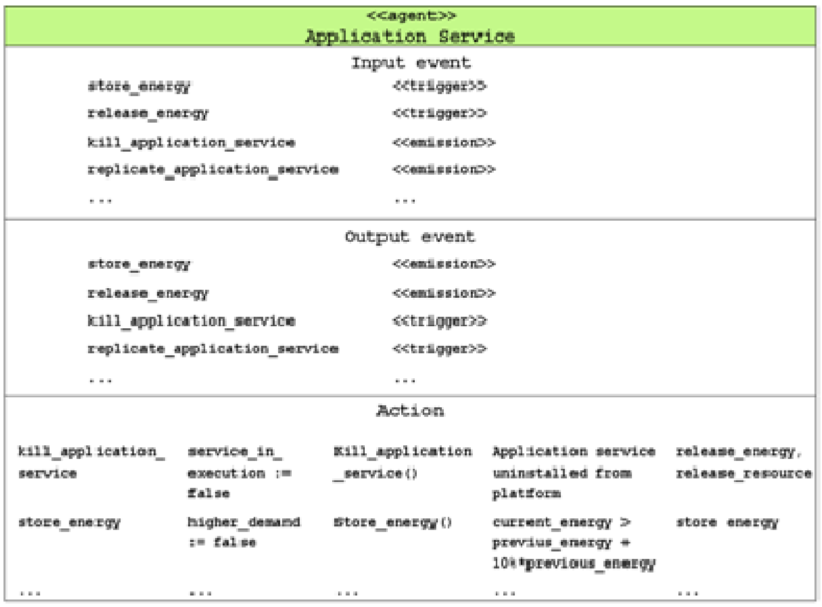
Fig. 7 exemplifies our meta-model by showing the partial class diagram from the autonomic network case study. There you can see the Platform being classified as an environment adaptive class, the Application Service being classified as an agent adaptive class and their relationships.

Fig. 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 the *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.

Fig. 9 provides the details of the input and output event and action features from the Platform class. For instance, the *Replicate_platform()* action will be executed if the platform triggers the *replicate_platform* input event raised from the energy exchange behavior and when the platform is on high demand. In this situation the *Replicate_platform()* action will raise three output events:

1. *replicate_application_service,* so all the application services occupying the platform will have the ability to replicate itself to the new platform spontaneously, which means, as soon as they can. The application services might have been executing an important service or behavior and might not be able to replicate themselves at the time the platform was replicated;

2. *release_energy*, when splitting it; and
3. *release_resource*.



<<agent>>
Application Service

Input event

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

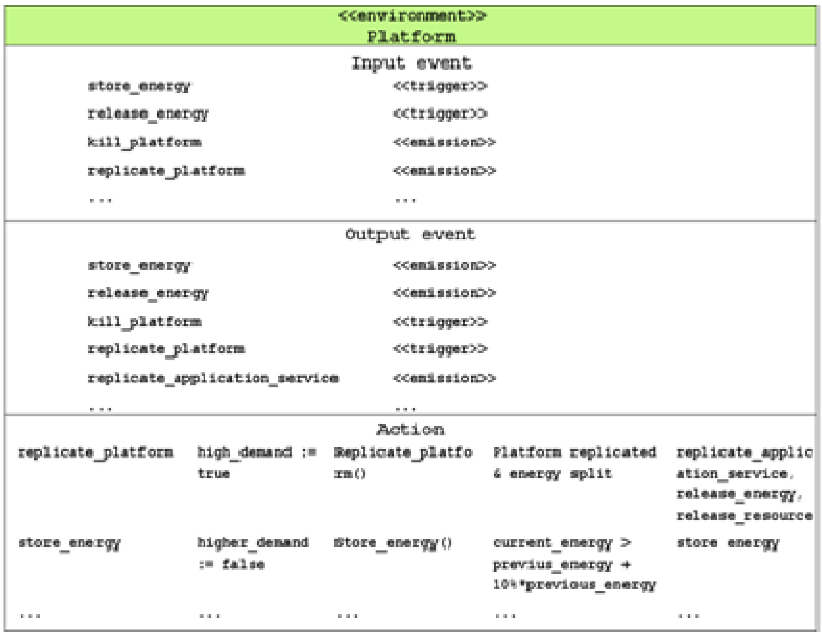
Output event

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



<<environment>>
Platform

Input event

store_energy	<<trigger>>
release_energy	<<trigger>>
kill_platform	<<emission>>
replicate_platform	<<emission>>
...	...

Output event

store_energy	<<emission>>
release_energy	<<emission>>
kill_platform	<<trigger>>
replicate_platform	<<trigger>>
replicate_application_service	<<emission>>
...	...

Action

replicate_platform	high_demand := true	Replicate_platform()	Platform replicated & energy split	replicate_application_service, release_energy, release_resource
store_energy	higher_demand := false	Store_energy()	current_energy > previus_energy + 10%*previous_energy	store energy
...	...	...	...	...

Figure 9. The Platform Environment Class

Dynamic Model

Our dynamic model is based on the UML State Machine diagram (which is basically state charts [58]), combined with action and interaction overview diagram [48][49][50]. 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 [9]; 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 adequate for our proposal.

A State Machine Diagram describes discrete behavior modeled through finite state-transition systems. The sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions can be modeled. State Machine Diagrams are applied for the state descriptions of e.g. classifiers, for detailing use cases, for behavior description of interfaces and ports, for detailed descriptions of event and signal handling. They describe states (simple, composite, submachine states), transitions, state machine, regions, initial and final states and pseudo states. In UML 2.0 interfaces can posses protocol state machines, state entry and exit and termination can be formulated and rules for transitions in inherited state machines are added and updated.

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. To date, state diagram composition is sequential. 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.

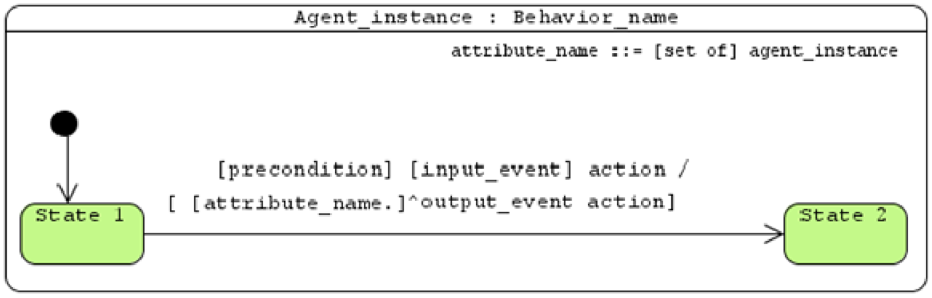

Agent_instance : Behavior_name
attribute_name ::= [set of] agent_instance
[precondition] [input_event] action /
[[attribute_name.]^output_event action]
State 1
State 2

Figure 10. The abstract state-action representation model for an agent's behavior

Fig. 10 illustrates the abstract state-action 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 than 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).

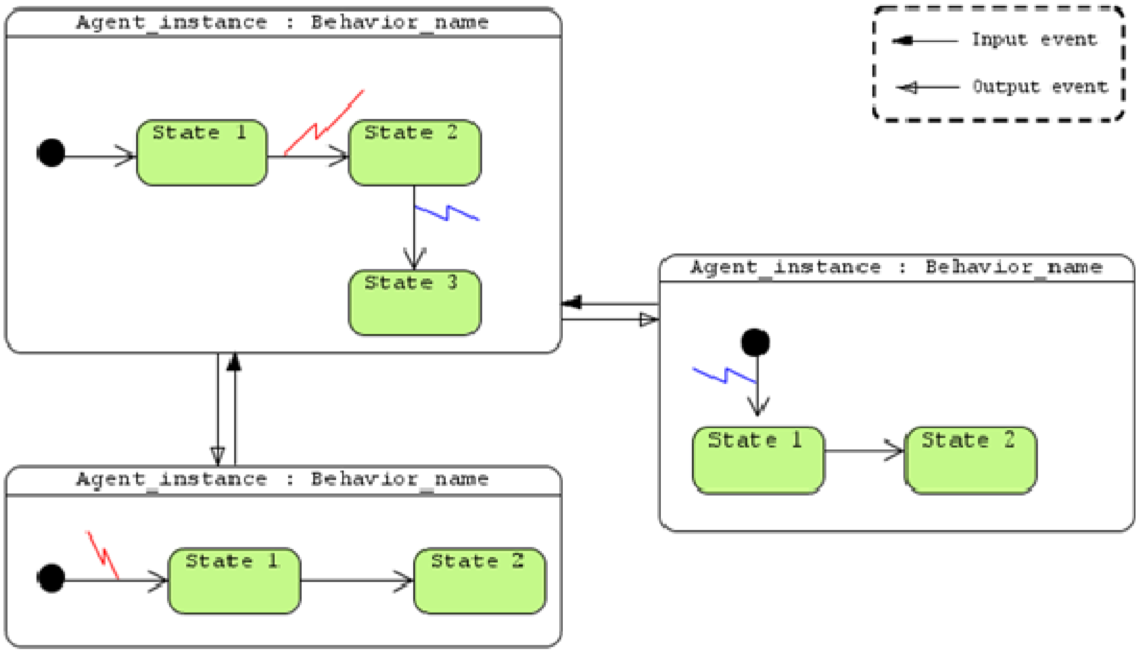

Agent_instance : Behavior_name
State 1
State 2
State 3
Input event
Output event
Agent_instance : Behavior_name
State 1
State 2
Agent_instance : Behavior_name
State 1
State 2

Figure 11. Abstract view of behavior communication channels and input versus output event perceptions.

Fig. 11 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 (red line) may be perceived as an input event that starts at the bottom left model and takes it to the 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.

We also placed some extra case study diagrams at the end of this paper (Fig. 16-17). There 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.

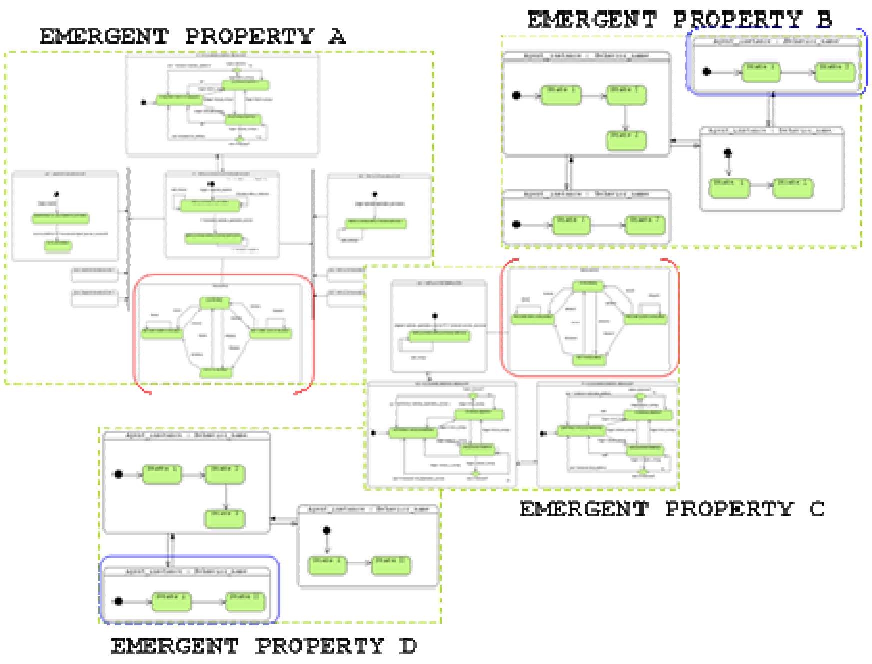


Figure 12. Emergent Properties decomposed in meso scale behaviors

In addition, there are several behavior state intersections between emergent properties. Through this modeling representation we can easy reuse them and determine how they are related with other emergent properties. For instance, in our case study we had nine emergent properties related to the modeled self-organizing patterns (migration, replication, death, energy exchange and runtime service execution) for either application service and platform. And we reused several behaviors from agent behaviors in different emergent property models. We also notice the presence of multiple environment static entities, as shared resources in those models. Fig. 12 illustrates these phenomena and, again, Fig.16-17 shows the case study models for the emergent properties desired.

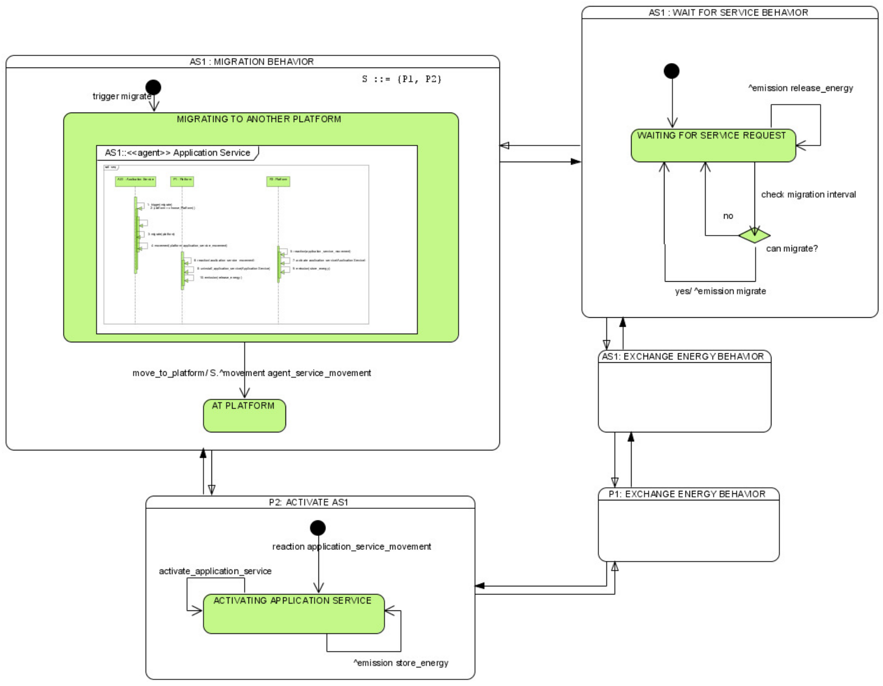

AS1 : MIGRATION BEHAVIOR
S ::= {P1, P2}
trigger migrate
MIGRATING TO ANOTHER PLATFORM
AS1::<<agent>> Application Service
move_to_platform / S.^movement agent_service_movement
AT PLATFORM
AS1 : WAIT FOR SERVICE BEHAVIOR
^emission release_energy
WAITING FOR SERVICE REQUEST
check migration interval
no
can migrate?
yes/ ^emission migrate
AS1: EXCHANGE ENERGY BEHAVIOR
P1: EXCHANGE ENERGY BEHAVIOR
P2: ACTIVATE AS1
reaction application_service_movement
activate_application_service
ACTIVATING APPLICATION SERVICE
^emission store_energy

Figure 13. From meso scale to micro scale in self-organizing multi-agent systems

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 our models 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 scale to the micro scale.

V.v. The Verification Method through Convergence

Definitely, self-organizing emergent MAS will only be acceptable in an industrial application if one can give guarantees about their macroscopic behavior. We need to measure the self-organization and we need to verify these systems. A self-organizing system can be studied through measurements at either local or global scales and more scales can be considered if self-organization spans multiple nested hierarchical levels.

Examples of typical self-organization measures include:

- capacity to reach an organization able to fulfill the goal of the system as a whole once the system is started (success/failure/time required, convergence);
- capacity to reach a re-organization after a perturbing event (success/failure/time required);
- degree of decentralized control (central/totally decentralized/hybrid);
- or capacity to withstand perturbations: stability/adaptability.

Those measures can also be obtained for our autonomic network case study.

The global behavior verification of the system consists in determining that the system complies with the desired function. The main question is how to know if a certain self-organizing emergent system exhibits the required macroscopic behavior considering that one cannot model all possible behaviors of an interaction model and thus formally proving correctness of interactive models such as self-organizing emergent systems [9][46]. Therefore, we need measurements that represent these self-organizing emergent systems and exhibit trends (an average system-wide behavior) that are predictable.

Current research [4][5][9] has been discussing configuration or parameter tuning to accomplish this task. In the Gardelli approach, if an agent is behaving differently from the average–especially for critical actions – we may decide to inspect the agent further or deny access to resources. To accomplish this task, simulation parameters are set: initial number of agents, normal vs. abnormal agent ratio, normal and abnormal agent entering rate. Anomaly detection system parameters such as the number and rate of inspections are also adjustable. In particular, in this approach we are able to make some assumptions about the percentage of abnormal behaving agents, the rate of agents entering/leaving the system, and the detection rate.

DeWolf applied the equation-free approach [9] as the verification technique. In scientific computing research, there exists a whole store of numerical analysis algorithms that support the analysis of the system dynamics and which have a mathematical foundation. Typically, these are applied to formal equation-based models. The "Equation-Free Macroscopic Analysis" approach supports the empirical application of these analysis algorithms without the need for a formal equation-based model. These techniques result in more valuable and advanced verification results compared to normal begin-to-end simulations. These results are supported by dynamical systems theory

[56]. The macroscopic equation-based models are replaced with microscopic realistic simulations. The analysis algorithms can be seen to steer the simulation process by iteratively deciding on which simulations are needed and generating appropriate initial conditions, parameter values, etc.

In both approaches it is necessary to have a manual external observation entity that evaluates the results and does parameter tuning. We go further. Let's have self-configuration during the verification phase using an intelligent verification tool.

We consider the DeWolf approach the most appropriate one. Besides being the earliest method proposed, it has been exhaustively demonstrated for self-organizing systems using agent-based solutions. Therefore we decided to extend it in order to define our convergent method.

Our method consists of having an autonomic computing approach. We encapsulate the DeWolf method in an Autonomic Manager [17][47] (Fig. 14). We complement and exploit the DeWolf approach by incorporating self-configurators and planners. Online Planners [57] are excellent tools for verifying global properties of self-organizing multi-agent systems, and indicating how local decisions perturb the system thus changing the global behavior. If the planner has the environment initial state, it finds an actions sequence that solves the problem goal. We propose to use online planning with the Learning Real Time A* algorithm [37].

The autonomic manager is responsible for monitoring the autonomic element, analyzing requirements and adapting the autonomic element so it can fulfill the requirements [17]. In our approach, there is one autonomic manager for the self-organizing system to be verified, which is represented by the autonomic element, i.e., the managed resource.

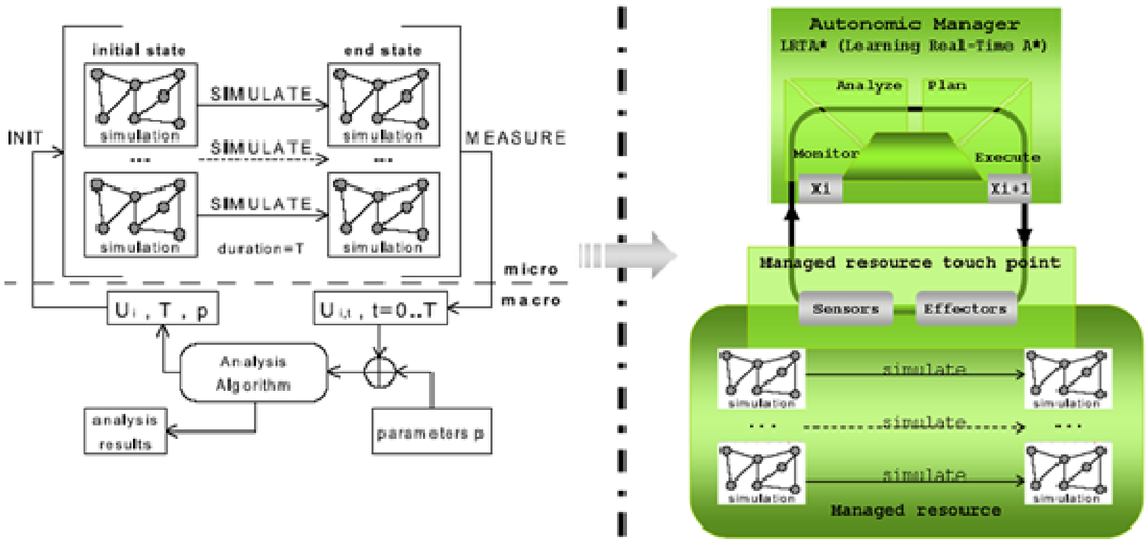


Figure 14. Autonomic Convergence Stability through Self-Configuration using LRTA*

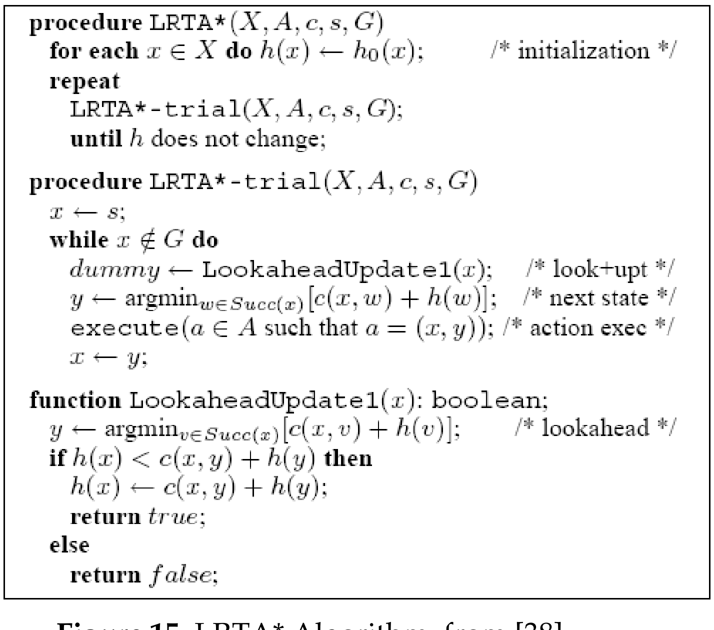
The autonomic manager has as requirements the self-organizing goals and emergent properties defined previously. The autonomic manager can perceive all the input and output events from the agents and environments through the monitor module. To create an instance of the autonomic manager, it is necessary to define symmetric actions so backward procedures can be performed, provide a domain-based algorithm that operates through the flow of control according to the actions, declare the subset of states that characterize a goal or emergent property (for each), and provide the state evaluation strategy which is based on trends or allowed average behavior.

Thus the analysis module has as input the subset of states that characterize a goal or emergent property (for each), and the state evaluation strategy. The plan module is responsible to plan effectively the system state backward steps so the execution module can execute the backward and the system may converge to a desired or optimum state. While executing those steps, the autonomic manager keeps the knowledge so it can also optimize its behavior.

Real-time search methods, such as LRTA* [37], have been used to solve a wide variety of planning problems because they can make decisions fast and still converge to a minimum-cost plan if they solve the same planning task repeatedly. It updates the heuristic estimate of the current state in each iteration.

This algorithm works on a search space where every state x has a heuristic estimate $h(x)$ of the cost from x to a goal. It is complete under a set of reasonable assumptions. If $h(x)$ is admissible, after a number of trials h(x) converges to their exact values along every optimal path.

The state space is defined as (X, A, c, s, G), where (X, A) is a finite graph, $c : A \mapsto \infty)$ is a cost function that associates each arc with a finite cost, s is the start state, and G ⊂ X is the set of goal states. X is a finite set of states, and A ⊂ X × X - $\{(x,x) \mid x \in X\}$ is a finite set of arcs. Each arc (v,w) represents an action whose execution causes the agent to move from v to w. The state space is undirected: for any action (x, y) ∈ A there exists its inverse (y; x) ∈ A with the same cost c(x, y) = c(y, x). The successors of a state x are $Succ(x)$ = $\{y \mid (x,y) \in X\}$ A path $(x_0, x_1, x_2, ..., x_n)$ is a sequence of states such that every pair $(x_i, x_{i+1}) \in A$. The cost of a path is the sum of costs of the actions in that path. A heuristic function $h : X \mapsto [0,\infty)$ associates with each state x an approximation $h(x)$ of the cost of a path from x to a goal. The exact cost $h^*(x)$ is the minimum cost to go from x to a goal h is admissible iff $\forall x \in X, h(x) \leq h^*(x)$. A path $(x_0, x_1, x_2, ..., x_n)$ with $h(x_i) = h^*(x_i), 0 \leq i \leq n$ is *optimal*.



```
procedure LRTA*$(X, A, c, s, G)$
  for each $x \in X$ do $h(x) \leftarrow h_0(x)$;          /* initialization */
  repeat
    LRTA*-trial$(X, A, c, s, G)$;
    until $h$ does not change;

procedure LRTA*-trial$(X, A, c, s, G)$
  $x \leftarrow s$;
  while $x \notin G$ do
    $dummy \leftarrow$ LookaheadUpdate1$(x)$;     /* look+upt */
    $y \leftarrow \text{argmin}_{w \in Succ(x)}[c(x,w) + h(w)]$;   /* next state */
    execute$(a \in A$ such that $a = (x,y))$; /* action exec */
    $x \leftarrow y$;

function LookaheadUpdate1$(x)$: boolean;
  $y \leftarrow \text{argmin}_{v \in Succ(x)}[c(x,v) + h(v)]$;        /* lookahead */
  if $h(x) < c(x,y) + h(y)$ then
    $h(x) \leftarrow c(x,y) + h(y)$;
    return $true$;
  else
    return $false$;
```

Figure 15. LRTA* Algorithm, from [38]

The LRTA* algorithm works as follows. From the current state x, it performs look ahead at depth d, and updates $h(x)$ to the $\max\{h(x), \min[k(x,v)+h(v)\}$, where v is a frontier state and $k(x,v)$ is the cost of going from x to v. Then, it moves to a state y, successor of x, with minimum $c(x, y)+h(y)$. This state becomes the current state and the process iterates, until a goal state is found. This process is called a trial. If the final heuristic estimates of a trial are used to solve the same problem instance, LRTA* improves its performance. Repeating this strategy, LRTA* eventually converges to optimal paths if h is admissible.

The LRTA* algorithm with lookahead at depth 1 and converging to optimal paths (with admissible) appears in Figure 15. The and functions [37], when applied to a state , generate its set of successors and its initial heuristic estimate, respectively. Procedure LRTA* initializes the heuristic estimate of every state using the function , and repeats the execution of LRTA*-trial until convergence (does not change). At this point, an optimal path has been found.

Procedure LRTA*-trial performs a solving trial on the problem instance. It initializes the current state with the start , and executes the following loop until finding a goal. First, it performs look ahead from at depth 1, updating its heuristic estimate accordingly (call to function LookaheadUpdate1). Second, it selects state y of with minimum value of as next state (breaking ties randomly). Third, it executes an action that passes from x to y. At this point, y is the new current state and the loop iterates. Note that the heuristic estimators computed in a trial are used as initial values in the next trial.

Function LookaheadUpdate1 performs look ahead from at depth 1, and updates if it is lower than the minimum cost of moving from x to one of its successors y plus its heuristic estimate . It returns true if changes, otherwise it returns false. In a state space like the one assumed here (finite, positive costs, finite heuristic estimates) where from every state there is a path to a goal it has been proved that LRTA* is complete. In addition, if h is admissible, over repeated trials the heuristic estimates eventually converge to their exact values along every optimal path [37].

Some recent work [38][39] has been proposed with new algorithms to produce better solutions in the first trial and converge faster when compared with other state-of-the-art algorithms on classical benchmarks for real-time search. They provide experimental evidence of the improvement in performance of new versions, at the extra cost of longer planning steps.

How to use the autonomic convergence method

To summarize, it is necessary to accomplish three steps:

1. Define the internal and external actions. The internal actions are the actions of the agents and the environment. The external actions are the input for the system. Considering our autonomic network case study, for each biology inspired behavior such as replication, death, migration and exchange energy there would be necessarily a corresponding reversing action so the systems can be run backwards. For example, the replication action would have a corresponding death action for the application service or platform that was replicated.
2. Define the goals. The goals are the macro properties and the emergent properties. The goals must at least match a subset of states of the system
3. Define the state evaluation. The state evaluation will analyze all the global states already reached and will verify which subset of the states matches the set of thresholds for the macro properties that represents the desired behavior.

Regarding the LRTA* algorithm, the optimal paths are the set of states being evaluated and the cost of each action can be incremented and decremented based on whether positive or negative energy is exchanged in the autonomic network case study.

Discussion

Our first evaluation on the convergent method proposed in this paper was using the Plansin open source framework [40]. This framework incorporates a search engine that may use several different strategies, and defines a structure that facilitates the construction of automated planners based on heuristic forward search that use discrete event simulators as the model for the process to be planned. Besides it already provides several different search strategies, including the LRTA* algorithm.

We planned to reuse Plansin when developing the autonomic convergence tool. However because of functionality constraints (Plansin considers the simulator as a black-box entity, but as we are dealing with hierarchies it did not completely fit our needs), we are currently developing the autonomic convergence tool addressing the method rationale described here and we will evaluate it against the case study presented and other cases studies in current development. The last section provides further details about this discussion and related work.

VI. Representations and Related Work for Self-Organizing Systems

This section presents the state of the art regarding agent-oriented methodologies and self-organization not necessarily agent-based methodology. We argue why they fall short on addressing current self-organizing engineering challenges or why our approach fulfills them better or in a complementary way.

Agent-based not-related to self-organization

Current agent-oriented methodologies focus on engineering microscopic issues [20] e.g. the agents, their rules, action-selection, knowledge-representation, how they interact, protocols, organizations, norms, etc. As argued in [25], 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 [26], Anote [21], MaSE [24], and Tropos.

We have used MAS-ML [28][29] for modeling self-organizing stem cells [30]. Again, MAS-ML is mainly focused on micro-scale issues. We tried it first because we were attracted by the existing organization concept in MAS-ML. Although it was very easy to model the organizational entities, and its interactions parts as proteins and cells, we could not model the emergent self-organized behavior. It was difficult to verify the system and adapting it would be a significant effort, as it would be necessary to adapt the conceptual framework TAO [31] on which MAS-ML is based. Also the implementation solution from the models had an undesired overhead CPU since it was not made regarding self-organizing systems.

The Agent Unified Modeling Language (AUML) [27] is a graphical modeling language that is being standardized by the Foundation for Intelligent Physical Agents (FIPA [36]) Modeling Technical Committee. AUML was proposed as an extension of the Unified Modeling Language (UML). In AUML, the agent–oriented organization defines a range of elements and notations as a requirements specification for domain modeling. It aims to provide a model and an internal architecture of an agent system. It provides a specialized abstract view of modeling (class, use-case, diagram, interface etc.). AUML use cases capture goal-oriented interactions between agents with specified roles in the software system. The agent class also defines the capabilities of that agent in the organization, the perceptions of the environment, which basically are the sensors, the protocols on which the agent interacts with other agents, and the set of organizations where the agent plays the roles with their constraints. Currently, there is no mechanism that allows the design of macro properties in AUML and no method for specifying self-organizing systems. We tried to use AUML in our experiments and it did not satisfy our self-organizing concerns regarding the engineering issues raised on section 3.

ADELFE [32] is a methodology devoted to software engineering of adaptive multi-agents. They define self-organization as the capacity of an agent to be locally "cooperative." This does not mean that it is always helping the other agents or that it is altruistic, but only that it is able to recognize cooperation failures called "Non Cooperative Situations" (NCS, which could be related to exceptions in classical programs) and to treat them. They argue that even if all the behaviors of the cooperative agents are given (agent model + NCS model), the MAS as a whole can adapt itself because the interactions are not a priori coded. Changing the interactions between agents (self-organization) changes the global function of the system and, then, allows the "strong adaptation" of the MAS.

First, the ADELFE approach simplified the concept of self-organization to cooperation. Second they do not address macro properties. They address non cooperative behaviors designed at local or agent level. Hence the approach does not allow us to design the self-organization patterns or emergent properties. In their model, the agent reasons about the emergent behavior. In self-organizing systems the local parts are not aware of the self-organizing emergent pattern as already discussed in section 2.

Hence, we can conclude that there is no explicit support for engineering macroscopic behavior in existing agent-oriented methodologies. Micro-scale issues are important, but for self-organizing emergent MAS it is necessary to deal explicitly with macro-scale issues.

Related self-organizing system research

Little work related to engineering self-organizing systems has appeared in the literature in the last five years. Joint work conducted in a technical forum within the AgentLink III NoE framework elaborated on issues concerning self-organization and emergence in multi-agent systems (MAS) and frameworks to describe self-organizing systems [35] .

The framework proposed is composed of the following six parts: the system description, the environment description, the perturbations coming from the environment, the entity description, the interaction description, and the self-organization engine. Although the framework proposed indicates the order of what should be done, it does not provide an approach to modeling; there is no indication of any method, representative model or specific methodology or language.

Van Parunak and Bruckner proposed a design guide for swarming systems engineering [33] consisting of ten design principles: the four first are derived from coupled processes (interactions through information's exchange coupled agents or processes) the next three are derived from autocatalysis and the final three last are derived from functional adjustment (the self-organizing system must produce functions that are useful to the system's stakeholders). Even if swarming systems have demonstrated their effectiveness as an alternative model of cognition and have been applied to number of applications, this approach is not very easy to apply because of the huge number of parameters to tune. The ten given principles are very general and no associated tools exist. No guide is given to indicate if the use of swarming systems is more relevant than conventional cognitive techniques for designing the current application or problem. No representation model is given which helps to map the design from micro scale to macro scale. No verification method is proposed.

Luca Gardelli [1][4][5][6] 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). Artifacts are first-class entities representing the environment which mediates agent interaction and enables emergent coordination: as such, they encapsulate and enact the stigmergic mechanisms (diffusion, aggregation, selection, …) and the shared knowledge upon which emergent coordination processes are based. He developed the engineering framework on top of the TuCSoN agent coordination infrastructure. Gardelli's approach is not a complete methodology, rather a collection of best practices to initiate the early design. It is composed of three phases: a) modeling, look for the desired global behavior among the self-organization pattern catalogue and model the strategy; b) simulation, preview the dynamics of the abstract models to see if emergent properties actually happen - models are expressed in a formal language (Pi-Calculus [5]) able to describe stochastic phenomena; c) tuning, tune the model and related parameters to obtain the specific required behavior and evaluate feasibility – they can detect abnormal behaving agents through analysis of the distribution of their actions. For instance, if an agent is behaving "differently" from the average, especially for critical actions – we may decide to further inspect the agent or deny access to resources. Although the use of formal tools allow us to gain a deeper insight in emergence and self-organization, there is a general belief and proof that emergent systems can not be specified formally [34]. There is also a gap between the emergent (macro) properties defined through self-organizing design patterns to the model itself. The meta-model allows the design of one scale. Hence if we have a multi-scale self-organizing system, we won't be able to design those cross-scales and hierarchies.

De Wolf [34] has defined a full life-cycle methodology based on the Unified Process customized to explicitly focus on engineering macroscopic behavior of such kind of systems. This customization takes place in the following steps of the process:

-After the requirements analysis is completed, one checks if an autonomous behavior is needed, if the available information is distributed, if the system is subject to high dynamics such as failures and frequent changes;

- In the design phase, general guidelines or principles, reference architectures, decentralized mechanisms allowing coordination between agents to achieved desirable macroscopic properties, have to be used to design self-organizing emergent MAS. In that sense, [34] proposes an initial catalogue including the most widely used coordination mechanisms such as digital pheromones, gradient fields, market based coordination, and tag based coordination. Furthermore, he proposes "Information flow" as a design abstraction (by extending the UML 2.0 Activity Diagram) which enables designing a solution independent of the coordination mechanism.

- In the verification and testing phase, he combines agent-based simulations with scientific numerical algorithms for dynamical system design.

Although DeWolf proposes a design approach we find two main problems as a sufficient design abstraction for self-organizing multi-agent systems. 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.

Regarding the verification and testing phase, DeWolf proposed three analysis algorithms to use in his research [9]: projective integration, Newton-based steady state determination, and bifurcation analysis. The projective integration algorithm aims at a considerable acceleration over time by minimizing the number of needed simulation steps through extrapolation. Simulations can be accelerated in other ways. Suppose the goal is to obtain the steady state behavior, i.e. we look for values of the macroscopic variables that remain constant as time evolves. One such numerical algorithm is Newton's algorithm. Then the cycle is repeated until the conditions and thresholds of the Newton algorithm decide to have reached a steady state. This results in finding steady states for every value of a given parameter, their stability, and a corresponding bifurcation diagram with respect to the parameter. A bifurcation analysis algorithm allows analyzing the quantitative and qualitative changes in the behavior of the system as a result of a changing parameter. In mathematics, particularly in dynamical systems, a bifurcation diagram shows the possible long-term values (equilibrium/ fixed points or periodic orbits) of a system as a function of a bifurcation parameter in the system. It is usual to represent stable solutions with a solid line and unstable solutions with a dotted line. We however proposed to use the LRTA* and its improved version rather than these three analysis algorithms, and we proposed to use an autonomic convergence using that algorithm (and improved versions one) to achieve self-configuration.

VII. Conclusions and Future Work

In contrast with current agent-oriented methodologies, which mainly focused on engineering such microscopic issues as the agents, the rules, the protocols, and their interaction, our focus in on explicit support for engineering the required outcome of the system and the adjustment of the emergence properties towards convergence. The novel emergence property adjustment requires a discrete agent-based model to interface with the continuous simulation tools and interact with them as they move from towards convergence.

We proposed a bio-inspired approach consisting of a method that allows a systematic specification (i.e., a representation model) of desirable macroscopic properties, which can be mapped into the behavior of individual agents, followed by development of a system, and the interactive adjustment of the required emergent macroscopic properties that need to be achieved. The bio-inspired method and representation model is used for engineering self-organizing emergent multi-agents systems. From the bio-inspired point of view, agents can interact with emergent functions being represented by emergent components composed of agents as exist in biological systems as cells. 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.

Regarding the representation model, we proposed a multiple scale design from macro to micro properties. To accomplish this design we proposed an 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 used this assumption because 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. Agents and environment behaviors can be decomposed and each of them has a communication channel in order to produce an agent state with the desired overall robust performance. Moreover, behaviors are composed of actions. Actions are executed through input events and pre conditions and raise output events. All those abstractions were created in order to achieve behavior composition through parallel behaviors.

We represented the emergent properties or self-organization patters 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.

To complement the engineering method for self-organizing systems we proposed an autonomic convergence method. We aim to know if a certain self-organizing emergent system exhibits the required macroscopic behavior. Our method consists of having an autonomic computing approach. We encapsulate the DeWolf method in, what we call, Autonomic Stability. We complement and exploit his approach incorporating self-configuration and planners. We argued that online planners are excellent tools for converging global properties of self-organizing multi-agent systems, and pointing up how local decisions perturb the system changing the global behavior. An important contribution to this paper also is that through the autonomic network case study, we concluded that engineering self-organizing multi-agent systems through the method proposed helps with engineering of autonomic computing systems.

We are still conducting several proofs of concept studies on the enabling technologies needed to realize the method presented in this paper. One of them is in the context of the stem cell project [30][43][44]. We are modeling and simulating stem cell behavior as self-organizing multi-agent systems in order to support the therapy in vitro process. We also need to develop the full distributed and multi-scale autonomic convergence tool addressing the method rationale described and apply it to the stem cell project. To date, we have done some proof of concepts only at one scale. In addition, we want to do some experiments with at least two of the available improved algorithms already proposed in the literature for Learning Real-Time A*

ACKNOWLEDGMENT. The authors wish to thank Rolando Blanco for suggesting some ideas incorporated to the work presented in this paper.

References

[1] Mirko Viroli,Matteo Casadei, Luca Gardelli 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
[2] Matteo Casadei, Luca Gardelli, Mirko Viroli. Simulating Emergent Properties of Coordination in Maude: the Collective Sorting Case. 5th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2006) a satellite workshop of CONCUR. August 2006 - Bonn -Germany
[3] Matteo Casadei, Luca Gardelli, Mirko Viroli. Collective Sorting TupleSpaces. Dallioggettiagliagenti: Sistemi Grid, P2P e Self-*, AI*IA/TABOO Joint Workshop (WOA 2006). September 2006, Catania–Italy
[4] Luca Gardelli, Mirko Viroli, Matteo Casadei. Engineering the Environment of Self-OrganisingMulti-Agent Systems Exploiting Formal Analysis Tools. AICA 2006 – Associazione Italianaper l'Informaticae ilCalcolo Automatico. September 2006, Cesena-Italy
[5] Luca Gardelli, Mirko Viroli, Andrea Omicini. 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 International Workshop "Engineering Self- Organising Applications"(ESOA 2005), Utrecht, The Netherlands (NL), July 2005.
[6] Luca Gardelli, Mirko Viroli, Matteo Casadei. On Engineering Self-Organizing Environments: Stochastic Methods for Dynamic Resource Allocation. The Third International Workshop on Environments for MultiagentSystems (E4MAS 06). To be held at The Fifth International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2006) –Future University-Hakodate, Japan, May 8, 2006.
[7] Alessandro Ricci, Andrea Omicini, Mirko Viroli, Luca Gardelli, Enrico Oliva Cognitive Stigmergy: A Framework Based on Agents and Artifacts. The Third International Workshop on Environments for Multiagent Systems (E4MAS 06). To be held at The Fifth International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2006) -Future University-Hakodate, Japan, May 8, 2006.
[8] Luca Gardelli, Mirko Viroli,Andrea Omicini. Exploring the Dynamics of Self-Organising Systems with Stochastic pi-Calculus: Detecting Abnormal Behaviour in MAS. Fifth International Symposium "From Agent Theory to Agent Implementation" (AT2AI-5). 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006) -Vienna, Austria (AT). April 2006.
[9] T. De Wolf, Analysing and engineering self-organising emergent applications, Ph.D. Thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, May, 2007, 183
[10] T. De Wolf, and T. Holvoet, Emergence Versus Self-Organisation: Different Concepts but Promising When Combined, Engineering Self Organising Systems: Methodologies and Applications (Brueckner, S. and Di Marzo Serugendo, G. and Karageorgos, A. and Nagpal, R., eds.), Lecture Notes in Computer Science, 2005, Volume 3464, May 2005, pages 1 - 15
[11] T. De Wolf, and T. Holvoet, Towards a Methodolgy for Engineering Self-Organising Emergent Systems, In Self-Organization and Autonomic Informatics (I), Volume 135 of Frontiers in Artificial Intelligence and Applications. H. Czap, R. Unland, C. Branki and H. Tianfield (editors), pp 18 - 34. ISBN: 1-58603-577-0, IOS Press. Proceedings of the International Conference on Self-Organization and Adaptation of Multi-agent and Grid Systems (SOAS 2005), Glasgow, Scotland, UK (PDF) [Best Paper Award]
[12] T. De Wolf, G. Samaey, T. Holvoet, and D. Roose, Decentralised autonomic computing: Analysing self-organising emergent behaviour using advanced numerical

methods, Proc of the Sec Int Conf on Autonomic Computing, IEEE Computer Society Proc, pp. 52-63, 2005
[13] T. De Wolf, and T. Holvoet, Using UML 2 Activity Diagrams to Design Information Flows and Feedback-loops in Self-Organising Emergent Systems, Proceedings of the Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems (EEDAS 2007), pp 52-61, Editors: Tom De Wolf, Fabrice Saffre, Richard Anthony, Publisher: CMS Press, University of Greenwich, UK, Event: co-located with ICAC 2007 @ Jacksonville, Florida, USA, 2007
[14] T. De Wolf, and T. Holvoet, Design Patterns for Decentralised Coordination in Self-organising Emergent Systems, Editors: Sven Brueckner, Salima Hassas, Màrk Jelasity and Daniel Yamins, Engineering Self-Organising Systems: Fourth International Workshop, ESOA 2006, Future University-Hakodate, Japan, 2006, Revised Selected Papers, Lecture Notes in Computer Science, Volume 4335, 2007, pp. 28–49, Springer Verlag
[15] Liporace, F. dos S.; Milidiú, R. L.; Planejadores para transporte em polidutos. Rio de Janeiro, 2005. 120 pg. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.
[16] Multi-Agent Systems: A Modern Approach to Distributed Artificial Intelligence. G. Weiss (ed). MIT Press, 2001. Chapter 4, pp. 165-199.
[17] Kephart, Jeffrey and Chess, David M.(2003). The Vision of Autonomic Computing. IEEE Computer Magazine 36(1): 41-50.
[18] P. Lin, A. MacArthur, J. Leaney. Defining Autonomic Computing: A Software Engineering Perspective. IEEE. Proc. of the 2005 Australian Soft. Eng. Conf. (ASWEC'05).
[19] S. Bandini, F. Celada, S. Manzoni, G. Vizzari: Modelling the immune system: the case of situated cellular agents. Natural Computing 6(1): 19-32 (2007)
[20] 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.
[21] Choren, R., Lucena, C. Modeling Multi-agent Systems with ANote. Software Systems Modeling Journal 4, 2005, p. 199-208.
[22] P. Giorgini, M. Kolp, J. Mylopoulos & M. Pistore. 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.
[23] H. V. D. Parunak & S. Brueckner. Engineering Swarming Systems. In F. Bergenti, M.-P. Gleizes & F. Zambonelli, editeurs, Methodologies and Software Engineering for Agent Systems, pages 341–376. Kluwer, 2004.
[24] M. F. Wood & S. A. DeLoach. An Overview of the Multiagent Systems Engineering Methodology. In Agent-Oriented Software Engineering, volume 1957 of LNCS. Springer, 2001.
[25] F. Zambonelli & A. Omicini. Challenges and Research Directions in Agent-Oriented Software Engineering. Autonomous Agents and Multi-Agent Systems, vol. 9, no. 3, pages 253–283, 2004.
[26] F. Zambonelli, N. Jennings & M. Wooldridge. Developing multiagent systems: The Gaia methodology. ACM Trans. Software Eng. Meth., vol. 12, no. 3, pages 417–470, 2003.
[27] B. Bauer, J. P. Müller, J. Odell. 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.

[28] 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.
[29] SILVA, Viviane Torres; LUCENA, Carlos José Pereira de; Modeling Multi-Agent System. In Communication of ACM (CACM), vol. 50, no. 5, Maio 2007, pp 103-108.
[30] 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.
[31] Silva, V., Garcia, A., Brandao, A., Chavez, C., Lucena, C., Alencar, P. (2003): Taming Agents and Objects in Software Engineering. In: Garcia, A., Lucena, C., Zamboneli, F., Omicini, A, and Castro, J., (Eds.), Software Engineering for Large-Scale Multi-Agent System, LNCS, Springer-Verlag, (2003).
[32] C. Bernon, et al. (2003). `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.
[33] Parunak, H., Brueckner, S. In: Engineering Swarming Systems. Kluwer (2004)
[34] DeWolf, T.: Analysing and engineering self-organising emergent applications. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium (2007).
[35] G. DI MARZO SERUGENDO, M.-P GLEIZES and A. KARAGEORGOS; Self-organization in multi-agent systems. The Knowledge Engineering Review, Vol. 20:2, 165–189. 2005, Cambridge University Press.
[36] FIPA - Foundation for Intelligent Physical Agents. http://www.fipa.org.
[37] R. Korf. Real-time heuristic search. Artificial Intelligence, 42(2-3):189{211, 1990.
[38] Carlos Hernández, Pedro Meseguer, Improving convergence of LRTA*(k). In Proceedings of Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains IJCAI-05.
[39] Carlos Hernández, Pedro Meseguer, Improving. HLRTA*(k). In Proceedings of the 12th Conference of the Spanish Association for Artificial Intelligence, CAEPIA'07. LNAI.
[40] Liporace, F. dos S.; Milidiú, R. L.; Planejadores para transporte em polidutos. Rio de Janeiro, 2005. 120 pg. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.
[41] J. Suzuki, "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.
[42] Denning, P. J. and Denning, P. J. 2007. Computing is a natural science. Commun. ACM 50, 7 (Jul. 2007), 13-18.
[43] Faustino, G., Gatti, M.A. de C., Lucena, C.J.P. de, Gattass, M., A Multi-Agent-based 3D Visualization of Stem Cell Behavior, 2008. *To be published*.
[44] Bispo, D., Gatti, M.A. de C., Lucena, C.J.P. de, An Agent-based Framework for Stem Cell Behavior Modeling and Simulation, 2008. *To be published*.
[45] A. Visser, G. Pavlin, S.P. van Gosliga, M. Maris. "Self-organization of multi-agent systems", Proc. of the International workshop Military Applications of Agent Technology in ICT and Robotics, The Hague, the Netherlands, 23-24 November 2004.
[46] P. Wegner. Why Interaction is More Powerful than Algorithms. Communications of the ACM, vol. 40, no. 5, pages 80–91, May 1997.
[47] IBM. Autonomic Computing: IBMs Perspective on the State of Information Technology. Rapport technique, IBM research, 2001.
[48] UML 2.x OMG Specification. http://www.omg.org/

[49] Martin Fowler. UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition) (The Addison-Wesley Object Technology Series)

[50] Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series). 2005.

[51] Luca Gardelli, Mirko Viroli, Andrea Omicini; Design Patterns for Self-Organizing Multiagent Systems.

[52] Marco Mameia, Ronaldo Menezesb, Robert Tolksdorfc and Franco Zambonellid; Case Studies for Self-Organization in Computer Science.

[53] Bernhard Bauer and James Odell. UML 2.0 and Agents:How to Build Agent-based Systems with the new UML Standard.

[54] Danny Weyns, Andrea Omicini, James Odell. Environment, First-Order Abstraction in Multiagent Systems.

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

[56] D. Kaplan & L. Glass. Understanding nonlinear dynamics. Springer, New York, 1995.

[57] Multi-Agent Systems: A Modern Approach to Distributed Artificial Intelligence. G. Weiss (ed). MIT Press, 2001. Chapter 4, pp. 165-199.

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

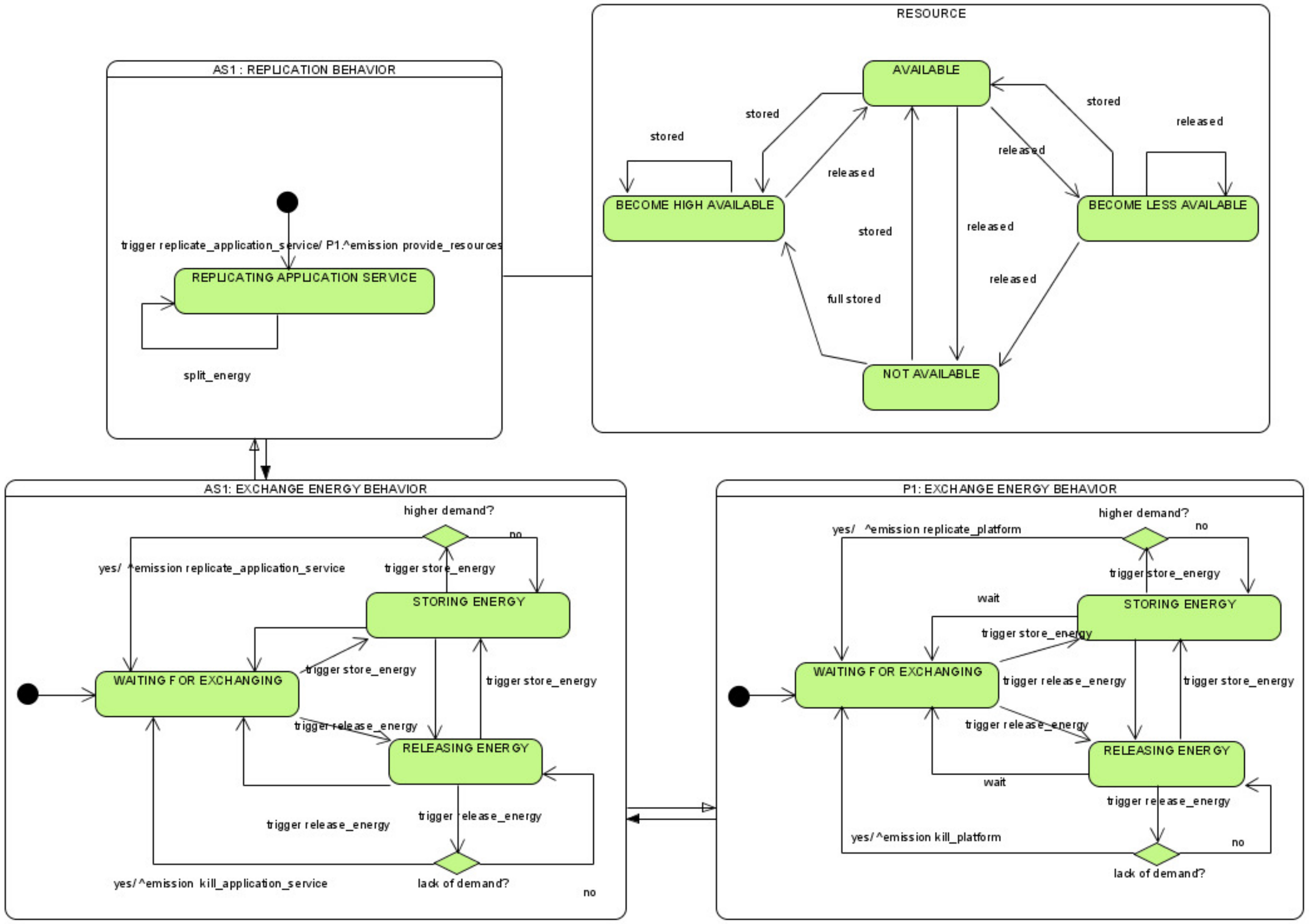


Figure 16. The Replication Pattern for the Application Service

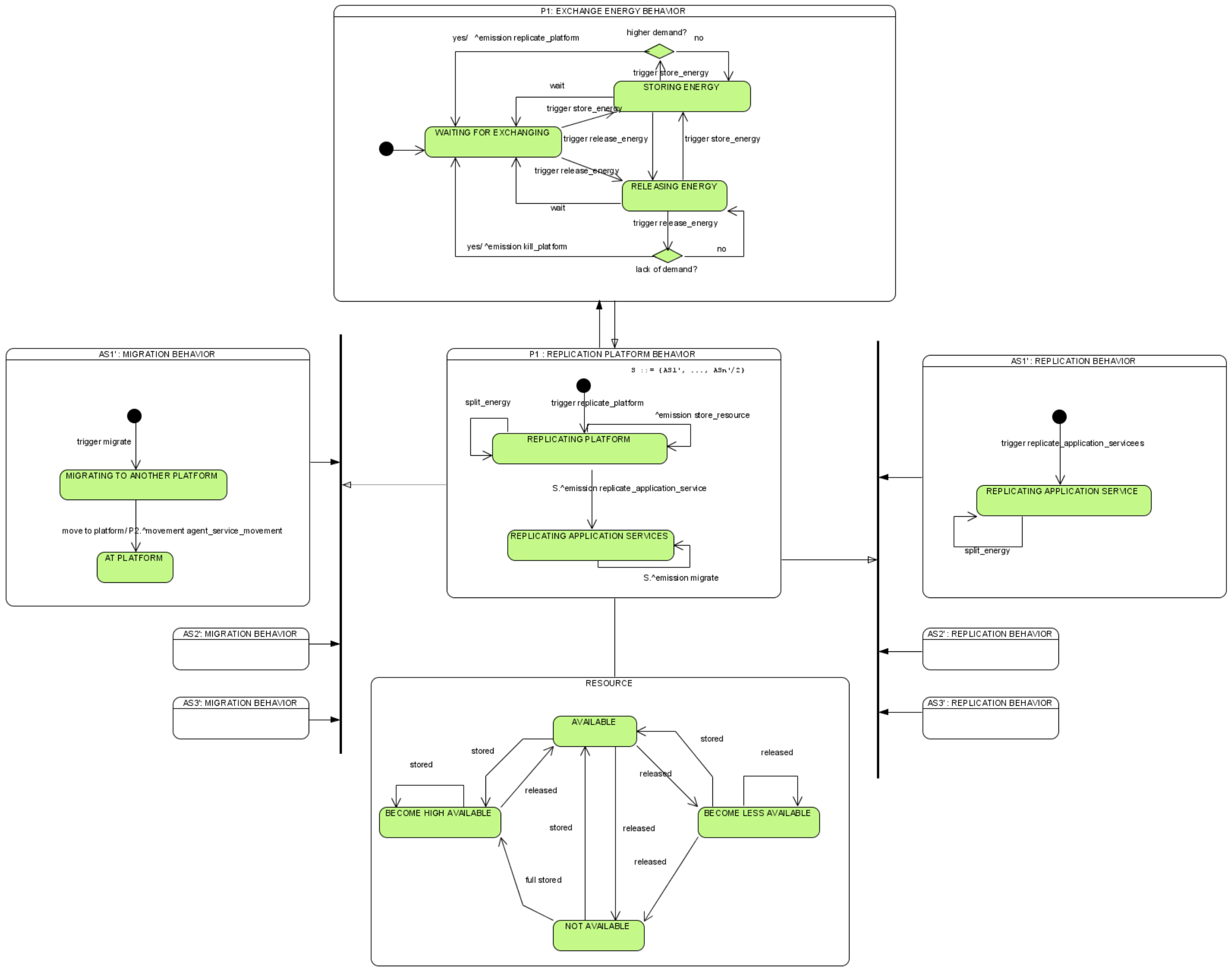


Figure 17. The Replication Pattern for the Platform

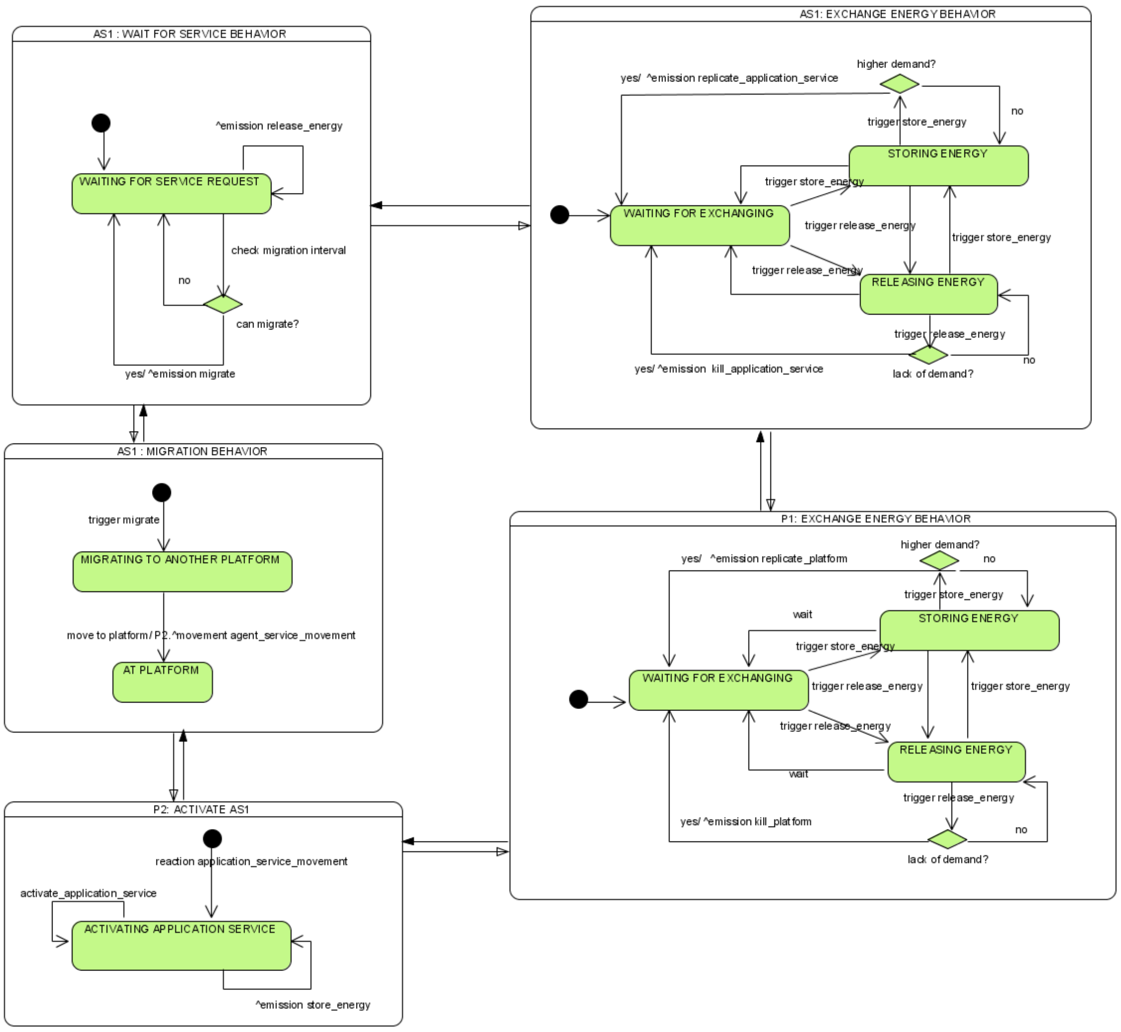


Figure 18. The Migrating Pattern for the Application Service