SIMULATION ENVIRONMENT FOR ARTIFICIAL LIFE AGENTS

Pavel Nahodil and Karel Kohout
Department of Cybernetics
Czech Technical University in Prague
Technická 2, 166 27 Prague, Czech Republic
E-mail: { nahodil, kohoutk }@fel.cvut.cz

KEYWORDS
Anticipation, Simulation, Hybrid Architecture, Animate, Behavior, Artificial Life, Artificial Creatures.

ABSTRACT

Our research is focused on simulation of agents - animates. The architecture of these agents is mainly inspired by nature; therefore they are sometimes called artificial creatures. The main contribution of this paper is the description of designed simulation environment architecture for the Artificial Life (ALife) domain. It was named the World of Artificial Life (WAL). Our platform incorporates also results of research in domain of hybrid agent architectures. Based on these results it focuses to the proposal and implementation of the simulation environment for them. First, we present the alternatives for our simulation environment which we evaluated and considered to use. Since we did not selected any of them further in the text we will formulate the problem we were solving and its important goals, followed by presentation of our proposal for such simulator and overview of achieved results.

STATE OF THE ART

This chapter is separated into two parts. First part describes the *Artificial Life* (ALife) field in general and discusses the current trends and streams. The second part is focused on the simulation environments for Artificial Life simulation.

Artificial Life

The trend in artificial intelligence leans towards communities of robots-agents. These structures appears in nature in all types of complexity starting from genes, cells, multi-cell structures, through plants, animals, groups of animals up to their societies. Similarly to the Nature also in robotics it is obvious that one super-intelligent robot (and therefore expensive) is with its abilities far behind a swarm of small, simple and less intelligent but also the less expensive robots. It is believed, that the power lies in quantity and simplicity. Also the range of possible types of tasks implemented on community of mutual cooperative robots is much wider than in case of one super-robot. Therefore approaches from *Artificial Life* are more often applied for control of community of robots. ALife approach (biological model) of implementation of intelligent behavior is inspired mostly by nature phenomena, instead of classical artificial intelligence (rational approach) which is more concerned about logic, rationality and just partially on algorithms inspired by nature. Another significant difference between AI and ALife approach is in the object of interest. Artificial intelligence is focused on complex thinking for example chess playing, text understanding, disease diagnostic etc. ALife if on the other hand focused on basic elements of natural behavior with strong stress on survival in environment. The most of existing ALife approaches are based on algorithms, which enables robots as artificially created creatures, to evolve and adapt [Kadlecek, 2001].
Biological systems have inspired the development of a large number of artificial intelligence techniques such as neural network architectures, genetic and evolutionary programming, robotic and multi-agent systems. Each biological system brings a large amount of evolutionary baggage unnecessary to support intelligent behavior, but focusing not on content but on principles, the study of animal behavior can provide a lot of models that can be successfully implemented within a robotic or agent system. Due to the inherent complexity of these systems, a multi-level analysis approach supported with a lot of experiments and simulations is required. Animals, in contrast to the majority of agent applications, in which agents are highly specialized in terms of behavior, deployment environment, learning capabilities etc., incorporate broad set of behaviors and high level of adaptability, mobility, social capabilities, proactivity, reactivity, and are employing various learning methods in one system [Wooldridge and Jennings, 1995]. They have to provide wide set of behaviors such as predator avoidance, nesting, fighting, eating, exploring, sleeping, reproduction and others. With computational power of computers we have reached the point when we can imitate Mother Nature. ALife tries to model "*the world as it is*" or "*the world as it could be*". ALife gives us chance to test our comprehension of intelligence, adaptation for living conditions and evolution [Kadlecek and Nahodil, 2001].
The philosophy of classical MAS has been reversed in Artificial Life. ALife draws inspiration from many science disciplines such as biology, ethology, sociology, psychology, mathematics (grammars) and physics. ALife is often being connected with emergence. By mutual local cooperation of primitives a new phenomenon on global level arises. This is called

Proceedings 22nd European Conference on Modelling and Simulation ©ECMS Loucas S. Louca, Yiorgos Chrysanthou, Zuzana Oplatková, Khalid Al-Begain (Editors)
ISBN: 978-0-9553018-5-8 / ISBN: 978-0-9553018-6-5 (CD)

emergence and is achieved without any central control. The principle of superposition of primitives is not valid – nonlinear behavior of elemental primitives. Tools for evolution towards more complex and more perfect structures are self-reproduction, mutation and selection.

Simulation Environments

There is a lot of a freely available application or simulators for ALife domain available on the Internet. The issue with these is that almost all of them are focused on one phenomenon or one particular problem. There are not much general frameworks that would give wide range of possibilities. We are searching for a simulation environment that would give us enough freedom to program mind of the agent while the virtual world (including not only the graphical representation but also the functionality in terms of provision of sensoric input) are provided by the application. There are just few such applications available. We will mention here two of them that in our opinion were close enough to be used. First of them is StarLogoTNG which is being developed at Media Laboratory, MIT, Cambridge, Massachusetts [StarLogoTNG, 2008]. Creators describe this project as programmable modeling environment for exploring the workings of decentralized systems. With StarLogoTNG, you can model many real-life phenomena, such as bird flocks, traffic jams, ant colonies, and market economies. This application is the closest one for our requirements hence we will go in the detail of description. It is fairly new it was published only last year. The main goal of this application is to lower the barrier to entry for programming by making programming easier. This means that it provides graphical programming language (block building) and also powerful and editable 3D environment. This gives wide area of possibilities but for our purposes it was not wide enough. We would need to use some programming constructions that are not part of the provided set.

Another simulation environment widely recognized is Swarm [Swarm, 2008]. It is a software package for multi-agent simulation of complex systems, originally developed at the Santa Fe Institute. The basic architecture is the simulation of collection of concurrently interacting agents. It allows the researcher to describe agent behaviors one by one, agent by agent, context by context, all while keeping an exact notion of time and concurrency in the world. Swarm also makes it possible to compose or decompose hierarchies of agents. Swarm Code is Object-Oriented. The swarm libraries are written in a computer language called "Objective-C", a superset of the C language. Objective-C adds the ability to create software "classes" from which individual instances can be created. These instances are self-contained entities, and the terminology of object-oriented programming turns out to be very well suited to discussions of agent-based models. Most swarm applications have a structure that roughly goes like this. First, a top level often called the "observer swarm" is created. That layer creates screen displays and it also creates the level below it, which is called the "model swarm". The model swarm in turn creates the individual agents, schedules their activities, collects information about them and relays that information when the observer swarm needs it. This terminology is not required by Swarm, but its use does facilitate it. Swarm libraries provide a number of convenient pieces of code that will facilitate the design of an agent-based model. These tools facilitate the management of memory, the maintenance of lists, scheduling of actions, and many other chores.

REQUIREMENTS AND GOALS

The main goal of our effort was to develop a simulator on a high modularity level and simple enough to be usable by anyone interested in the ALife research. Our simulator helped us to focus on the studied topic while abstracting from implementation details of the environment itself. Special care has been applied to possibilities of analysis, either during the simulation or after the simulation from saved data. Visualization modules are covering not only displaying the simulated agent world in 3D, but are targeted on efficient analysis of agents' behavior. Visualization can provide both simplified and attractive view in order to present the simulation either to broader or non-technical audience. Together with scientific views with various statistics and value details of the agent world, in or to satisfy needs of scientists for detailed analysis of behavior of the simulated system.

DESIGNED ABSTRACT ARCHITECTURE

In order to implement the set task a general abstract architecture was proposed and named WAL Abstract Architecture, WALA^2 in short. This should provide general guide or instructions, how an application for simulation of Artificial Life should be defined and implemented with care for high interoperability and modularity between various implementations. This design was not work of one person, but result of tight cooperation and many discussions. While designing this abstract architecture we kept in mind that our implementation can be superseded in future by better ones, but if it will stick with the philosophy and recommendations of the abstract architecture, agents will be transferable with none or minor reprogramming and redesign. Another goal of WALA^2 is to assure that agents can also run and compete on other implementations of same architecture. The reason for this is evaluation of results when different agent's architectures and approaches can be evaluated in base environment, or to simply test agents' behavior in different environment than he was designed for. We can observe if he can adapt and to new circumstances and survive. WALA^2 itself defines modular block architecture of platform as it is shown on Figure 1.

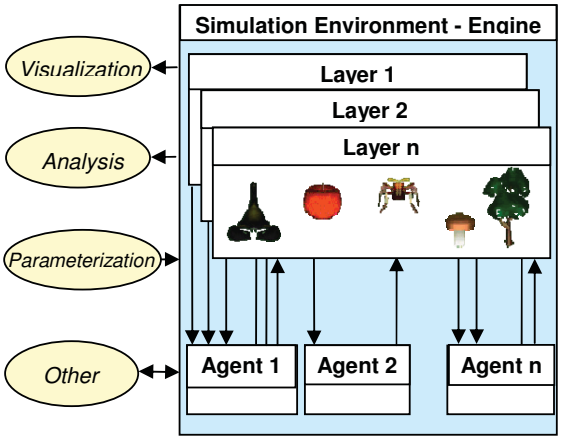

Figure 1: Block Scheme of WAL Abstract Architecture

Platform - Engine

The core part of the simulation environment will be referred to as the engine or platform. It is the base unit and it controls run of the simulation on a program level. This means it synchronizes the whole application – gives impulses on start and end of each step. It contains an interface for modules and it contains and maintains its parts. There are two components of the environment the layers and agents. In one simulation step engine asks all layers to evaluate actions of all agents and environmental changes according to these actions. The distribution of evaluation to layers means distribution of simulation control so that each layer can run in different computation thread (on multiprocessor unit they also might run on different processors). The main data structure where parameters of all layers and agents are stored is maintained by engine. This data can be view or modified by agent actions or even by external modules. It is important to distinguish between the control part of the engine, which interacts mostly with the operating system (graphical interface, loading and saving configuration, user interaction etc.) and the part providing and simulating the virtual world for agents. The first is done by above described engine. The second function is described further in the text and is handled by layers.

Engine Interfaces

Interface between simulation environment and its program surrounding (as for example visualization, analysis tool or parameterization) is important part of application. This is what makes WAL modular and distributable. From speed of data processing point of view it is suitable to exchange information in binary format. It is also possible to use text based formats such as XML. The textual format is in principle highly redundant (but descriptive) and its processing could be slow. Still, it can be used for offline analysis. Engine contains all its data, the data of layers and agents in inner tree-based data structure. It can provide all of these data or just part of it to external modules. Each connected external module can ask for data and use them for its purposes. The inner data representation is not defined in abstract architecture. It can be implemented in various ways and it does not matter as long as the interface for exchanging of this data remains the same. This interface should work in both directions for exporting the data to be read by external modules as well as for receiving updates of the data structure from modules. The running simulation can also be stopped at any moment and even traced back to certain point and run again to observe if any change of behavior will occur in exactly the same situation. Change of simulation parameters should be available while the simulation is running. As an agent we understand any object in simulation either virtually alive (creature, predator) or virtually non-living (trees, food, water, rocks). Sensors and effectors of the agent are their interface with virtual world and therefore they are part of environment and layers. Besides that agent mind and control are not part of the environment and can be also remote. The binary format is good for both, as it can be parsed very quickly.

Simulated World in Layers

This is the part of the application directly interacting with the agent via his sensors and effectors. Layers define the virtual or simulated world in which agents live. It serves for logical and computational separation of operations which has to be controlled by engine. The layer is a logically separable part of the environment which can be used standalone and which when combined with others defines the environment as whole. All agents having influence in a particular layer or being influenced by this layer must be registered in it. This means that the layer has the full information for computing the next step. The layer will evaluate and execute agents' actions in each step. According to the executed actions of the agents the layer will modify its own values and then will provide a new sensory data to agents. The layer must have the ability to register and deregister the agent and also fill the sensors of those registered agents. This means, that it must have interface for communicating with sensors. For example the thermal layer should have the ability to provide information for sensors of temperature. In each step the layer must read agents executed actions, evaluate them (whether they are possible or executable), modify the environment and provide new sensory data. Basically there were two types of layers defined. The point layer in which can the value be evaluated directly or can be obtained immediately from layers data. The gradient layer where the value in the point of interest cannot be evaluated just from the actual information but also the history of the value must be taken in account. The physical body of the agent and his sensors and effectors are also part of the environment, so it is necessary to interpret them in it. The layer must have the

information how much space the body, sensors and effectors takes. This could enable building of more complex agent from basic blocks. The potential of the layers was then used in several works, when the implementations had up to seven different layers [Foltyn, 2005]. The advantage of layers is that they can solve communication between agents in sense of "physical" distribution of the signal. We can implement the acoustic layer, and propagate the sound based on the physical laws. This solves the problem of transporting the message, not the understanding and context of the message. To sum up, layers in one simple sentence, they implement various physical laws. Complete physical description is almost unachievable. Layers can bring us closer to it.

Human Interface and Analysis

All described above is just an algorithm with no human interface. Visualization of designed world can be both attractive and useful tool. For this purpose external visualization module or internal (default) can be used. Internal visualization is meant to debug and observe simulation by creator (Figure 2).

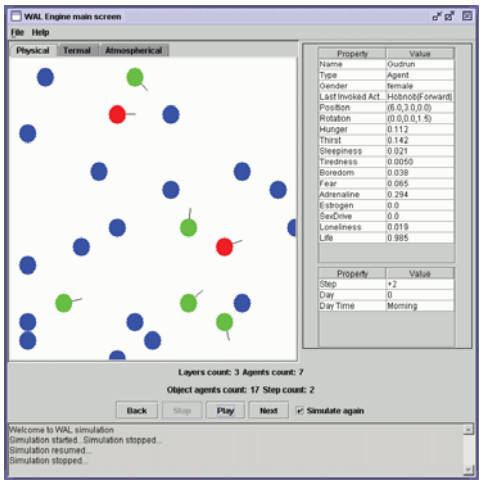

Figure 2: Example of Internal Visualization

The external module can be the exact opposite. It can be used to present this simulation to wider audience than the science community (Figure 3). On-line or off-line tools for parameters flow analysis are also supported. Proposed environment is compatible with highly effective 3D visualization tool called *Visualization Analytical Tool* (VAT in short). It can be used to observe agents parameters at any time of simulation. I will not go into much detail about the parameter visualization problem. Information about it can be found in [Kadlecek et al. 03]. By using the third dimension for data visualization the analysis is more comprehensive and computer graphics knows various methods to visualize even more than thee dimensions. That is why the 3D analytical tools are strongly supported. This leads to many advantages, because the value of the parameter can be also mapped to shape, height, width or length or many others. Another advantage is possibility to use sensitivity analysis. Analytical tools provide offline or online evaluation of simulation together with fast orientation in complex situations. It also enables the possibility of backward analysis of interesting simulation. It can be also used to observe relations between sensory inputs and executed actions (i.e. what action was triggered under when there was a specific sensory input and vice versa).

Figure 3: Example of External Visualization

Agent of WAL

WALA[2] separates the body of agent (physical representation of agent) from the mind (behaviour control mechanism). The agent's body is part of the environment and therefore it is covered in environment architecture. The mind of an agent on the other hand communicates with the body through data from sensors. Please note that even agents own state has to be observed by sensors. This covers the state of agents sensors and effectors (some of them can be damaged or partial malfunctioning) and the Vegetative System Block (capturing the agent body hormone levels). After agents sensors are filled with data, the body sends them to the mind of agent, where they are processed. The way in which the data are being processed is not a subject of this abstract architecture. There are a large number of approaches for agent mind design (architecture). Examples of such architectures, widely recognized and published can be found in following works [Kadlecek and Nahodil 2001], [Foltyn 2005], [Mach 2005]. The mind evaluates the situation and selects the action or actions for execution (parallel actions are considered). The body tries to perform these actions using its effectors. The layer mentioned above will then evaluate the cause of actions by setting new sensory data. Agent's body is part of the environment and as such it has physical properties such as position, shape or temperature. The decomposition is shown on the next Figure 4.

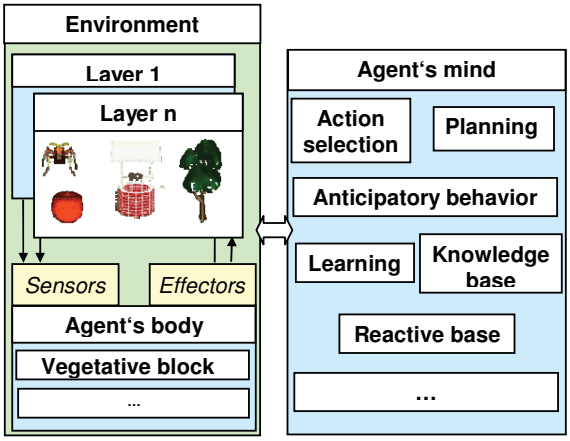

Figure 4: WAL Agent Decomposition

Agent's Sensors and Effectors

Sensors are the agent's senses which they use to perceive the surroundings and also its state. The richness of information about the surrounding world the agent can obtain depends just on the type and number of the sensors. Layers should know all types of possible sensors to be able to fill them with data. This means that creating a new layer necessarily requires also the creation of adequate sensors or altering the layer so it can work with current sensors and vice versa. When adding a new sensor it is also necessary to alter the layers so they know this sensor and are able to fill it. Effectors serves agent for interaction with its surroundings. We use simplified effectors. For example we use effectors of motion which can move agents in certain direction with certain speed. Of cause we could go deeper in detail and implement effectors such as leg or wheel, but this would distract us (by solving inverse and forward kinematics tasks etc...) from observing the behavior. We do not require this level of detail, but we are not running away from it, and the possibility to implement it is still open.

Communication

Communication between the agent body and the layers is internal communication so there is no need for explicit data sending. This communication can be done via the internal data structure, which is in this case the shared medium. The communication between body and mind is in general done via messages. It can be done even remotely via various media (for example over TCP/IP see Figure 5). Communication on the agent level means sending a message from one agent to another or group of agents. Here we would like to let layers decide who receives the information and who not. We are trying to reflect the real world behavior where information is carried via different media and can be received by various entities based on their sensor capabilities. When one agent wants to send a message ("say something") to another it will use its effectors and certain media (acoustic wave). The information can be then received not only by the addressee but also by another agent who is in the range of the signal (even if it did not request this information). It can disregard it, or abuse it for its own purposes.

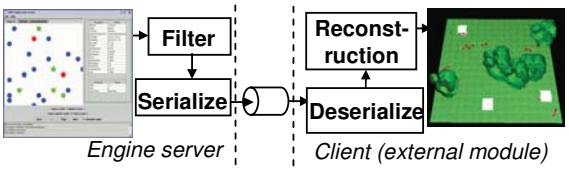

Figure 5: Block Scheme of External Communication

VISUALIZATION CASE STUDY

We would like to proof usefulness of external modules for simulation analysis on a case study performed while redesigning the agent to WAL environment. The simulation scenario concludes a single agent which was intended to move an object between two places. The agent had enough food and water to satisfy its needs. Figure 6 shows the visualized data from this simulation. On the left there is a 3D mesh; on the right is a detail of values in the 60th step. Even a very first look at the 3D mesh could advise that there is something wrong with the simulation. Almost all of parameters are zero (the mesh is flat) [Řehor, 2003].

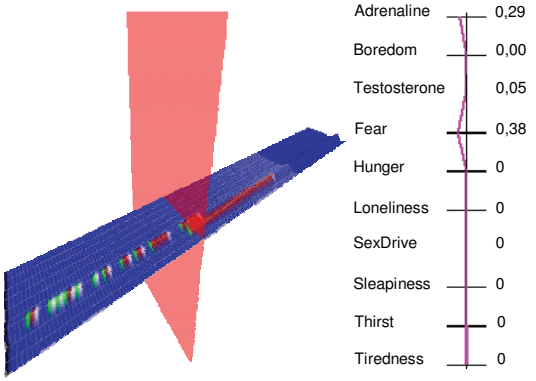

Figure 6: Visualized Data of Simulation -Using VAT

This means that the agent is neither hungry nor thirsty; it is neither tired nor sleepy. The reason for this could be a data export failure, a mistake in implementation of the inner agent vegetative block or a bad initial configuration of an agent. Because we run the simulation previously and export of data and the vegetative block were working properly, there is no problem with implementation itself. Brief check of the configuration showed that - there was to high value set to the time function for the chemical increasing/decreasing. Figure 7 shows the mesh after the configuration mistake correction.

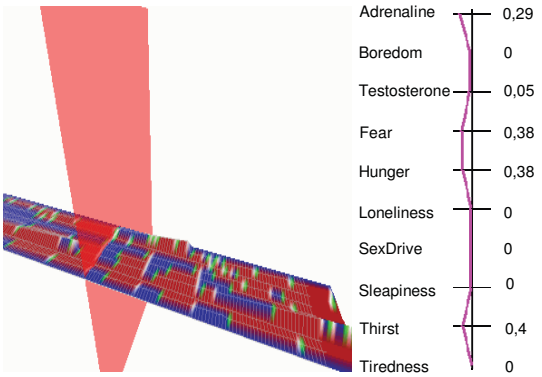

Figure 7: Use Case: Correct Setup – Using VAT

RESULTS DISCUSSION

It is very difficult to provide comparison of simulation environments. However the usefulness and applicability of the platform needs to be measured somehow in order to evaluate which parameters are better in our proposition from other works. The main difficulty is that there are two components to run the simulation. First of them is the hereby described environment, second is the control of the agent itself (agent mind). Comparing just the environments without agents would limit us just to naming the programming tools, used 3D engines and maybe reaction time of the application itself. This would not be enough to compare from the ALife simulation point of view. If we place an agent into such environment that it is again hard to distinguish between the qualities of an agent and qualities of the environment. The scientific way to solve this situation would be to create a reference agent and place it into multiple environments and then compare. This approach unfortunately faces technical difficulties in terms of modularity of compared applications, different implementation languages etc. This would lead to the abstracted comparison where we would need to define just qualities of the agent and implement these agents separately. Another objection might be raised here for objectivity of such approach. Different implementations would skew or results. Each programming language used for that has its strengths and weaknesses. To avoid these we would really need a very simple abstract agent. Would this agent test the qualities of the simulation environment? We believe not. This whole discussion leads us to conclusion that on the field of simulators especially from the Artificial life point of view we still do not have a comparison method. Also this topic will be in the scope of our further research. We would like to briefly introduce two architectures which are powered by WAL simulation environment and has been published lately.

Architectures Using WAL

One of the first architectures using the WAL environment was anticipatory agent architecture named Lemming [Foltyn, 2005]. It uses the algorithms known from Artificial Intelligence for agent's learning and offers an alternative to genetic programming and the artificial neural networks commonly used in ALife domain. This model ensures agent's survival in unknown dynamically changing environment. As you can se on the Figure 8 the visualization of the world changed but the engine is powered by the WAL architecture.

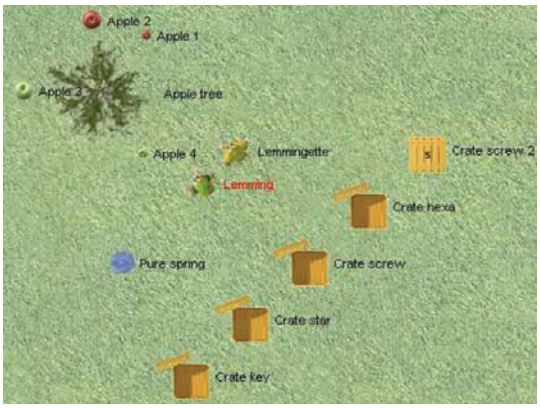

Figure 8: Architecture Lemming

Architecture is ACS proposed by M. Mach [Mach, 2005]. It uses successfully the Hidden Markov models combined by a reinforcement learning method.

REFERENCES

Kohout, K. *Simulation of Animates Behavior.* Diploma thesis. Prague: Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics, 2004.

Kadleček, D., Nahodil, P. *New Hybrid Architecture in Artificial Life Simulation.* In: Lecture Notes in AI 2159, Berlin Heidelberg: Springer Verlag, 2001. pp. 143-146.

Foltýn, L. *Realization of Intelligent Agents Architecture for Artificial Life Domain*. Diploma thesis. Prague: Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics, 2005.

Mach, M. *Data Mining Knowledge Mechanism of Environment Based on Behavior and Functionality of it's Partial Objects.* PhD thesis. Prague: Czech Technical University in Prague, FEE, Dept. of Cybernetics, 2005.

StarLogo on the Web [online]. Last revision Jan 2008 [cit. 2008-04-10]. < *http://education.mit.edu/starlogo-tng/*>.

Řehoř, D., Kadleček, D., Slavik, P., Nahodil, P. *VAT - A New Approach for Multi-Agent Visualization.* In: 3rd Int. Conference on Visualization, Imaging and Image Processing, 2003, Spain: ACME Press, pp. 849 – 854.

The Swarm Development Group [online]. Last revision Feb 2008 [cit. 2008-04-10]. < http://www.swarm.org/>.

Wooldridge, M. J. - Jennings, N. R.: *Intelligent Agents.* Lecture Notes in Computer Science Vol. 890, Berlin Heidelberg: Springer-Verlag, 1995, pp. 194 – 199.