Acta Polytechnica Vol. 47 No. 6/2007

Simulation Environment for Artificial Creatures

K. Kohout, P. Nahodil

Our research is focused on the design and simulation of artificial creatures – animates. This topic has been addressed in our research group for the last decade. The designed animates have been greatly inspired by sciences such as ethology, biology and psychology. Several agent architectures have been proposed and tested in recent years. We started to face the problem of comparing various architectures in order to benchmark them. Intelligence is embodied in our agent, and it needs an environment to be placed into. We have solved both these problems by first proposing and then implementing a common simulation environment in which these agents can run and compete. The main contribution of this paper is to offer a description of this designed simulation environment. It has been named the World of Artificial Life (WAL).

Keywords: anticipation, ALife, hybrid architecture, behavior, animate, artificial creatures.

1 Introduction

Facing the need for a simulation environment, we investigated whether we can use some of the existing enviroments. We found that are many freely available simulation platforms. We analyzed them with the aim to find a platform capable of running the required simulations, but none of those analyzed was fully able to satisfy our needs. We also found that they are mainly focused on just one specific domain of Artificial Life. Almost none of them are capable of simulating Artificial Life on a more general level. And just a handful of them also provide a set of analytical tools to evaluate the simulation in a broader context. We therefore decided to design our own simulation environment. The main goal was to develop a simulator on a high modularity level and simple enough to be usable by anyone interested in ALife research. Special attention was given to the possibility of making an analysis, either during the simulation or after the simulation, from the saved data. Visualization modules involve not only displaying the simulated agent world but are targeted on efficient analysis of agents' behavior. Visualization can provide both a simplified and an attractive view in order to present the simulation to a broader or non-technical audience. This simulator has helped us to focus on the topic under study (whatever it was) while abstracting from the implementation details of the environment itself.

2 Designed abstract architecture

Our requirements for this platform were as follows:

- The ability to simulate various phenomena of Artificial Life from cellular automata, boids, bimorphs, ant colonies etc., up to complex and socially behaving agents.
- The ability to export simulation inner data so that it can be used for parameter visualization.
- Variability of simulation with easy modifiability and step by step run.
- Interesting visualization of the agent world, in order to present the simulation to a wider or non-technical audience.
- Meaningful and helpful visualization of parameters in time.
- Modularity
- Easy extendibility
- Interoperability.

In order to implement such task, a general abstract architecture was proposed and named WAL Abstract Architecture, WALA^2 in short. It should provide a general guide or instructions on 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 the result of tight cooperation and many discussions among all MRG members [1]. While designing this abstract architecture we kept in mind that our implementation can be superseded in future by better ones, but if it sticks with the philosophy and recommendations of the abstract architecture, i.e. if the interfaces remain the same, the agents will be transferable with no or only minor reprogramming and redesign. Another goal of WALA^2 is to ensure that agents can also run and compete on other implementations of the same architecture. The aim is to ensure that when different agent architectures and approaches are used the results can be evaluated in the basic environment, or that test agents' behavior can be tested in a different environment than the agent was designed for. We can observe if the agent can adapt and to new circumstances

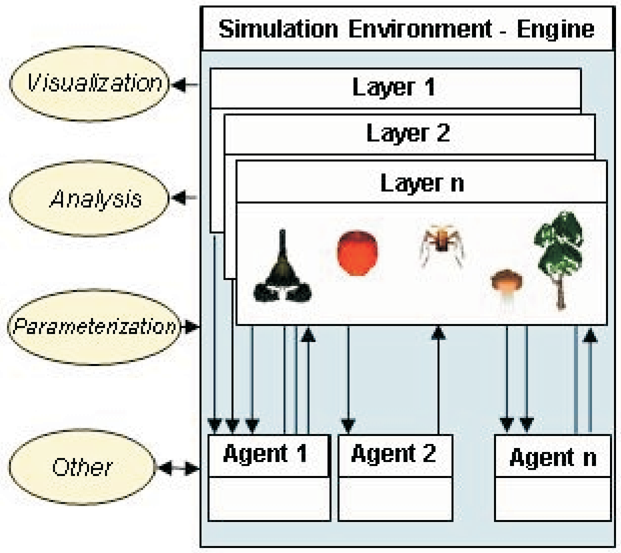


Fig. 1: Block scheme of WAL abstract architecture

and survive. WALA[2] itself defines the modular block architecture of the platform, as shown in Fig. 1. This architecture was designed to enable easy parameterization of simulation and distributivity of its parts (body and mind can be separated and even run on different computation units). One of the benefits is that the environment is divided into layers. This is not layered architecture in the agent design but in the environment design. This decomposition of the environment leads to simpler and more comprehensive simulation and also gives an opportunity to describe more complex environments.

2.1 Platform – engine

The core part of the simulation environment will be referred to as the engine or platform. It is the basic unit and it controls the run of the simulation on the program level. This means that it synchronizes the whole application – it gives impulses at the start and end of each step. It contains an interface for modules. There are two components of the environment: the layers and the agents. In one simulation step, the engine asks all layers to evaluate the actions of all agents and perform appropriate environmental changes. The distribution of evaluation to the layers means distribution of simulation control. Each layer can run in a different computation thread (on a multiprocessor unit they might also run on different processors). The main data structure where the parameters of all layers and agents are stored is also maintained by the engine. This data can be viewed or modified by the agent 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 the agents. The first is done by the engine described above. The second function is described below, and is handled by the layers.

2.2 Engine interface

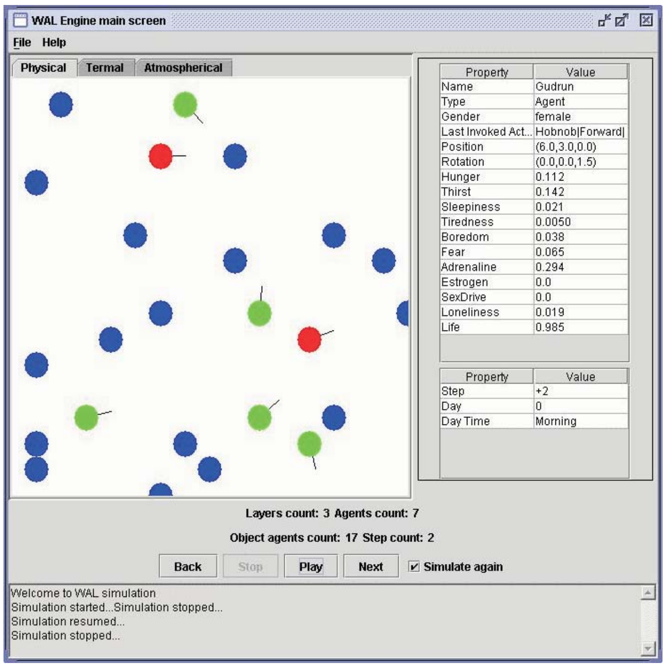
The interface between the simulation environment and its program surrounding (e.g., visualization, analysis tool or parameterization) is an important part of the application. This is what makes WAL modular and distributable. From the point of view of the processing speed of the data, it is suitable to 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 can be slow. Still, it can be used for offline analysis. The engine contains all its data, the data of the layers and agents in an inner tree-based data structure. It can provide all of this data or just part of it to the external modules. Each connected external module can ask for data. The inner data representation is not defined in the abstract architecture. It can be implemented in various ways and it does not matter as long as the interface for exchanging this data remains the same. This interface should work in both directions: for exporting the data to be read by the external modules, as well as for receiving updates of the data structure from the modules. The running simulation can also be stopped at any moment. Thanks to the single data structure it can be saved at each step, hence the simulation can even be traced back to a certain point in history and run again to observe if any change of behavior will occur (emerge) given the same starting conditions. Change of simulation parameters should be available while the simulation is running. An agent is understood to be 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 the virtual world and therefore they are part of the environment and layers. The agent mind (decision control) is not part of the environment and can be remote.

2.3 Simulation world in layers

Layers are the part of the application directly interacting with the agent via its sensors and effectors. Layers define the virtual world in which the agents live. They separate operations which would otherwise be controlled by the 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 a whole. Basically, by using layers we segregate the simulation of physical laws. As an example, we can have a physical layer taking care of agents' positions and collisions. We can add a thermal layer taking care of propagation of heat in the virtual world. All agents influenced by a particular layer or having an influence in this particular layer must be registered with it. This ensures that the layer has full information for computing the next step. The layer will evaluate agents' actions in each step, and through their sensors it will provide them with the new state of the environment. According to the executed actions the layer will modify its own values and then will provide new sensoric data to the agents. The layer must have the ability to register and deregister the agent and also fill the agents' sensors. This means that it must have an interface for communicating with sensors. For example, the thermal layer should have the ability to provide information for sensors of temperature. To sum this up in each step the layer must read agents' executed actions, validate them (whether they are possible or executable), modify the environment and provide new sensory data. We have defined two types of layers. There is the point layer, where the value can be evaluated directly, or it can be obtained immediately from the layers data. The gradient layer is where the value at the point of interest cannot be evaluated just from the current information but the history of the value must also be taken into account. The physical body of the agent and its sensors and effectors are also part of the environment, so it is necessary to interpret them in it. The layer must have information about how much space the body, sensors and effectors take. This could enable a more complex agent to be built from the basic blocks. The potential of layers has been used in several works, where the implementations have had up to seven different layers [5]. Another advantage of layers is that they can solve communication between agents in the sense of distribution of the signal. We can implement the acoustic layer to propagate sound based on the physical laws. This solves the problem of transporting the message, but not the understanding and context of the message. To sum up layers in a single simple sentence, they implement various physical laws.

2.4 Human interface and analysis

Everything that has been described above is just an algorithm with no human interface. Visualization of the designed world can be an attractive and also a useful tool. For this purpose, an external visualization module or an internal (default)



Property	Value
Name	Gudrun
Type	Agent
Gender	female
Last Invoked Act...	Hobnob|Forward|
Position	(6.0,3.0,0.0)
Rotation	(0.0,0.0,1.5)
Hunger	0.112
Thirst	0.142
Sleepiness	0.021
Tiredness	0.0050
Boredom	0.038
Fear	0.065
Adrenaline	0.294
Estrogen	0.0
SexDrive	0.0
Loneliness	0.019
Life	0.985

Property	Value
Step	+2
Day	0
Day Time	Morning

Layers count: 3 Agents count: 7

Object agents count: 17 Step count: 2

☑ Simulate again

```
Welcome to WAL simulation
Simulation started...Simulation stopped...
Simulation resumed...
Simulation stopped...
```

Fig. 2: Example of internal visualization

module can be used. Internal visualization is meant for debugging and observing the simulation by the creator (Fig. 2). The external module can be used to present this simulation to a wider audience (Fig. 3). On-line or off-line tools for analysis of changes in agent attributes in time are also supported. The proposed environment is compatible with the visualization tool called VAT. It can be used to observe agent parameters at any time of simulation. We will not go into much detail about the parameter visualization problem. Information about this can be found in [4]. Using the third dimension for data visualization makes the analysis more comprehensive, and computer graphics has various methods for visualizing even more than three dimensions. This is why 3D analytical tools are strongly supported. They offer many advantages, because the value of the parameter can be mapped to shape, height, width or length, etc. Another advantage is the possibility to use sensitivity analysis. Analytical tools provide an offline or online evaluation of the simulation together with fast orientation in complex situations. They also enable backward analysis of an interesting simulation, and they can be used for observing the relations between sensory inputs and executed actions (i.e. what action was triggered when there was a specific sensory input, and vice versa).

2.5 *Influencing the simulation*

Parameterization provides the ability to alter the simulation either as an initial setup of simulation or a direct change to the simulation in runtime. This means changing the agent or layer parameters while the simulation is in progress. For example, you can set a new target for the agent, decrease the temperature at a particular spot or even create a new object (agent, food, etc.). This should provide the ability to run longer simulations. User intervention via parameterization can be used for example in a learning process (imitation, conditioned and unconditioned reflexes, etc.). Moreover, combined with visual analytical tools it is possible to observe the interesting moments of the simulation and change the scenario to see how the agents will adapt to this change. Pausing and resuming the simulation and tracing step by step (even backwards) are also a part of parameterization. Backward run is a key element that had been missing until now in simulations. For observing emergent behavior, it is useful when we can trace the simulation back to some interesting point and run it again in order observe if the situation will end exactly as it did before or if it will differ.

2.6 *Agent of WAL*

WALA^2 separates the body of the agent (physical representation of agent) from the mind (control mechanism). The agent's body is part of the environment and therefore it is covered here in the environment architecture. The mind of an agent on the other hand communicates with the body through data from sensors. Please note that even the agent's own state has to be observed by sensors. This internal state covers the state of agent sensors and effectors (some of these may be damaged or partial by malfunctioning) and the Vegetative System Block. The data from sensors is send to the mind of the agent, where it is processed. How the data is processed is not a subject of this abstract architecture. Several approaches to agent mind design can be found in [2, 5, 6]. Finally, the mind evaluates the situation and selects the action or actions for execution. The body tries to perform these ac-

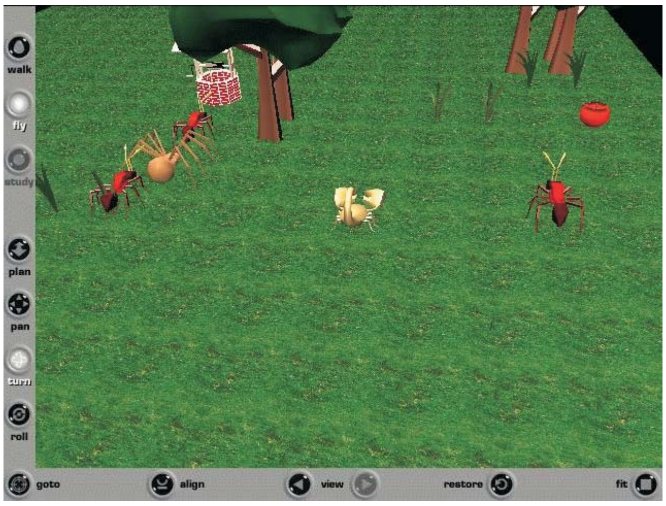


Fig. 3: Example of external 3D visualization

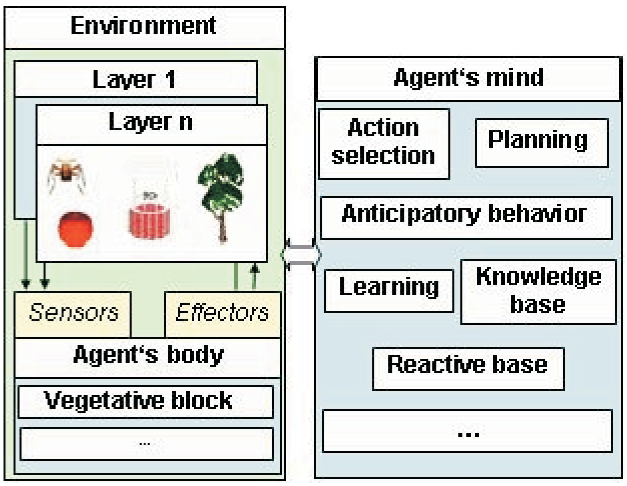


Fig. 4: WAL Agent decomposition

tions using its effectors. The layer mentioned above will then provide feedback about the effect of these actions by setting new sensory data. The agent's body is part of the environment and as such it has physical properties such as position, shape, temperature, etc. The decomposition and the interfaces of the agent are shown in Fig. 4.

2.6.1 Agent's sensors

Sensors are the agent's senses. They are used to perceive its surroundings and also its internal state. The richness of the information about the surrounding world that the agent can obtain depends just on the type and number of sensors. Layers should know all types of possible sensors in order to be able to fill them with data. This means that creating a new layer necessarily also requires the creation of adequate sensors and, conversely, when adding a new sensor it is also necessary to alter the layers so that they are able to fill it. Sensors are the part of an environment that contains exact data (a numerical value). It is not always desirable to provide this crisp data to the agent's mind. Creatures in nature are also not able to perceive for example "that the object is 35.56 meters from them". Rather, they are able to perceive relative distance (closer/further) or inprecise data (close/mid range/far). They can attempt to estimate the value based on their experience (it might be 30 to 40 meters). To simulate this kind of perception in an agent's world we want to implement fuzzy information rather than crisp values. This can be done by filtering the exact floating point value to a fuzzy value after it has been obtained by the sensor.

2.6.2 Agent's effectors

Effectors enable an agent to interact with its surroundings. In our simulated world we use simplified effectors. For example we use effectors of motion which can move agents in a certain direction at a certain speed. Of course we could go into greater detail and implement effectors such as leg or wheel, but this would distract us (by solving inverse and forward kinematics tasks, friction, etc.) from our subject of research – behavior. We do not require this level of detail, but we do not run away from it. The possibility to implement it is open. In the field of effectors there is space for improvement. Instead of using effectors as part of the environment and controlling their action, we implemented them only as the action result. In the movement example above, we move the agent from one position to another, instead of sending a signal to the agent's locomotion system.

2.7 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 Fig. 5). Communication on the agent level means sending a message from one agent to another, or to a group of agents. Here we would like to let the layers decide who receives the information and who not. We are trying to reflect real world behavior, where information is carried via various media and can be received by various entities based on their sensor capabilities. When one agent wants

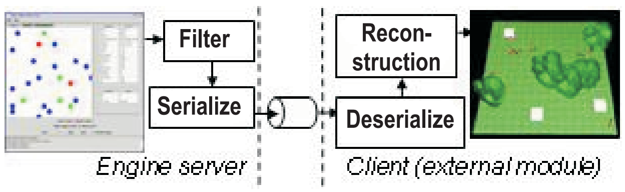


Fig. 5: Block scheme of communication to external client

to send a message (tell something) to another agent it will use its effectors and certain media of communication (an acoustic wave, for example). The information can be then received not only by the addressee but also by another agent who is within the range of the signal (even if it did not request this information). The interesting thing about this is that it can disregard unwanted information, or make use of it for its own purposes.

3 Simulations and results

We mentioned above that there are two components to run the simulation. The first of these is the environment described here, while the second is the control of the agent itself (agent mind). Several agent behavior control architectures have been introduced. The basis for the agent architecture was designed by D. Kadleček [2, 3]. This agent architecture with was redesigned for the WAL environment by K. Kohout in [1]. Several simulations, including the Lotka-Volterra system (also known as predator-prey system, see Fig. 6), and several task oriented scenarios were tested. In one of the simulations a task was given to the agent. This means that the agent, in addition to assuring its own survival should complete a task. In our case the task was to deliver messages. This simulation tested whether the thresholds were set correctly. If the agent's "need" to fulfill the task was low, it focused almost only on its own survival (lazy agents). If the need was high, the agent was busy with his task and he fulfilled his survival needs only when necessary (hardworking agent). This simulation showed that various creature or human qualities can be reproduced in agents.

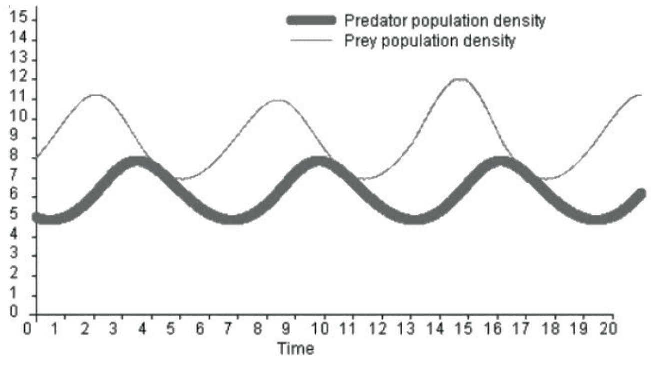


Fig. 6: Lotka-Volterra simulation results

3.1 Case study

In chapter 2.4 we mentioned the usefulness of external modules namely for simulation analysis. We wanted to test this on a case study performed while redesigning the agent to the WAL environment. The above mentioned VAT tool was used for the analysis. The simulation scenario involved a single agent which was intended to move an object between two

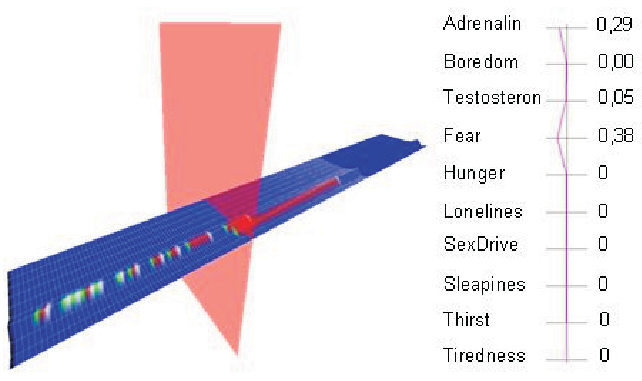


Fig. 7: Use case – simulation analysis

places. The agent had enough food and water to satisfy its needs. Fig. 7 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 brief look at the mesh indicates that there is something wrong with the simulation. Almost all the parameters are zero (the mesh is flat). This means that the agent is not hungry or thirsty; neither is it tired or sleepy. But we implemented and designed all these features. The reason for this could be a data export failure, a mistake in implementation of the inner agent vegetative block (part taking care of the "chemicals" in the agent's body) or bad initial configuration of the agent. Because we had run the simulation previously and the vegetative block had worked properly, there is no problem with the implementation itself. A brief check of the configuration showed that an excessively high value had been set to the time function for increasing/decreasing the chemicals. Fig. 8 shows the mesh after correction of the configuration mistake. The values now change with time as they should.

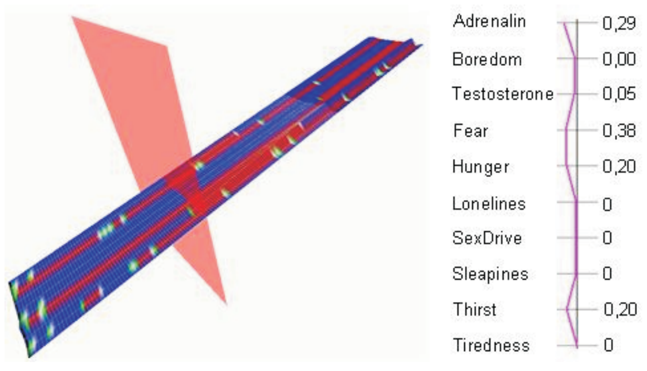


Fig. 8: Use case – simulation analysis correct setup

4 Conclusion

In this paper, we have described the simulation environment architecture for artificial creatures. It was used by several agent architectures. The first agent architecture tested in this environment was described in section 3 above. This agent architecture was superseded by several other architectures namely Lemming, designed by L. Foltýn [5], ACS proposed by M. Mach [6] and AnimatSim introduced by A. Svrček [7]. These architectures focused on different topics or different approaches to agent learning. AnimatSim focuses on reinforced learning; Lemming uses the TDIDT algorithm to create knowledge about the environment and to reason about it. ACS uses Hidden Markov models and reinforced learning. They have one thing in common. They were designed in various programming languages (C, Java, Matlab) but took WAL architecture into account and are capable of running in the WAL environment.

References

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

[2] Kadleček, D., Nahodil, P.: New Hybrid Architecture in Artificial Life Simulation. In: *Lecture Notes in Artificial Intelligence No. 2159*, Berlin: Springer Verlag, 2001. p. 143–146.

[3] Kadleček, D.: *Simulation of an Agent – Mobot in a Virtual Environment*. Diploma thesis. Prague: Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics, 2001.

[4] Kadleček, D., Řehoř. D., Nahodil, P., Slavik, P., Kohout, K.: Transparent visualization of multi-agent systems. In: *Proceedings of 4th International Carpathian Control Conference*, May 26–29, 2003, Vysoké Tatry. p. 723 – 726, ISBN 80-7099-509-2.

[5] 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.

[6] Mach, M.: *Data mining knowledge mechanism of an environment based on behavior and functionality of its partial objects*. Diploma thesis. Prague: Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics, 2005.

[7] Svrček, A.: *Selection and evaluation of robots-animates behavior*. Diploma thesis. Prague: Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics, 2005.

Ing. Karel Kohout
phone: +420 224 357 350
e-mail: kohoutk@fel.cvut.cz

Doc. Ing. Pavel Nahodil, CSc.
phone: +420 224 357 353
Fax: +420 224 353 677
e-mail: nahodil@felk.cvut.cz

Department of Cybernetics

Faculty fo Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13
121 35 Prague, Czech Republic