EcoSFERES: A Tool for the Design of Self-Organized Agent-Based Applications

Stéphane Doncieux[1], Samuel Landau[1,2], and Nicolas Guelfi[2]

[1] Université Pierre et Marie Curie, Laboratoire d'Informatique de Paris 6,
8 rue du Capitaine Scott, 75 015 Paris, France,
{Stephane.Doncieux,Samuel.Landau}@lip6.fr
[2] University of Luxembourg, Software Engineering Competence Center,
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg,
{Nicolas.Guelfi,Samuel.Landau}@uni.lu

Abstract. In this article we present EcoSFERES, a framework that helps applying learning and evolutionary algorithms to the design of multi-agent systems (MAS). This framework includes tightly linked abstractions for evolutionary computing and multi-agent based simulations, and additionally, agent learning policies can also be included and reused straightforwardly. In particular, EcoSFERES makes it possible to implement a variety of learning and evolutionary techniques in order to compare their results in identical conditions, and it also facilitates important changes in the simulation setup.
The main added values of this article are, on the one hand, to introduce the first development framework that provides a generic way of using evolutionary techniques for the design of MAS, and, on the other hand, to show that the tuning of MAS can be facilitated by a well designed development framework.

1 Introduction

The design of agent-based self-organising systems is now tackling problems of increasing difficulty, as the design of such systems leans toward involving more and more agents, with increasingly individual capacities, and increasingly interactions. Moreover, such agents are often immersed in a dynamical environment that changes in a more or less predictable manner, with consequences on the agents that may be impossible to foresee. Accordingly, it now becomes extremely difficult for a human to design such multi-agent systems and several researchers explore the alternative way of semi-automatic design, in which agents autonomously adapt to each other and to the environment, so that the overall system exhibits the sought behavior or characteristics.

Basically, such autonomous adaptation can be done through two major processes – learning and evolution [1] – the former having been more investigated than the latter as of today. In particular, the major issues raised by learning in a Multi-Agent System are now acknowledged [2], notably those entailed by the selection of an individual or a collective learning strategy [3]. Therefore, to facilitate the exploration of the potentialities and limits of semi-automatic design through evolution and learning, we developed a framework called EcoSFERES[3], which will be succinctly described in this article.

[3] EcoSFERES is an evolution of SFERES [4], which is the acronym of "SFERES is a Framework Enabling Research on Evolution and Simulation"

EcoSFERES implements a variety of evolutionary algorithms that are easy to substitute to each other. Problem dependant parts of the tool, *i.e.* every part that is involved in the evaluation of a potential solution, can also be replaced by another, thus taking advantage of the genericity of such algorithms, that were successfully used in numerous applications [5]. Each part of EcoSFERES, may it concern the selection algorithm, the genotype or the evaluation part can be changed with one another. Thus different genotypes can be tested and compared on a same problem in order to search for new algorithms, or different problems can be tackled with the same genotype and selection algorithm, thus exploiting available knowledge on new problems.

Section 2 will describe the features common to all evolutionary computing experiments that we used as a starting point in the design of EcoSFERES, while section 3 will describe the structure of EcoSFERES. Section 4 will sketch how some parts of the framework may be instantiated within the context of a simple agent-based self-organising application, while section 5 will discuss how the tuning of MAS can be facilitated by the EcoSFERES development framework.

2 Evolutionary Computing experiments: similarities and differences

Evolutionary computing experiments work on a population of individuals, that represent potential solutions. These solutions are progressively optimized thanks to two different genetic operators: mutation and crossing-over.

Each individual has one *genotype*, which is the structure optimized by evolution. The genotype describes the individual traits, and is the support for the hereditary transmission. A genotype can be instantiated in various structures, depending on the application. Such a genotype may be a binary string as in canonical Genetic Algorithms [6], or it may be a tree-like program as in canonical Genetic Programming [7]. Likewise, it may encode a direct representation of the system to optimize, or it may indirectly encode the sought solution, which will be generated after a developmental step. Genetic operators depend on the genotype that is used, they have to be defined at the same time as they directly act on the structure of the genotype.

The generation of a new population from an old one, which is the core of evolutionary algorithms, is actually independant from the genotype. It only needs to be able to rank individuals and generate new ones. We can further divide it into four different steps: the selection of the parents, the reproduction, the evaluation and the selection of the survivors. Every evolutionary algorithm defines its own variant of these steps, but they are generally used in this order.

There are however evolutionary computing applications (*e.g.* Echo[8]) where these operations (parents selection, reproduction, evaluation and survivors selection) do not take place in sequence, but all at the same time, as in an ecosystem model: individual “live” in the environment, reproduce, and die. These applications are often used to model and observe the evolution of self-organizing dynamical distributed systems, that are subject to changes according to environmental selection pressure.

The evaluation part, which is crucial for evolutionary experiments, is the only part that is entirely problem dependant. If evolutionary methods can be used to optimize

functions, they can also be used to evolve the behaviors of agents. The main difference between the two situations lies in the dependence of the agent's behavior to spatial and temporal contexts. The fitness of an agent acting in response to sensory stimuli and to its history cannot be evaluated in a one-step function evaluation, and the corresponding behavior must be assessed during a minimal period of time, long enough to be representative of the agent's capacities. This entails using a simulation of the agent and its environment, or an interface to a real agent like a robot, in combination with the evolutionary algorithm.

3 EcoSFERES structure

In EcoSFERES, a Simulation package, an Evolution package, an Environment package and an Agent package are tightly coupled to simulate and let evolve the behavior of multiple agents (see figure 1). Although some frameworks solely devoted to evolution [9] or to multi-agent simulation [10] already exist, no other framework, as far as we know, combines generic evolutionary computing traits with the design of multi-agent systems.

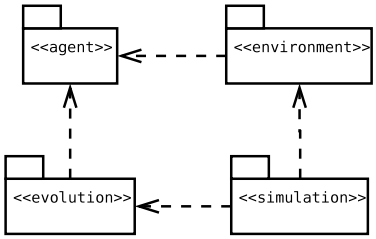

Fig. 1. Overview of EcoSFERES packages.

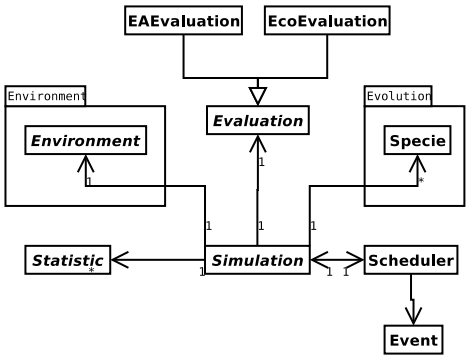

Fig. 2. Simulation Package.

3.1 Simulation Package

The Multi-Agent Simulation part (see figure 2) allows the simulation of Multi-Agent Systems, where agents can be tuned by evolution, by hand, or in both ways.

The **Simulation** class is at the core of the simulation and schedules the use of evolution with regards to simulation timesteps. This is a generic class that is sufficient for the vast majority of applications. The implementation of either a classical, generational Evolutionary Algorithm, or a non-generational "ecosystem-like" Evolutionary Simulation, relies on the kind of Evaluation and evolutionary operations that were chosen. The control of events occurring during the simulation is under the responsability of the

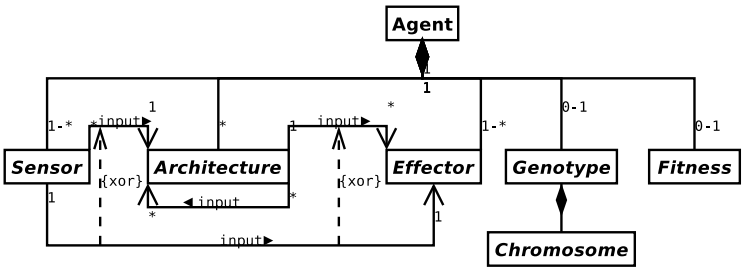

Fig. 3. Agent Package.

Scheduler. It can handle punctual or repetitive events and triggered events can be any method of any class of the system.

The **Evaluation** class controls the policy of the evaluation of the agents during a simulation cycle. Before or during the simulation, Evaluation sets the EvaluationSet of Agents for each Specie (see figure 4), which corresponds to the set of Agents that are currently evaluated.

3.2 Agent Package

The simulation includes a list of **Agents**. Agents can be predefined or tuned by evolution and, in the course of an experiment, they can be dynamically added or removed.

Each agent is made up of **Sensors**, **Architectures** and **Effectors** that can be linked together as shown on figure 3. Sensors provides informations to the agent that are relevant to the behavior to be exhibited. It may concern informations about agent's own state, about the environment or about other agents. Effectors correspond to the active part of the agent relative to the environment. Any part of the agent that will have an effect on the environment will be derived from this class. Architecture contains the control part of the agent. From sensory values, it has to decide the behavior to exhibit and then transmit corresponding orders to its outputs, may it be Effectors or another Architectures. The whole control system of the agent is then a collection of such Architectures, appropriately linked together and to Sensors and Effectors. Sensors and Effectors are dependant on a given Environment, but not Architectures that can be reused from one Environment to another.

Every component of an agent can be designed by evolution, but generally only the Architecture part is concerned.

As mentioned previously, Sensors and Effectors are the only interface between the Architectures and the simulated world. It should be noted that if Sensors and Effectors are generally linked to a simulated environment, it is possible to use them as an interface with "real" sensors and actuators (*e.g.* with another information system or with a real robot). This way, it is possible to evolve a controller either directly on a real robot, or on a simulated robot. In the latter case, it is also possible, at the end of the simulated evolution, to replace the virtual sensors and effectors by real ones and to test the solution thus discovered on a real robot.

In EcoSFERES, a population is made of agents the structure of which is given in figure 3. An agent's **Genotype** contains the information that is manipulated by the evolutionary process together with the genetic operators that will modify it, *e.g.* mutation and crossing-over operators. It is made of chromosomes the structures of which are easily tailored to the problem to solve. In particular, as already mentioned, such structures may be simple strings of bits, or tree-like programs, for instance. A Genotype may even be a set of such structures in cases where several parts of an Agent are to be simultaneously designed. This is the case in some experiments with so-called animats [11], when evolution has to design both the morphology and the nervous system of a simulated animal or a real robot [12, 13]. In such circumstances, a robot's morphology may be encoded in a first chromosome as a string of bits, while the nervous system may be encoded in a second chromosome as a developmental program that generates a neural network controlling the animat.

To be evaluated a genotype must be first decoded and transformed into a phenotype that will be used for the evaluation. Then the performance of the phenotype is determined thanks to a **Fitness** function provided by the experimenter. The Environment and the Fitness are separated within EcoSFERES, thus making it possible to test different variations of the Fitness on the same Environment.

The computation method associated with a fitness function, and the fitness value it generates, are grouped in a same class. The fitness value assesses the performance of an individual with respect to the objectives of the experiment. It is used to rank the phenotypes and to apply selection algorithms on the corresponding genotypes. Although it may be of any type, as long as an order can be defined on it, the most commonly used value is a real value.

Fitness objective is only to rank individuals for the selection algorithms. It is thus completely independent from the other part of the evolutionary algorithm, thus facilitating the design step of such a function, which plays a critical role in evolutionary algorithms.

3.3 Evolution Package

The general organization of the Evolution module in EcoSFERES is described on figure 4.

The evolutionary computing operations – parents selection, survivors selection and reproduction – are reified into independent abstract classes that will implement each of the corresponding aspect of the evolutionary algorithm. The selection algorithms, both parents and survivors selection, act as a filter on a population to keep individuals that have exhibited a good behavior according to the fitness function. The reproduction class generates new individuals from models and from previously chosen parents. These three parts are associated with a **Specie** that represents a specific kind of Agents, following a given model.

This makes it possible to evolve multi-agent systems with heterogeneous agents that may have different objectives. For instance, several types of individuals may evolve at the same time on concurrent tasks as in co-evolution experiments involving both preys and predators[14, 15], where the goal is to simultaneously evolve preys that must escape from the predators and predators that must catch the preys. To simulate such sys-

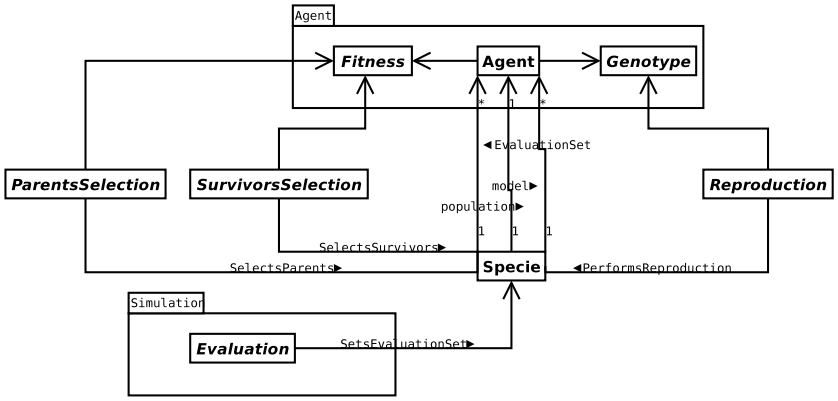

Fig. 4. The Evolution package of EcoSFERES.

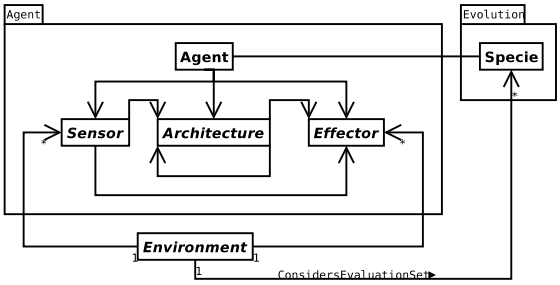

Fig. 5. Environment Package.

tems, two Species instantiations are required because agents are heterogeneous and do not share the same goals, and because the selection algorithm must perform a selection of opponents during evaluation. Increasing the complexity of such an experiment is straightforward in EcoSFERES, adding for instance a third species – like a super-predator – requires only to add such a species and to adapt the selection mechanism to take it into account.

3.4 Environment Package

The environment package is only composed of an Environment class, that reifies the environment of the Agents. During the evaluation phases, the Environment only takes into account the EvaluationSet of Agents for each Specie of the system. The Effectors status of these Agents are read by the Environment and taken into account in order to update information about the Agent (*e.g.* its position in a physical world), and then update the Agents' Sensors status, according to the environmental perceptions of the Agent.

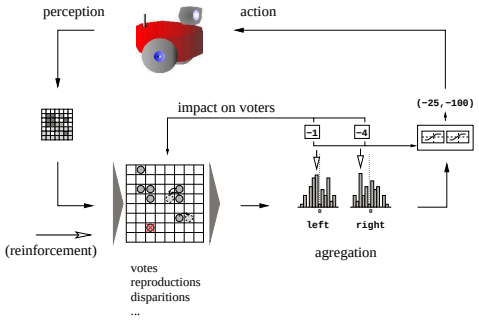

Fig. 6. Overview of the system. The robot visual perceptions is grabbed as a pixel matrix from the camera, and is used by the system to decide what to do next, by aggregating agents votes. The human teacher interacts with the system by distributing scalar rewards, that have an influence on the survival of the agents that participated in the previous actions.

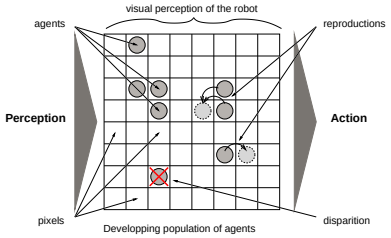

Fig. 7. Illustration of the environment and the agents. Agents live, reproduce and die in the environment formed by the pixel matrix. They are fed thanks to the external scalar reward, distributed by the human teacher. Their actions consist in voting for an action for the robot to take.

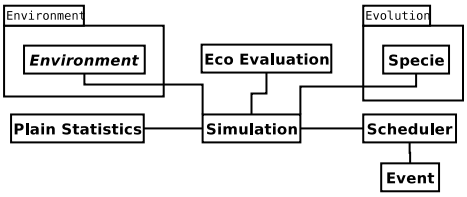

Fig. 8. ATNoCells Simulation Package.

4 Example

This section will describe an instantiation of several elements of EcoSFERES within the context of a simple self-organising multi-agent based application. In this application, called ATNoCells [16], evolution has to design the behaviors of several lightweight agents, that collectively control a Pioneer P2DX mobile robot by analysing the visual information grabbed from its camera, and by outputing speed instructions for each of the two wheels of the robot. This general functioning is sketched on figure 6. The goal is to endow the robot with the ability to learn a behavior by demonstration, as shown by a human teacher.

4.1 ATNoCells Simulation Package

The Simulation used is the generic one (see section 3.1), with the EcoEvaluation. The EcoEvaluation just imposes to every Specie an EvaluationSet consisting in all the available population. The Scheduler and Event classes are also the generic ones. Finally, the

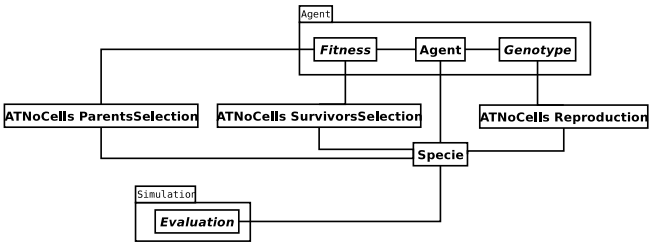

Fig. 9. ATNoCells Evolution package.

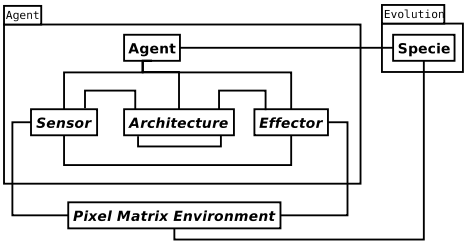

Fig. 10. ATNoCells Environment Package.

Statistics that are used are plain, common statistics, like the Best Fitness Statistic that records the best fitness.

4.2 ATNoCells Evolution Package

The Specie class used is the generic one, while, on the other hand, all the evolutionary operations are specialized in order to implement the reinforcement learning and evolutionary algorithms as detailed in [16]. In particular, the parents selection take into account the "age" of agents and their fitness; the reproduction generates children in empty cells next to parents' ones; and the survivors selection takes into account the fitness, and a level of participation of the agents in the previous collective decision taken.

4.3 ATNoCells Environment Package

The environment is specific to the application, since it must feed agents sensors with the pixel value of their position cell, grabbed from the camera of the robot, and must aggregate the agents votes in order for the robot to take action.

4.4 ATNoCells Agent Package

The agent used in this application just has some attributes that are used for the reinforcement learning and evolutionary algorithm (*e.g.* participation measure, age). The sensors

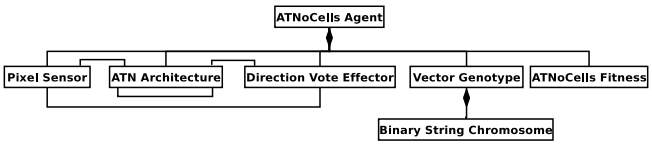

Fig. 11. ATNoCells Agent Package.

and effectors are also specialized for the application, the sensor being able to read the pixel value associated to the cell the agent is located on, and the effector transmitting information of the direction votes for this agent. The architecture is more generic. It is based on a finite state automaton (namely, an ATN[4]), and was already used for other applications [17, 18]. Its reuse in different contexts is straightforward, as well as its decoding from the binary string chromosome (like in standard Genetic Algorithms). This decoding has been one of our points of interest in these previous works. The genotype used is a generic and standard one, a plain vector of chromosomes, and, finally, the fitness was designed according to the learning and evolutionary algorithms used.

5 Discussion

Most of the time, the targets of adaptive techniques will be the architectures of the agents, that govern their behaviors. However, it might happen that one rather wishes to tune a set of parameters or behaviors that control the *organizational* level of the agents [3]. In this case, the target of the adaptive techniques would rather be specialized instances of the species.

We will now describe how the tuning using adaptive techniques can be achieved within our framework, by describing how straightforward it is to implement them, or apply them to different experimental protocols. We emphasize that the reuse of an already available adaptive technique is straightforward, since it just involves modifying a XML configuration file[5].

5.1 The Use of Different Evolutionary Techniques

Once one has decided whether a generational or a non-generational algorithm should be used – and therefore, chosen to use the corresponding generic subclass of Simulation (see section 3.1) – the use of an evolutionary technique is most of the time a matter of derivating some classes.

If a non-generational algorithm was chosen, the generic EcoEvaluation will do, but in the case of a generational algorithm, the generic EAEvaluation might need to be sligthly modified (*e.g.* for algorithms using tournament evaluations). This is done by just modifying the method that sets the EvaluationSet of the species for each evaluation cycle.

[4] Augmented Transition Network

[5] see http://www-poleia.lip6.fr/~landau/EcoSFERES/ecosferes.dtd for its DTD

With respect to the evolutionary operations, since they have been reified and are connected to the remaining of the framework in a generic fashion, the implementation of a different one just consists in deriving the corresponding class (*e.g.* using an elitist mating algorithm rather than a roulette wheel).

Finally, if the focus is rather on the genetic code, the only classes that need to be derived are the specific kind of chromosomes and architectures to be used (*e.g.* our Stack-Based Gene Expression genetic code [19]), and eventually a specific genome if the generic VectorGenome is not satisfactory.

5.2 The Use of Different Learning Techniques

We stress that the use of evolutionary techniques is not mandatory in EcoSFERES. In this case, the EcoEvaluation and dumb evolution operation classes are used, and no evolution occurs.

In particular, learning techniques can be applied (be it in conjunction with evolution or not). As a matter of fact, learning at the architecture level is achieved by implementing a virtual method that updates architecture parameter values, with regard to the information available to the owning agent (if needed this method can be disabled, for example if a satisfactory level of performances is reached). This way, most of the individual learning techniques can be implemented. Learning can also be applied at the organizational level, to whole populations, by deriving an appropriate specie. At each simulation time step, the specie would update the relevant parameters according to the chosen learning algorithm.

5.3 Application to Different Multi-Agent Simulations

The application of the evolutionary and learning techniques are quite independent from the experimental protocol (environment and simulation settings). Therefore, the application of these techniques to different multi-agent simulations is, once the supporting classes are implemented, just a matter of setting a configuration file.

Besides, the implementation of a new experimental protocol most of the time relies on the implementation of an adequate environment and the corresponding sensors and effectors. Sometimes, deriving a specific agent can be required.

5.4 Benefits of the framework for the tuning of MAS

The developpement framework might be used as follow for the tuning of a MAS. During a preliminary study, the behaviors of the agents might be designed at hand and fixed, so that one can inspect the problem to be solved, and determine where adaptive and self-organized processes should intervene. This first phase would involve no evolutionary computing, and would only use the multi-agent simulation subset of EcoSFERES. In particular, the agent architectures are not subject to modifications by any evolutionary or learning algorithm. Thanks to the interactive mode of EcoSFERES, the behavior of the system can be observed, paused, restarted, executed step-by-step or even modified online. On the other hand, the statistics also help the designer to have a clue about the relevance of the different set of parameter values he is trying.

Then, having acquired some experimental knowledge on the cause and consequence of the complexity of the collective task to be realized, the designer can use or reuse adaptive techniques by just specifying in the configuration file which architectures should become adaptive thanks to which evolutionary and/or learning technique. Free parameters governing the collective behavior of the system are then modified during this second phase, according to the algorithms chosen, and their effect can be evaluated thanks to the same statistics set used during phase one.

6 Conclusion

We have presented the EcoSFERES framework, which provides a generic way of using evolutionary and learning techniques to multi-agent based simulations. This tool increases the flexibility of use of different adaptive techniques for the tuning of multi-agent systems, and is therefore a good candidate as a tool for the design of agent-based self-organising systems.

We have exploited the implementation of SFERES, the preliminary version of this framework for more than 4 years at the AnimatLab team of the LIP6 and have started to exploit the EcoSFERES version. It has supported many of the published research on adaptive systems at the AnimatLab, *e.g.* the ATNoSFERES project[17] (study of a genetic encoding) and the ALPhA project [20] (study of the control of a lenticular blimp by evolved neural networks). It was also used to design controllers for a simulated helicopter using neural networks [21] or using classical PID-like controllers together with optical flow exploitation to implement an obstacle avoidance strategy [22]. Many generic components were already developped. An extension of the framework to the design of neural networks in which the ModNet encoding has been implemented [21], an environment for 3D-realistic engine (taking physics into account) based on the ODE engine[6] and several simulators of flying robots, ranging from blimps to helicopters, are among the modules that are available now.

References

1. Meyer, J.A.: Artificial life and the animat approach to artificial intelligence. In Boden, M., ed.: Artificial Intelligence. Academic Press (1996)
2. Weiss, G.M., Sen, S., eds.: Adaptation and Learning in Multi-Agent Systems. Lecture Notes in Artificial Intelligence. Springer Verlag. Berlin (1996)
3. Drogoul, A., Zucker, J.D.: Methodological issues for designing multi-agent systems with machine learning techniques: Capitalizing experiences from the robocup challenge. Technical Report LIP6 1998/041, LIP6 (1998)
4. Landau, S., Doncieux, S., Drogoul, A., Meyer, J.A.: SFERES: un framework pour la conception de systèmes multi-agents adaptatifs. Technique et Science Informatiques **21** (2002) 427–446
5. Tettamanzi, A., Tomassini, M.: Evolutionary Algorithms and their Applications. In: Bio-Inspired Computing Machines. Presses Polytechniques et Universitaires Romandes (1998)

[6] http://q12.org/ode/

6. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)
7. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press (1992)
8. Holland, J.H.: Adaptation in Natural and Artificial Systems, 2nd Ed. MIT Press (1992)
9. Wall, M.: Galib. http://lancet.mit.edu/ga/ (2000)
10. Gutknecht, O., Ferber, J.: Madkit (a multi-agent development kit), http://www.madkit.org (1997)
11. Meyer, J.A., Berthoz, A., Floreano, D., Roitblat, H., Stewart Wilson, W., eds.: From Animals to Animats 6, Proceedings of the sixth international conference on Simulation of Adaptive Behavior, MIT Press/ISAB (2000)
12. Sims, K.: Evolving 3d morphology and behavior by competition. In Brooks, R., Maes, P., eds.: Proceedings of the Fourth International Workshop on Artificial Life, The MIT Press (1994)
13. Harvey, I., Husbands, P., Cliff, D.: Seeing the light: artificial evolution; real vision. In Cliff, D., Husbands, P., Meyer, J.A., Wilson, S., eds.: From Animals to Animats 3, Proceedings of the third international conference on Simulation of Adaptive Behavior, MIT Press/Bradford Books (1994)
14. Haynes, T., Wainwright, R., Sen, S.: Evolving cooperation strategies. Technical Report UTULSA-MCS-94-10, The University of Tulsa (1994)
15. Floreano, D., Nolfi, S., Mondada, F.: Competitive co-evolutionary robotics: From theory to practice. In Pfeifer, R., Blumberg, B., Meyer, J.A., Wilson, S.W., eds.: From Animals to Animats: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior, MIT Press (1998)
16. Landau, S.: Des AG vers la GA – Sélection Darwinienne et Systèmes Multi-Agents. Phd thesis, Université Paris VI (2003)
17. Landau, S., Picault, S., Drogoul, A.: ATNoSFERES : a Model for Evolutive Agent Behaviors. In: Proc. of AISB'01 symposium on Adaptive Agents and Multi-agent systems. (2001) 95–99
18. Landau, S., Sigaud, O.: A comparison between atnosferes and lcss on non-markov problems *(accepted)*. Information Sciences (2004)
19. Landau, S., Picault, S.: Stack-Based Gene Expression. Lip6 technical report 2002/011, LIP6, Paris (2002)
20. Doncieux, S., Meyer, J.A.: Evolving neural networks for the control of a lenticular blimp. In Raidl, G.R., Cagnoni, S., Romero Cardalda, J.J., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E., Meyer, J.A., Middendorf, M., eds.: Applications of Evolutionary Computing, EvoWorkshops2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM. (2003) 626–637
21. Doncieux, S.: Évolution de contrôleurs neuronaux pour animats volants : méthodologie et applications. PhD thesis, LIP6/AnimatLab, Université Pierre et Marie Curie, Paris, France (2003)
22. Muratet, L., Doncieux, S., Meyer, J.A.: A biomimetic reactive navigation system using the optical flow for a rotary-wing uav in urban environment. In: Proceedings of ISR2004. (2004)