Lifelong Learning by Evolution in Robotics: Bridging the Gap from Theory to Reality

B. Santos-Diez, F. Bellas, *Member, IEEE,* A. Faiña, R.J. Duro, *Senior Member, IEEE*

***Abstract*—In this paper, we describe the practical computational implementation of the Multilevel Darwinist Brain (MDB) cognitive architecture. The MDB follows a developmental robotics approach, where open-ended, autonomous learning systems that continually adapt to their environment are designed. Consequently, the acquisition of knowledge, represented by models, required to select robot actions is obtained through the interaction with the environment. The computational implementation of this type of architecture implies several aspects that are very relevant: evolutionary algorithms that must support a real time operation architecture, processes running in different time scales, memory elements that are continuously interplaying, and so on. In this work, we will focus on these aspects, describing the current implementation of the MDB and its successful performance in a real robotics learning scenario.**

I. Introduction

The application of evolutionary techniques in autonomous robotics has been a very prolific topic in the last twenty years [1]. But most of these approaches, instead of designing architectures that provide robots with real autonomy, were focused on designing controllers for particular tasks, where the acquisition of knowledge was very limited, mainly obtained in a controlled fashion, and the problems of adaptation to the dynamics of the real environments were simplified [2][3][4].

Nowadays, we know that a control system such as the one that is necessary for realistic autonomy is something that goes beyond traditional control in terms of the specifications or requirements. These additional requirements imply the ability to learn the control function from scratch, the ability to change or adapt it to new circumstances, the ability to interact with the world in real time while performing the aforementioned processes and, in some instances, even to change the objectives that guide the control system. As in the case of animals, open-ended lifelong learning systems present the potential of achieving a more realistic level of autonomy.

Starting from this background, in the early 2000's Weng proposed an autonomous mental development (AMD) [5][6] line, particularized to robotics through the cognitive developmental robotics (CDR) [7] approach. The key aspect of CDR is that the control structure should reflect the robot's own process of understanding through lifelong interactions with the environment. We have followed this CDR approach in the design of the Multilevel Darwinist Brain cognitive architecture, but we have addressed the problem making use of some of the concepts of traditional cognition, introducing ontogenetic evolutionary processes for the on-line adaptation of the knowledge bearing structures.

Therefore, our approach has been to obtain a developmental control architecture for real robots that uses evolutionary algorithms in real time. In this line, the number of examples found in literature is very small. We can point out [8], where the authors present an approach called "embodied evolution" in which a group of robots can improve in real time a basic set of behaviors through evolution and use a system whereby they mate to transmit genetic information. In [9][10] the authors resort to competitive coevolution to make two physical robots adapt their behaviors in real time.

The main drawback of evolutionary approaches is their computational cost. Therefore, the computational aspects of the implementation must be carefully analyzed and the main objective of this paper is to discuss these aspects within the Multilevel Darwinist Brain (MDB) cognitive architecture. We will first describe the architecture from a theoretical point of view to better understand all the implementation decisions that have been taken, and that have resulted in successful performance in real robotics learning tasks.

II. Cognitive architecture for Lifelong Learning

Cognition may be defined as "The mental process of knowing, including aspects such as awareness, perception, reasoning, and judgment". Designing a cognitive architecture for a robot requires dividing the problem into simple elements that can be studied separately. On one hand, we have the mental process of knowing, that is, in more mathematical terms, of extracting models from data can be employed in the process of decision making so that appropriate actions may be taken as a function of sensing and motivation. On the other, we have the decision making process itself, and, in robotics, a decision is always related to an action or sequence of actions. In a sense, the models must be used in order to decide the appropriate actions so as to fulfill the robot's motivations. It is in how the model making and the action determination processes take place that cognitive architectures differ from each other.

B. Santos-Diez, F. Bellas. A. Faiña, and R.J. Duro are with the Integrated Group for Engineering Research in the University of Coruna, Spain (e-mail: borja.santosdiez@udc.es, fran@udc.es, afaina@udc.es, richard@udc.es).

A. *A little formalism*

The particular cognitive model that formalizes the previous ideas starts from the premise that, to carry out any task, a *motivation* (defined as the need or desire that makes an robot act) must exist that guides its behavior. Therefore, the *satisfaction* of the robot can be defined as a magnitude or vector that represents the degree of fulfilment of the motivation or motivations of the robot. The final objective of the cognitive architecture is to explore the possible action space in order to maximize the resulting satisfaction.

To obtain a system that can be applied in reality implies internally (without interaction with the environment) deciding on the best action to apply within a time span that permits real time interaction with the environment. To internally evaluate candidate actions some theoretical functions are required that must be somehow obtained. These functions correspond to what are traditionally called:

- *World model (W):* function that relates the *external perceptions* before and after applying an action.
- *Internal model (I):* function that relates the *internal perceptions* before and after applying an action.
- *Satisfaction model (S):* function that provides a predicted *satisfaction* from predicted perceptions provided by the World and Internal models.

As commented before, the main starting point in the design of a developmental cognitive architecture was that the acquisition of knowledge should be automatic, so we establish that the three models *W*, *I* and *S* must be obtained during execution time as the robot interacts with the world. To perform this modelling process, information can be extracted from the real data the robot has after each interaction with the environment. Hereafter, these data will be called *action-perception pairs* and are made up of the sensorial data in instant t, the action applied in instant t, the sensorial data in instant t+1 and the satisfaction in t+1.

Summarizing, for every interaction of the robot with its environment, functions W, I and S must be modelled using the information in the action-perception pairs and the best action for the robot selected using the models available at that time and maximizing predicted satisfaction.

To create models is to try to minimize the difference between the reality that is being modeled and the predictions provided by the model. Consequently, a cognitive architecture must involve some type of optimization strategy or algorithm. As the search spaces are not really known beforehand and are usually very complex, the MDB is based on using one of the most powerful stochastic multipoint optimization techniques: artificial evolution.

B. *The Multilevel Darwinist Brain architecture*

The Multilevel Darwinist Brain (MDB) is a general Cognitive Architecture first presented in [11] that permits an automatic acquisition of knowledge in a real robot through the interaction with its environment, making it capable of adapting its behaviour to the context and objectives. The inspiration for the approach followed comes from the classical bio-psychological theories by Changeaux [12], Conrad [13] and Edelman [14] in the field of cognitive science relating the brain and its operation through a Darwinist process. All of these theories lead to the same concept of cognitive structure based on the brain adapting its neural connections in real time through evolutionary or selectionist processes. That is, the Multilevel Darwinist Brain uses *artificial evolution* as the tool for incorporating model learning.

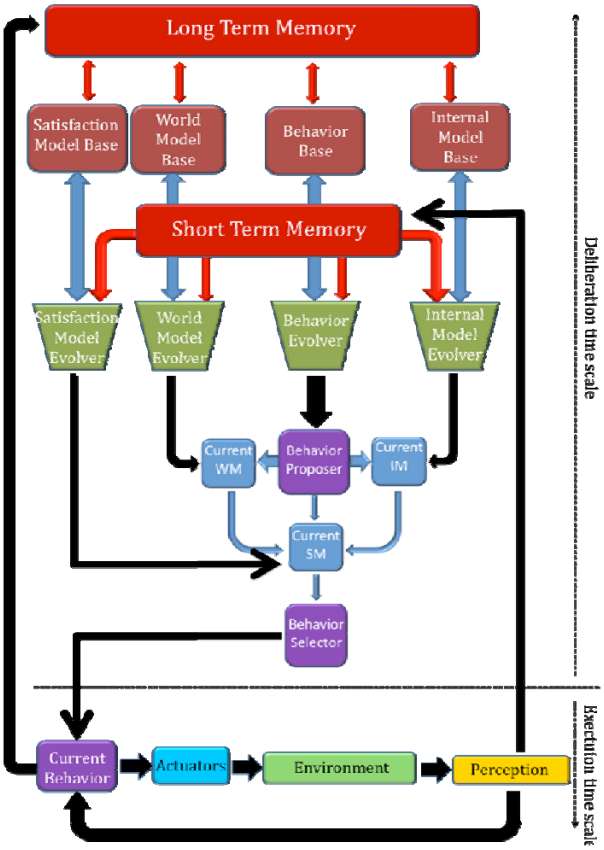

Fig. 1. MDB block diagram

Fig. 1 displays a block diagram of the MDB. It follows the previous cognitive model, which has been generalized by basically adding two new aspects: *behavior structures* that generalize the concept of action and *memory elements*, basically a short-term and a long-term memory required in the learning processes.

As shown in Fig. 1, the MDB is structured in two different time scales, one devoted to the execution of the actions in the environment (reactive part) and the other devoted to the learning of the models and behaviors (deliberative part). We can describe the operation of the MDB in these two scales:

1. *Execution time scale (reactive):* which continuously repeats these two steps in a sequential manner:
   - There is a *current behavior*, selected in the

deliberative process, which provides the action the robot must apply in a given instant of time as a function of the perceptual values it has as inputs.

- The selected action is applied to the *environment* through the *actuators* of the robot obtaining a new *perception*, that is, new sensing values.

2. *Deliberation time scale:* the processes included here can also be taken as operating concurrently and in different time scales, thus, *they are not sequential*:
   - The acting and sensing values obtained after the execution of an action in the environment in the execution time scale provide a new action-perception pair that is stored in a *Short-Term Memory* (STM).
   - The evolutionary model learning processes (for world, internal and satisfaction models) try to find functions that generalize the real samples stored in the STM. Each evolutionary process has been represented by two blocks in Fig. 1, one representing the evolution itself (labelled *evolver*) and the other one representing the population for each evolution (labelled *base*).

     Each evolutionary process will be continuously running and producing new models in their own time depending on the complexity of the models required and their practical implementation will require avoiding incoherences during the interplay of the elements involved in the real time operation.
   - The best models in a given instant of time are taken as *current world, internal and satisfaction models* and are used by the *behavior evolver* to select the best behavior (the one that produces the highest predicted satisfaction).

     The blocks labelled Current WM, Current IM, Current SM and Behavior Proposer are really included in the individual's evaluation stage within the behavior evolver.
   - The behavior evolver is continuously proposing new behaviors that are better adapted to the STM contents. Upon request, the behavior selector provides the best one to the reactive part of the MDB, the *current behavior* block in Fig. 1. This behaviour should be better adapted to the current STM and, consequently, to the current "reality" of the robot.
   - The block labelled *Long-Term Memory* (LTM) in Fig. 1 stores those models and behaviors that have provided successful and stable results in their application to a given task in order to be reused as seeds in the population for new learning processes.

Each interaction of the robot with the environment (*iteration*) is taken as the basic time unit within the MDB. As more iterations take place, the MDB acquires more information from the real environment and thus the model learning processes have more information available and the models, and consequently the behaviors obtained from them, are more reliable, leading to more appropriate robot actions.

C. Lifelong learning by evolution

The main difference of the MDB with respect to other architectures lays in the way the process of modelling functions W, I and S is carried out. Evolutionary Algorithms are the basis of this architecture and as representation for the models we have selected Artificial Neural Networks. Thus, the acquisition of knowledge in the MDB is basically a neuroevolutionary process.

The modelling here is not an optimization process but a learning process taking into account that we seek the best generalization for all times, or, at least, an extended period of time, which is different from minimizing an error function in a given instant t. Consequently, the modelling technique selected must allow for gradual application, as the information is known progressively and in real time. Evolutionary techniques permit gradual learning by controlling the number of generations of evolution for a given content of the STM. Thus, if evolutions last just a few generations per iteration (interactions with the environment), gradual learning by all the individuals is achieved.

To obtain general modelling properties in the MDB, the population of the evolutionary algorithms must be preserved between iterations, leading to a sort of inertia learning effect where what is being learnt are not the particular contents of the STM in a given instant of time, but of sets of STMs that were previously seen. In addition, the dynamics of the real environments where the MDB will be applied imply that the architecture must be intrinsically adaptive. This strategy of evolving for a few generations and preserving populations between iterations permits a quick adaptation of models to the dynamics of the environment, as we have a collection of possible solutions in the populations that can be easily adapted to the new situation.

In the case of behaviors, in the current version of the MDB they are also represented by ANNs, and consequently, they can be viewed as neural controllers that provide the action the robot must apply in the environment according to the sensorial inputs.

D. Learning from memories

The management of the Short Term Memory is critical in the real time learning processes within the MDB because the quality of the learned models depends on what is stored in this memory and the way it changes. The data stored in the STM are acquired in real time as the system interacts with the environment and, obviously, it is not practical or even useful, to store all the samples acquired in the robot's lifetime. A dynamic replacement strategy was designed that labels the samples using four basic features (distance, complexity, initial relevance and time) related to the saliency of the data and temporal relevance.

The Long Term Memory is a higher level memory element. From a psychological point of view, the LTM stores the knowledge acquired by the robot during its lifetime. This knowledge is represented in the MDB as

models (world, internal and satisfaction models), behaviors, and their context. The models and behaviors stored in the LTM in a given instant of time are introduced in the evolving populations as seeds so that if the robot returns to a previously learnt situation (or a similar one), the model will be present in the population and the prediction will be accurate soon. For a detailed description of the operation of the STM and LTM and the interplay between them see [15].

III. Computational implementation

The computational implementation of the concepts presented above is the key to their applicability in real robotic systems. In what follows we are going to consider some of the main elements in the current JAVA implementation of the MDB.

We can distinguish 4 basic packages that make up the computational core of the MDB: robot, evolutionary algorithm, model and memory.

1) Robot

As a first design requirement, already in the initial versions of the MDB, we imposed that the basic onboard processor of the real robots (usually a microcontroller) should be used just to execute actions and to collect the sensorial information in real time. Regarding the two different time scales shown in Fig. 1, this means that the onboard processor is in charge of the execution time scale elements. Thus, the deliberative part of the MDB is always executed in a separate processor (on or off the robot), and the communications between these two structures are carried out using the standard TCP/IP protocol.

The second basic design requirement related with the robot is that the MDB should be as independent from the particular hardware as possible. That is, we assume that the MDB receives sensorial information and provides actions to be applied in a robot, but its core cannot include any particularity about the robot. To this end, we have taken inspiration from the Player/Stage project [16], which uses a network server for robot control that provides an interface to the robot's sensors and actuators over the IP network.

The robot package includes all the classes implementing the previous two requirements. The designer must create a simple configuration file including a description of the robot sensors and actuators and the IP port where it is connected. On the other hand, for each particular robot or simulator, an interface program must be developed to provide sensorial information and to capture the actions obtained by the MDB using the standard TCP/IP protocol. The onboard processor's computational load is minimized and, consequently, these data are always in raw format. If any kind of processing is required, it will be carried out by the perception package.

2) Evolutionary Algorithm

This package includes all the classes in charge of executing the evolutionary processes for the models and behaviors, which constitute the core of the architecture. One of the main drawbacks of the application of evolutionary algorithms in real robotics is the computational cost they imply, that makes them apparently unsuitable for real time operation. We have addressed this problem through the following design decisions:

- First, as commented in the MDB description, the evolution of the models and behaviors only last a few generations per iteration in order to obtain a smooth learning curve. This obviously reduces the computational cost in between interactions with the world and makes its real time implementation feasible.
- In addition, the MDB is intrinsically concurrent and each evolutionary process is automatically assigned to a different processor when available: each evolutionary process is an independent thread forcing the use of a different processor. In the case of having a local area network, the threads may be executed in different computers over the network automatically.

The behavior evolution module uses the current models to evaluate candidate behaviors. Taking into account the distributed execution of each model evolution, we have implemented a coherence protocol that ensures an updated evaluation of the individuals, using always the most recent models. A similar procedure has been implemented for the update of the current behavior in the reactive part of the MDB, which is replaced every time a better one is obtained but ensuring a coherent action selection.

On the other hand, the MDB principles establish that it is independent from the particular type of evolutionary algorithm and artificial neural network. This is very easy to support with the object-oriented design of the MDB. In fact, we have tested the architecture with the different algorithms implemented in the JEAF library [17], by simply changing the class in the configuration file.

3) Model

To create a new experiment using the MDB, the first step is to set up the model configuration. This implies formally describing the world, internal and satisfaction models, that is, we must decide the knowledge representation within the architecture. This selection is very relevant because the success or failure of learning depends highly on the complexity of the models.

What the designer must do is simply indicate the inputs and outputs corresponding to each model. This information corresponds to the external and internal sensorial information and to the action space. We must take into account that the robot configuration file already includes all the real sensors and actuators, but here, as inputs and outputs to the models, we can use virtual sensors and actuators that process the raw data provided by the robot allowing the designer to use the sensorial information freely.

To automate the model configuration, we have developed a general procedure based on the division of independent sensorial information into different models. For example, in the case of the world models, if we have a robot with 4

infrared sensors and 2 light sensors, the MDB will create 2 models as a first approach. The first one relating the 4 infrared inputs in instant t and t+1and the action applied, and the second one relating the 2 light inputs with the action in the same way. This procedure has provided the best results in several experiments [18][19].

What is relevant for the computational implementation is that we could have several concurrent evolutionary processes for all the submodels making up the world, internal and satisfaction models as well as for the behaviors. As commented above, a mechanism has been implemented that automatically executes these concurrent processes in different, and even remote, threads.

4) Memory

The evolutionary processes in the MDB are continuously trying to model the STM action-perception pairs, which represent all the sensorial and acting information for each iteration of the architecture. As commented in section II, the replacement strategy of the STM determines the type of learning achieved, and it must be adjusted depending on the complexity of the model. Taking into account the previously explained subdivision into models, using a single STM for all the models is not feasible.

Hence, in the current implementation of the MDB, each model evolution has its own STM with its particular replacement strategy. All the classes required for this management are included in the memory package. The corresponding configuration file only requires establishing the type of replacement strategy, while the creation and execution of the STM is automatic.

Regarding the LTM, the design and implementation follows basically the same principles: there is a different LTM for each evolutionary process, which is executed concurrently with it.

To summarize this section, we want to highlight the following implementation aspects within the MDB, all related with the operation of the architecture in real robots:

- Object-oriented design and JAVA implementation.
- Hardware/simulator independence through the use of a TCP/IP middleware approach.
- Time scale independence: reactive elements are executed onboard and deliberative elements in remote or onboard computers through TCP/IP communication.
- Automatic concurrent execution of the evolutionary processes in remote processors.
- Automatic division of STM and LTM memories and execution according to the evolutionary processes.
- Persistent evolutionary processes that never stop although the fitness function can change.
- Easy integration with evolutionary and ANN libraries

IV. AN ILLUSTRATIVE EXAMPLE

The MDB architecture has been widely tested in real examples [18][19]. But here we want to illustrate the operation of the MDB implementation in terms of real operation. To this end, in this section we describe a real robot learning experiment that involves the Sony AIBO robot and a pink ball (Fig. 2). The goal is simple: the robot must learn to catch the ball as quickly as possible from its interaction with the environment through its sensors and actuators. To this end, we have followed a developmental approach where a teacher places the ball in the robot's surroundings. In the first iterations, the robot will lose the ball, so the teacher must place it again near it. Once the task is learned, the ball position is not so important because the robot has acquired the general skills.

Fig. 2. Behavior of the robot in the initial (left) and final (right) iterations of the experiment

Regarding the robot configuration, we are using camera data and head angle as sensorial information and a predefined gait for the actions the robot can execute, in this case, just moving. An IP port was assigned to the robot, and the MDB was run in a remote computer receiving e sensorial information and sending the actions through the network.

In terms of the models, as usual, they do not use the camera information directly, but processed in the perception package classes. The head angle is used directly and the gait of the robot is specified using linear and angular speeds. In particular, the information we are managing in the models is:

- *Distance to the ball:* calculated from the camera data.
- *Angle to the ball:* calculated from the camera data and the head angle.
- *Angular speed:* linear speed is fixed and the MDB must provide the angular speed required to reach the ball.

Following the previously commented procedure, we have created two different world models according to the different sensorial information they manage: one for distance prediction and another for angle prediction. In this case, the models will be represented by multilayer perceptrons, one with 3 inputs (distance, angle and angular speed in time t) and 1 output (predicted distance), and another one with the same 3 inputs and 1 output (predicted angle). The satisfaction model is another ANN with 2 inputs (predicted distance and angle) and one output (predicted satisfaction). As commented before, the goal for the robot is to catch the ball as soon as possible, so we have established that its motivation is to minimize the distance and angle (reaching the ball frontally) in the minimum number of iterations. As a consequence, the satisfaction value is simply the difference with respect to this motivation. In this example, we did not use internal models (see [19] for an example with them).

With this setup we have configured 3 concurrent evolutionary processes that were run in an Intel Core 2 Duo processor, using 3 concurrent threads. We have used the JEAF library and, in particular, the Differential Evolution evolutionary algorithm for the 3 cases because it provided the best results with a low population size. The STM was implemented using a purely temporal strategy (FIFO).

In this case, the behaviors are simple actions, so the behavior evolution is not required. The MDB operation is exactly the same, but instead of using an evolutionary process to obtain behaviors, we use a simple heuristic to select the action for each instant of time.

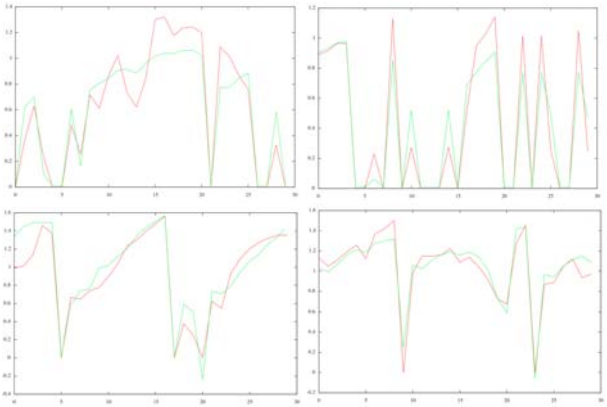

Fig. 3. Modelling (dark line) provided by the best satisfaction model with respect to the STM contents (light line). The x-axis corresponds to the number of sample in the STM and the y-axis to the satisfaction value. Top left graph corresponds to iteration 5, top right to iteration 15, bottom left to iteration 25 and bottom right to iteration 30

With all these elements, the experiment execution was successful requiring, on average, 30 iterations of the robot to learn the task. This implies around 25 minutes of teacher real time interaction, which is an affordable time for this task. Fig. 2 shows a typical trajectory of the robot in the initial iterations (left) where the models are still poor and in iteration 30 (right) where the models have been improved and the robot reaches the ball successfully.

Fig. 3 displays the modeling provided by the best satisfaction model with respect to STM content in four separate iterations. This behavior is the same for the other two models and, as we can see the MDB, works as expected obtaining a gradual learning that improves with time. Furthermore, the case shown in this figure is interesting because it illustrates the potential of using a satisfaction model to guide robot behavior. If in a given instant of time the motivation changes, the only implication it has is that the satisfaction model will fail in its prediction of the satisfaction, and will eventually adapt to the new situation, without affecting the world and internal models. This feature provides real autonomy to the robot as it allows for changing motivations through the satisfaction model and changing environments or even robot structure through world and internal models.

V. Conclusion

We have analyzed the computational implementation of the Multilevel Darwinist Brain in order to demonstrate how an evolutionary algorithm can allow for real time robot operation if it is adequately implemented by separating the reactive and the thinking parts, and allowing them to operate concurrently on different time scales providing the appropriate mechanisms for their coherent operation.

Acknowledgment

This work was supported by Xunta de Galicia (09DPI012166PR) and European Reg. Development Funds.

References

[1] Walker, J, Garrett, A., Wilson, M. “Evolving Controllers for Real Robots: A Survey of the Literature”, Adaptive Behavior vol 11, (2003) 179-203.

[2] Marocco, D., Floreano, D., “Active vision and feature selection in evolutionary behavioural systems”, From Animals to Animats: Proceeings SAB’02, (2002), 247–255.

[3] Watson, J.B. “Behavior-Based Control for Autonomous Robotics“. Proc. of the 3rd Annual Conf. on Evolutionary Programming, World Scientific, Singapore, (1994), 185-190.

[4] Nordin, P., Banzhaf, W., and Brameier, M. “Evolution of a World Model for a Miniature Robot Using Genetic Programming“. Robotics and Auton. Systems, V. 25, (1998), 105-116.

[5] J. Weng, “Developmental Robotics: Theory and Experiments“, *Int. Journal of Humanoid Robotics, vol. 1, no. 2*, 199–236, (2004)

[6] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur and E. Thelen, “Autonomous Mental Development by Robots and Animals“, *Science, vol. 291, no. 5504*, pp. 599 - 600, (2000)

[7] Asada, M., MacDorman, K. F., Ishiguro, H., Juniyoshi, Y., “Cognitive Developmental Robotics as a New Paradigm for the Design of Humanoid Robots“, *Rob. and Auton. Syst., V. 37*, pp. 185-193 (2001)

[8] Richard A. Watson, Sevan G. Ficici, Jordan B. Pollack “Embodied evolution: Distributing an evolutionary algorithm in a population of robots”. Robotics and Autonomous Systems, 39(1):1–18, April 2002.

[9] Floreano, D., Nolfi, S., Mondada, F. “Co-evolution and ontogenetic change in competing robots. Advances in the Evolutionary Synthesis of Intelligent Agents“, MIT Press (2001).

[10] Ostergaard, E. & Lund, H. “Co-evolving complex robot behavior“. In ICES’03, The 5th Int. Conf. on Evolvable Systems: From Biology to Hardware. Springer, (2003).

[11] Duro, R. J., Santos, J., Bellas, F., Lamas, A. “On Line Darwinist Cognitive Mechanism for an Artificial Organism“, Proceedings supplement book SAB2000, (2000), 215-224.

[12] Changeux, J., Courrege, P., Danchin, A. “A Theory of the Epigenesis of Neural Networks by Selective Stabilization of Synapses“, Proc. Nat. Acad. Sci. USA 70, (1973), 2974-2978

[13] Conrad, M. “Evolutionary Learning Circuits. “ Journal of Theoretical Biology, 46, (1974).

[14] Edelman, G. “Neural Darwinism. The Theory of Neuronal Group Selection“. Basic Books (1987), 167-188.

[15] Bellas, F., Becerra, J. A., Duro, R.J. “Construction of a Memory Management System in an On-line Learning Mechanism“, ESANN 2006 Proceedings book (2006), 95-100.

[16] Toby H.J. Collett, Bruce A. MacDonald, and Brian P. Gerkey, "Player 2.0: Toward a Practical Robot Programming Framework". In Proc. ACRA 2005, Sydney, Australia, December 2005.

[17] P. Caamano, R. Tedin, J.A. Becerra, “Java Evolutionary Algorithm Framework”, http://www.gii.udc.es/jeaf

[18] Bellas, F., Duro, R.J. “Multilevel Darwinist Brain in Robots: Initial Implementation”, ICINCO2004 Proc. Book (vol. 2), (2004), 25-32.

[19] F. Bellas, A. Faiña, A. Prieto, and R.J. Duro, “Adaptive Learning Application of the MDB Evolutionary Cognitive Architecture in Physical Agents”, LNAI 4095, 434-445, 2006