Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Learning and Adaptation of Skills in Autonomous Physical Agents

Sandor M Veres * **Aron G Veres** **

* *University of Southampton, UK, s.m.veres@soton.ac.uk*
** *SysBrain Ltd, Birmingham, UK, Email:aron@sysbrain.org*

Abstract: A skills learning methodology is presented for autonomous physical agents. Adaptation of skills and learning is a fundamental part of the simple agent behaviours outlined. A general framework of skills learning is described that uses skill macros to define simple behaviours by agents that communicate, sense and act in the physical world. Programmed playfulness can be easily implemented in this framework that plays an important part in acquiring sophisticated skills. Reusability of results in learning algorithms is supported by ontology based classification of learning in skills. Ontologies provide references to object instances that enable modularization of software and easy interfacing of skills with learning algorithms.

Keywords: Intelligent physical agents, formal methods, learning, adaptive control modelling for control, .

1. INTRODUCTION

Autonomous agents have been used in computer science for some time Wooldridge [2002], Meystel and Albus [2002]. In the field of engineering we want autonomous machines to be (1) capable of solving and executing complex problems in the physical world and we also want them to be (2) reliable. Reliability in essence means that we know how they behave. Reactive agents are good for this as we can prescribe or let them learn the rules by which they behave. Deliberative agents using modelling and complex decision making are however difficult to verify for all physical environments and hence their reliability is more difficult to prove. This apparent dilemma is attempted to be partially solved by a reactive agent architecture in this paper that prescribes a clearly defined behaviour logic for learning, adaptation. Complete verifiability and reliability is not claimed by our method: the main message of the paper is how skills learning can eventually make our architecture more and more reliable.

The well known subsumption architecture for mobile robots was proposed by Brooks based on a set of task-accomplishing behaviours. This approach did away with symbolic reasoning and was choosing the next action that was feasible (some sensory conditions were satisfied='fired') and had the highest priority according to an inhibition relation on the set of all behaviours. In this approach an intelligent behaviour emerged from the dynamical interaction between the environment and the agents as generated by the behaviour rules of the agents. Obvious disadvantages are the lack of memory and the inability to plan the execution of complex tasks under changing conditions.

In Hahnel et al. [1998] the GOLOG logic based robot command language was extended with the GOLEX execution and monitoring system. The Prolog based GOLEX translated high level commands into low level robot execution primitives. The agent architecture of the RHINO mobile robot using GOLOG/GOLEX was based on a task planner, map builder, path planner, collision avoidance and localization modules and the robot had to work out the details of how to execute high level commands. To aim for higher levels of intelligence various kinds of layered architectures have been developed that separate reactive behaviour for quick response and proactive planning for complex tasks at least into two layers P et al. [1995]. Another architecture is TouringMachines A [1992] that uses a reactive and a planning layer with memory for plans to achieve goals that is supported by a modelling layer, all working in parallel to support action selection. The InteRRaP J [1997] is a three layer architecture for cooperation, planning and behaviour selection that relies on a world model, planning knowledge memory and social knowledge about cooperation rules and properties of other agents. A cognitive controller (CoCo) architecture defined in Qureshi et al. [2004] is a three-tiered control architecture for autonomous agents that combines reactive and deliberative components.

Most of the agent architectures proposed in the literature either assume that autonomous systems know how to execute subtasks via their sensor and control algorithm, or they just ignore the problem of learning skills. If learning is considered in the agent literature then it is usually aimed at higher level learning policies using given skills in a discrete world. Learning then addresses policies of behaviour by executing skills at will. For instance they may address the problem of how to systematically search a labyrinth or a room but learning for skills of how to move various types of objects and place them back are usually out of the scope of learning studies and are considered as given. Hence a combination of *learning skills* as well as *learning successful behaviour* by the execution of those skills is very much needed.

In this paper adaptation and learning of agent skills will be examined and formal methodology proposed. The methods presented are applicable to a range of reactive and deliberative agent architectures.

2. SKILL MACROS

Skills are rarely performed by an agent on their own, they are often performed simultaneously or in a sequence: walking and holding an object, turning while watching or tracking. Sometimes or executing a sequence of skills that together form a more complex skill: search for the required object, plan how to approach it, go to it, lift it up, turn around and take it to required position. For us human this is a simple as saying "can you please fetch that tissue for me from the shelf" which is a very high level command "fetch it for me". This simple task means the execution of a sequence of complex skills for a robot. The robot may not abe able to do well some steps, that is why it needs skills learning. In this section composed behaviours are defined by logic formulae built from skill symbols, these formulae will be called skill macros.

The language $\mathbf{L}_{ABL}$ of temporal *agent behaviour logic* is defined over a set of atomic formulae $O_m = \{p, q, ...\}$ called *basic skills* (also called OMs for operational modes) as follows:

$$\phi = p|\neg\phi|\phi \vee \phi|\phi \wedge \phi|\Box\phi|\Diamond\phi|\phi \rightharpoonup \phi|\top|\bot \tag{1}$$

A formula in $\mathbf{L}_{ABL}$ over O_m is called a *skill macros.*

With each skill in O_m there is an activity dynamics associated by the *activity function* defined as

$$A : O_m \mapsto F_b \tag{2}$$

where F_b is a set of feedback loops between the agent's actuators and a part of the agents internal or external environment (internal is for instance for iterative refinement of plans of future actions). There are three important temporal functions defined over the set F_b. The first one is the Boolean temporal *activation function* $a : F_b \rightarrow \{0, 1\}$ and the second one is the *activity value function* $v : F_b \rightarrow [-1, 1]$ and the third one is the timeout function $t : F_b \rightarrow [0, \infty]$. The activation function provides a semantics for the logic of skills (OMs) as the a can be used to evaluate any temporal logic behaviour formula through the activity functions associated with OMs.

Definition 1. A logic formula is called a *simple skill macro* if it only contains the operations $\vee$, $\wedge$ and $\rightharpoonup$.

A simple skill macro defines parallel activities connected by ($\wedge$), sequential activities connected by $\rightharpoonup$ and activity options connected by $\vee$ relations. For instance the $p \vee (q \wedge r) \vee (s \rightharpoonup u \rightharpoonup w)$ can mean that either the skill p is on or the skill q and r are simultaneously active or the skill s is first on then followed by skill u which is followed by w. When w stops operating then either p must start, or q and r or s needs to restart again.

Definition 2. An agent *simple behaviour* $\mathbf{A}$ is defined by the tuple $\mathbf{A} = \{O_m, A, F_b, a, v, t, B\}$ where B is a simple skill macro in terms of the basic skills in O_m.

A simple behaviour defines its semantics in terms of its activity function a. At any moment of time its behaviour formula B can be evaluated in terms of a. Note that satisfaction of B at any time instant does not mean anything about the success or reliability of the agent, it merely says that at any time the agent will activate skills in accordance with satisfying formula B . For any A and B the $A \rightharpoonup B$ is defined true if either A or B holds true and in temporal sense B follows A. Operations of $\vee, \wedge$ are defined as usual as "or" and "and" between skills.

Definition 3. A simple behaviour $\mathbf{A}$ is is called *consistent* if B is true as a temporal logic formula while the behaviour is executed by the agnet, i.e. $\Box B$ evaluates to true using a.

The semantics of behaviour logic formulae is that they take on true 1 or false 0 values during the course of time via the mapping a. $s \rightharpoonup u$ means that s is followed by u in time, hence temporal logic is needed instead of propositional logic (if only $\vee, \wedge, \neg$ were used then propositional logic would suffice).

Note that function a is evaluated over the temporally changing skills F_b and expresses the fact that the agent runs the algorithms of F_b. The qualification of success of operations F_b is expressed by function v, which means poor performance for low positive values, very good performance for values near to 1, instability or totally unacceptable performance for small negative values of v and damaging or dangerous performance for v close to -1.

Definition 4. A consistent simple agent behaviour $\mathbf{A}$ with behaviour formula B is called *safe* if it always evaluates any active skill with $v > 0$ while being active. $\mathbf{A}$ is called *reliable at level* $\epsilon > 0$ if the active skills are always evaluated to $v > 1 - \epsilon$ eventually.

Note that the $v > 0$ condition being defined at any time instant does not mean that $v > 0$ is only dependent on feedback loop data at that time instant. $v > 0$ is typically a function of a control criterion (or self-made goals and external rewards) that is obtained from a sequence of past feedback input-output data.

Safe operation depends on the actual interaction of the agent with its environment. A lot can be achieved for safety by simply altering the a switching function so that if any skill approaches the $v < 0$ region then the agent is switched to another skill that is used to rescue the situation. Whether that will help to achieve overall objectives is another matter and is part of the overall performance evaluation of the agent. Automating a consistent switching mechanism is the topic of the next section.

3. SWITCHING BETWEEN SKILLS

The activity function a of a level-1 autonomous physical agent (APA) is changing over skills as the skills progress. There are the following practically important cases to consider:

(1) An active skill is successfully completed at a required level of reliably.
(2) An active skill is not successful at the required level (e.g. $v < 0.5$) but it still operates safely.
(3) An active skill is timed out.
(4) An active skill is being aborted by another active skill or by higher level decision.

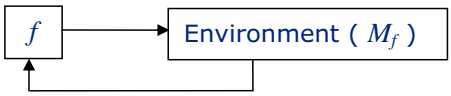

Fig. 1. The feedback interaction of a skill $f \in O_m$ with the environment. An uncertain environmental model M_f can be used to assess or verify and agent's performance under skill f.

The interaction of an agent with its environment under a single skill is normally the topic of control engineering. This control engineering problem can however be widened with the realtime signal choices made for actuators and sensors signals to be used in the feedback loop.

The objective of this paper is to help to make operational mechanisms safer by leaning of autonomous physical agents. The previous section introduced the important concept of the consistency simple behaviours. A further step towards safe operation and good performance is to analyze the conditions of successful operation and bring them together with the above logical framework. To keep the new formalism to a minimum, any $f \in F_b$ will be associated with an initial condition $I(f, M_f)$ that is a 0 or 1 Boolean valued relation function between the agent and its environment.

Definition 5. We say that the *adaptation condition is ϵ-satisfied* by a basic skill f if it holds that whenever $I(f, M_f)$ is satisfied and the agent has f active, the performance $v(f)$ is guaranteed to converge to and stay inside $[1-\epsilon, 1]$ within time period $t(v)$ under uncertain environmental model M_f.

For instance a biped robot may be able to start and walk nicely ($v(f) \to [1-\epsilon, 1]$) if not starting from a lying or fallen-over position but if it already stands reasonably upright ($I(f, M_f)$), even if perhaps a rucksack has been placed on its back. The latter means that under some initial condition the skill of the walking of the robot is adaptive. (Note that here adaptivity is meant in a broad sense, if some modelling is done by the agent it may be part of the basic skill to take that into account for control adaptation.) If the condition $I(f, M_f)$ of walking is not satisfied then the robot may decide to switch to another skill f_1, meaning for instance that the robot is "trying to stand up" first, an action that is a skill itself.

Individual skills of the agent can be tested by formal analysis and practical testing. Hence the above analysis highlights the relevance of enforcing such an a on agent behaviour that starts any $f \in F_b$ under condition $I(f, M_f)$ being satisfied that leads to eventually $v(f) > 1-\epsilon$.

Lemma 6. A simple agent behaviour **A** is reliable at level $\epsilon > 0$ if the following two conditions are satisfied:

(a) All skills $f \in M_f$ satisfy the adaptation condition at level ϵ.
(b) The activity function a is such that whenever an $a(f)$ becomes 1 for an $f \in F_b$ then $I(f, M_f)$ is satisfied.

Proof. Straightforward from the definitions: as all operational conditions are adaptive, and start from correct initial conditions, the performance function v will rise above $1-\epsilon$ for any skill within its time limit.□

The most used main classes of learning skills can be

(1)Parametric feedback/feedforward tuning

(2)Learning for model predictive control

(3)Neural network based FB/FF tuning

(4)Support vector machines

(5)Reinforcement learning.

3.1 Example 1

Assume that a garden robot only has a simple behaviour and it can: either (1) mow the lawn or (2) turn on the watering system or (3) recharge its own batteries or (4) empty the grass from its container to a prescribed dump site (5) report to a human operator for maintenance. Within each of these tasks there are several skills to be executed consecutively:

- $O_1 \to F_{p1}$: *Mowing.*
- $O_2 \to F_{p2}$: *Planning of mowing.*
- $O_3 \to F_{p3}$: *Watering.*
- $O_4 \to F_{p4}$: *Planning of watering.*
- $O_5 \to F_{p5}$: *Empty grass container.*
- $O_6 \to F_{p6}$: *Planning route to charging point.*
- $O_7 \to F_{p7}$: *Recharge.*
- $O_8 \to F_{p8}$: *Decide on and request maintenance.*
- $O_9 \to F_{p9}$: *Write problems report.*
- $O_{10} \to F_{p10}$: *Map building.*
- $O_{11} \to F_{p11}$: *Self modelling of hardware for diagnostics.*
- $O_{12} \to F_{p12}$: *Modelling of past mowing, watering and maintenance work completed.*
- $O_{13} \to F_{p13}$: *Planning for emptying grass container.*
- $O_{13} \to F_{p14}$: *Idle (standby).*

Out of these O_1, O_3, O_5, O_7 are feedback-loop based operational modes that need sensing and control of actions accordingly. O_2, O_4, O_6 are on the other hand skills that need algorithms working on models only and do not need sensing or actuation. skills O_{10}, O_{11} and O_{12} need sensing only and algorithms that build models from sensor data. Decisions by O_8 are based on internal models and may result in sending a message to the human operator after writing report O_9 on the problems that may need maintenance.

A skill macro that the autonomous lawnmower needs to satisfy (for its simple behaviour) can for instance be

$$B_1 = (\ (O_2 \rightharpoonup O_1 \rightharpoonup O_{13} \rightharpoonup O_5) \vee (O_4 \rightharpoonup O_3) \vee \\ \vee (O_6 \rightharpoonup O_7)\ ) \wedge (O_8 \rightharpoonup O_9) \wedge O_{11} \wedge O_{12} \wedge O_{10} \wedge O_{14} \quad (3)$$

The a switch must be such, as decided within each operational mode, that the total formula B must always hold true. This means that the parallel modelling and maintenance monitoring operational activity can carry on while one of the mowing, watering or recharging tasks are executed. Despite the essentially reactive behaviour the lawnmower has the ability of interpretation of the environment (via the basic skill of map building) and planning

while strict discipline of behaviour code is maintained. Based on sensing or assessment of algorithmic results, the evaluation of v is constantly carried out for each operational mode.

Now the reliability at level ϵ of this simple behaviour is achieved if all feedback and open-loop skills are proven to work under uncertain models of the environment and the initial conditions are always achieved when switching to a new basic skill. To achieve the necessary initial conditions the basic skill algorithms need careful action around the switching points. When the physical and control algorithmic work on the lawnmower robot has been completed, then the satisfactory nature of a, v, t can be formally tested. As this may be difficult in practice, adaptation and learning in the skills is therefore vital to reduce development effort and to achieve level-ϵ reliability of the autonomous lawnmower.

4. PLAYFULNESS FOR BEHAVIOUR LEARNING

Why do kittens play? Why do children play? Why do adults play games, and solve crosswords? Playing provides opportunity to practice skills. From the previous sections it is clear that one aspect of learning is to gather data for learning under non-dangerous circumstances. If an agent only executes a simple behaviour in 'live-missions' then it is likely that a huge amount of development effort will be needed to make it to operate safely, all skills need to be performed very well from the very start and also at any switch of operational modes the initial condition must be strictly kept. This is a very demanding job for agent developers.

An alternative is to build a basic structure (using parametric controllers, NNs, self-organizing NNs, reinforcement learning structures, adaptive modelling, etc.) of each skill of a simple behaviour then endow it with the ability to randomly play with the purpose of improving its skills. This section provides a solution by adding a "play skill" to the simple skill macro.

Let B be a simple skill macro for the simple behaviour $\mathbf{A}$ of an agent. An extended formula $B_p = B \vee O_p$ is obtained by adding a skill O_p that takes now a supervisory role. The job of O_p is to monitor performance of each skill under various environmental circumstances and randomly choose from a set of playing activities and execute them by interfering with the normally used activity function a . Playing activities are designed such that they do not interfere with the overall final goals of a mission and they are always obtained as a modified portion of the B. This is achieved by suitable changes in the switching of a.

For instance in the above examples of the lawnmower, some playing activities can be obtained as follows: (a) practice docking for recharging; (b) practice fast and slow mowing on rough ground or high grass; (c) practice finding the boundary of the garden lawn; (d) practice activating the watering system and sensing of how it works. The purpose of the practice is not merely to repeat tasks as that would be useless: its purpose is to fine-tune the skills of the level-1 autonomous system, i.e. to adapt the discrete and continuous control parameters in skills. For that each skill must have a tuneable structure with learning mechanisms. Playing activities for O_p can be preprogrammed by the engineer developing the agent or O_p can also be generated automatically from B. Given the very simple nature of level-1 agents, a straightforward method is that the engineer designs a series of playing activities for the agent. The task of O_p is then to seek out opportunities when these can be played, or depending on learning skills, to activate them. When playing activities are executed learning should take place automatically as all skills should be programmed with learning ability.

4.1 Ontology for skill learning

Modularization and standardization of learning algorithms for skills. In the following ontology description the root class is "learning algorithm" its subclass is "action learning":

```
>learning algorithm
  @default settings: structure
  @v-function : char
  @t-function : char
  @control sequence : signal
  @sensed reactions : signal

  >>action learning
     @performance constraints : text
     @performance criteria : text
     @situation model associations : cell
     @sensory associations: cell

    >>>NN learning
      @NN-type : char
      @structural parameters : cell
      @weights : cell

    >>>modelled action learning
      @predictive model type : char
      @control optimisation method : char

  >>learning dynamical models
      @model-type : char
      @initialisation method : text
      @adaptation method : text
    >>>learning of parametric models
    >>>training of NN
    >>>learning of associative models

  >>learning spacial static scene
      @model type : char
      @method : text
      @model resolution: char
```

4.2 Interfacing of learning algorithms

Learning algorithms are interfaced to the skill via the

(1) measured sensor signals,
(2) v and t functions,
(3) actuator signals used.

These can make the algorithms in principle easy to replace as long as there are built-in mechanisms in the agent architecture to adjust (1) array dimension and (2) initializations to the circumstances by association of past

experience. A shared database of learning algorithms as for the suitability of learning algorithms in various skills is therefore desirable. Such a data base can be made accessible to agents via the Internet. Control engineers who today design adaptive and learning algorithms, can place their work onto this database from where agents could pick them up for and try them to improve their performance (see for instance sysbrain.org)

The learning algorithms are classified and can be searched by shared ontologies of the developer community specified for some families of agents. High level code based algorithms (in MATLAB or MATLAB-like languages such as Octave, SciLab, CC, etc.) can be uploaded for free or can be offered to users at sysbrain.org. *sysbrain.org* offers a central service of storing and redistributing learning algorithms for skills to development engineers of autonomous systems. Each database entry can belong to one of the classes in the above ontology. To provide reusability by others, the following detailed information is to be submitted online:

Table 1. Learning algorithms submission data to sysbrain.org

FAS	Full Application Specification
GVS	Goals and Variables Specification
LMS	Learning Method Specification
ADS	Application Dynamics Specification

4.3 Library of basic skills

In the rest of this section it is analyzed how learning algorithms of skills can be classified, labelled and their input-output classes defined to make them easily reusable by agents. By learning skills we mean algorithms with prescribed input-output object structures as defined by shared ontologies to make them reusable. Classification is needed to cover application areas with suitable learning algorithms and labelling is needed for agents to recognize whether they can be used in some given situations. We intend to define here purpose equivalent learning algorithms.

Classification of learning algorithm is possible at least four ways. The basic categories for learning routine specifications are as follows.

FAS *Full Application Specification.* This is the most specific specification of a routine's applicability. It provides exact make of sensors, actuator set and the make of robot, vehicle or system. There are options for the learning techniques to be used: despite the application fully specified, itself the algorithm is not restricted to one type.

GVS *Goals and Variables Specification.* There are sensor and actuator variables given for specific types of devices without their makes and settings being defined. There is the type of performance criterion function also given that has to be improved by the routine. The performance may be measurable over a longer time period than the response updating period. Again the learning method is not defined. As long as the GVS specification fits the routines are practically replaceable which does not mean that their performance will be the same.

LMS *Learning Method Specification.* The type of method such as some type of artificial neural networks (NN), self-organizing NN (SONN), reinforcement learning (RL), analytic model based adaptive control (AC), associative leaning, including hidden Markov processes (ASL), support vector machines (SVM), empirical tuning of controllers by iterative feedback tuning (ET), etc.

ADS *Application Dynamics Specification* Application dynamics specification is by type and complexity of dynamics as described by state-space or PDE descriptions. Also the supervised or non-supervised nature of the learning algorithm is to be specified here as that is part of the interaction dynamics between the autonomous system and the environment.

Note that these four methods of categorization can actually be reduced to three as FAS is merely a higher resolution version of GVS. GVS and LMS are however "orthogonal" and it is possible to consider their "cross products" as follows. Then a third dimension to locate a learning routine is by type of application dynamics that it is suitable for. Part of ADS is to say whether learning is supervised or unsupervised but actual model types of environmental dynamics also need to be defined under ADS. Table 2 summarizes this classification.

Table 2. Classification of learning algorithms for reuse

GVS/LMS	SONN	RL	AC	ASL	SVM	...
G1V1	ADS	ADS	ADS	ADS	ADS	...
G1V2	ADS	ADS	ADS	ADS	ADS	...
G2V1	ADS	ADS	ADS	ADS	ADS	...
...	...	...	...	...	...	...

Hence a labelling of a learning algorithm category is now possible by constructs like ADS.NN.G1V1 or ADS.AC.G2V1 etc. where ADS, NN, G1V1 are identifiers of the application dynamics description, neural networks structure and temporal variable use and goal descriptions, respectively.

The practical objective is to categorize these learning algorithms so that we try to avoid overlaps within any of the three dimensional boxes used. Even so there can be thousands of combinations of practically useful learning algorithms for autonomous systems that are essentially different and may not be substitutable for each other.

As the learning methods listed above are probably well known by the reader, here we illustrate the classification by goals and temporal variables. For instance a very common problem in control engineering of vehicles and processes is the following: given are

- an array Y of current environmental measured variables,
- an array U of current variables through which the autonomous system can influence variables Y,
- an array Y_r of future reference sequences for Y for time period N,
- a performance function $V(Y_r, Y_{sequence})$,

find an algorithms that can improve (reduce) V as time progresses.

We will call this the G1V1 type of goal-variable specification of learning (for control). It can find solutions among

the most common learning methods as listed above but the methods it can use are heavily affected by the application dynamics that influences learning structures complexity. Note that this type of performance specification is extremely broad: it includes the control problem specification for a jumbo jet, a satellite or two legged robot.

G1V1 essentially says that for a given set of desired output sequence find the control mechanism that strives to keep the performance function V low as the monitored time window for Y progresses. Here we do not include the problem of learning how to set the desired Yr autonomously. Specifying a suitable Y_r can be part of another learning algorithm directed at improving the systems ability of problem solving. This aspect of learning can be well covered by associative learning techniques. The idea of G1V1 is for instance in the form of ADS.G1V1.SVM is based on the use of support vector machines and it would accept various dimensions of arrays for Y, Yr, U and would also accept an input variable for and ADS label. The routine would apply somewhat different algorithms depending on the type and complexity of the application dynamics, depending its ADS variable input. The number of ADS variables labels that the routine would accept depends on with how many types of application dynamics the developer tested the algorithm.

4.4 Initialization of learning algorithms

A major and most important issue is that many learning algorithms cannot efficiently run without some initialization. Insisting on that the user of the learning algorithm should find the initialization would seriously limit the reusability of learning routines. It is therefore the most practical to required that an ADS specification by an author of an ADS, that is taken as an input by the author's algorithm, must also provide an example model of the ADS for simulation purposes that then can be used by users of the routine to initialize the learning algorithm with a method also provided by the developer of the learning algorithm. Although this may be a very demanding job it is fair to say that there is no real prospect of wide reusability for a learning algorithms if others cannot initialize the learning algorithm with reasonable effort by the application developer.

For instance for ADS.G1V1.SVM a possible calling MATLAB m-function interface of a learning routine could be

```
function U = G1V1_SVM(Y,Yr,U,Ads,In_parms)
```

that must be accompanied by two routines when distributed to users:

```
function [Y,state] = G1V1_SVM_model(Ads,state,U)
 function In_parms =
   = G1V1_SVM_init_trainin('G1V1_SVM_model')
```

The implementer of the learning algorithm could then adopt the Ads model to suit the particular application to find initial parameters for the G1V1_SMV that can be implemented realtime in a skill.

A convenient way is to call routines by English sentences. This type of coding is called sEnglish for 'system English' (see www.sysbrain.org for further details).

5. CONCLUSIONS

The main results of this paper is the formalization of skills learning that is applicable to simple reactive as well as complex multilayered deliberative agent architectures. All agents need skills learning as higher levels of abstraction and deliberation does not solve the problem of acquiring and using skills of interaction with the physical environment and other agents. Physical agents need to operate in the real world "skillfully" that is difficult to pre-program. The problem is not only the number of engineering hours needed (to do the development work) but also that the agents will not be able to perform well in changing environments if basic skills are non-adaptive. Need for learning has been broadly recognized before but in this paper we formally separate skills learning from policy learning in deliberative architectures.

On the other hand the ideas of this paper also give recipes to simplifying autonomous agent architectures and not to apply deliberative architectures where they are not really essential. It is illustrated that the capability of higher levels of abstraction are not required to be able to exhibit highly adaptive, planning-based-behaviour that is normally associated with higher levels of intelligence. Instead of higher levels of deliberation, simply some planning and modelling skills can be added to simple behaviours. Finally the paper presented an initial classification and interfacing system for learning algorithms that can ease the programming of agents by community wide sharing of code and structures.

REFERENCES

Fergusion I A. Towards an architecure for dynamic, rational, mobile agents. *Proc. 3rd European Wroshop in Modelling Autonmous Agents*, MAARMAW-91:249–262, 1992.

D. Hahnel, W. Burgard, and G. Lakemeyer. GOLEX - bridging the gap between logic (GOLOG) and a real robot. *Proc. of the 22nd German Conf. on AI (KI-98)*, page 12, 1998.

Muller J. A cooperation model for autonmous agents. *In: Intelligent Agents, III, Edts: Muller, Wooldridge, Jennings*, LNAI 1193:245–260, 1997.

A. M. Meystel and J. S. Albus. *Intelligent Systems: Architecure, design and Control*. Wiley Series on Intelligent Systems. John Wiley and Sons, Inc., New York, 2002. ISBN ISBN 0-471-19374-7.

Muller J P, Pishel M, and Thiel M. Modelling reactive behaviour in vertically layered agent acrhitectures. *LNAI, In Intelligent Agents: Theories, Acrhitectures and Languages*, LNAI 890:261–276, 1995.

F. Z. Qureshi, D. Terzopoulos, and R. Gillett. The cognitive controller: A hybrid, deliberative/reactive control architecture for autonomous robots. In B. Orchard, C. Yang, and M. Ali, editors, *Innovations in Applied Artificial Intelligence. 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert System (IEA/AIE 2004)*, volume 3029 of *Lecture notes in Artificial Intelligence*, pages 1102–1111, Ottawa, Canada, May 2004. Springer-Verlag.

M Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, 2002. ISBN ISBN 0-471-49691-X.