2015

The Self-Adaptive Context Learning Pattern: Overview and Proposal

Jeremy Boes
Universite de Toulouse

Julien Nigon
Universite de Toulouse

Nicolas R. Verstaevel
University of Wollongong, nicolasv@uow.edu.au

Marie-Pierre Gleizes
Universite Paul Sabatier

Frederic Migeon
Universite Paul Sabatier

The Self-Adaptive Context Learning Pattern: Overview and Proposal

Abstract

Over the years, our research group has designed and developed many self-adaptive multi-agent systems to tackle real-world complex problems, such as robot control and heat engine optimization. A recurrent key feature of these systems is the ability to learn how to handle the context they are plunged in, in other words to map the current state of their perceptions to actions and effects. This paper presents the pattern enabling the dynamic and interactive learning of the mapping between context and actions by our multi-agent systems.

Keywords

context, self-adaptive, learning, proposal, pattern:, overview

Publication Details

Boes, J., Nigon, J., Verstaevel, N., Gleizes, M. & Migeon, F. (2015). The Self-Adaptive Context Learning Pattern: Overview and Proposal. Lecture Notes in Artificial Intelligence, 9405 91-104.

The Self-Adaptive Context Learning Pattern: Overview and Proposal

Jérémy Boes, Julien Nigon, Nicolas Verstaevel,
Marie-Pierre Gleizes, and Frédéric Migeon

SMAC Team, IRIT, 118 rte de Narbonne, Toulouse, France
{boes,jnigon,verstaev,gleizes,migeon}@irit.fr
http://irit.fr/SMAC/

Abstract. Over the years, our research group has designed and developed many self-adaptive multi-agent systems to tackle real-world complex problems, such as robot control and heat engine optimization. A recurrent key feature of these systems is the ability to learn how to handle the context they are plunged in, in other words to map the current state of their perceptions to actions and effects. This paper presents the pattern enabling the dynamic and interactive learning of the mapping between context and actions by our multi-agent systems.

Key words: Self-Organisation, Context, Learning, Adaptation, Multi-Agent System, Cooperation, Machine Learning

1 Introduction

Real-world problems offer challenging properties, such as non-linear dynamics, distributed information, noisy data, and unpredictable behaviours. These properties are often quoted as key features of complexity [10]. Dealing with the complexity of the real world is an active research field. Complexity implies that models are insufficient. Systems designed for real-world have to be able to learn and self-adapt, they cannot rely on predefined models.

Thanks to their natural distribution and flexibility, self-organizing Multi-Agent Systems (MAS) are one of the most promising approaches [14]. A good way to design MASs for this type of problems is to decompose the problem following its organisation [13]. For many applications, such as bio-process control [19], engine optimization [3], learning in robotics [17], and user satisfaction in ambient systems [9], this decomposition leads to a crucial sub-problem: mapping the current state of the context with actions and their effects. Context is a word used in many domains and each domain comes with its own definition. There is several proposals for a definition of what context is [2]. In the field of problem solving, Brezillon [5] defines the context as "what constrains a step of a problem solving without intervening in it explicitly". This paper stems from the field of multi-agent systems. Agents are autonomous entities with a local perception of their environment. In this paper the *context* refers in this paper to all information

which is external to the activity of an agent and affects its activity. It describes the environment as the agent sees it [9].

In general a system is coupled to its environment by a cycle of observation-action. For instance for a self-adaptive system plunged in a dynamic environment, the observations become the inputs of the system and the actions its outputs. The goal of the system is to find an adequate action for the current state of inputs coming from the environment. This current state of inputs is the context. This is a mapping problem, where the current context must be mapped to an action.

When we solved the context mapping sub-problem in different applications, we have highlighted a recurrent pattern. This pattern is an abstraction of the core of several adaptive systems applied to various real-world problems and sharing the context mapping sub-problem. This recurrent pattern is named *Self-Adaptive Context-Learning Pattern* (SACL) in this paper. It is composed of an Adaptation Mechanism coupled with an Exploitation Mechanism. The Adaptation Mechanism feeds the Exploitation Mechanism with information about possible actions in the current context, while the Exploitation Mechanism is in charge of finding the most adequate action. Both mechanisms can be implemented using different approaches. In this paper, we propose an implementation of the Adaptation Mechanism based on adaptive multi-agent systems. This implementation is composed of a set of adaptive agents. Each agent represents the *effects* of a given *action* in a given *context*, the three of them being learned at runtime, in interaction with both the Exploitation Mechanism and the environment.

First, section 2 details motivations and the structure of the Self-Adaptive Context-Learning (SACL) pattern. Section 3 proposes an implementation of the Adaptation Mechanism based on the AMAS approach. Section 4 shows how the Self-Adaptive Context-Learning pattern and AMAS4CL are used in two of the aforementioned applications. Finally, section 5 explores an interesting relation with schema learning, before section 6 concludes with some perspectives.

2 Self-Adaptive Context-Learning Pattern

The *Self-Adaptive Context-Learning Pattern* is composed of an *Exploitation Mechanism* and an *Adaptation Mechanism* both in interaction with the environment. The SACL environment is every entity which is outside SACL but affects its behaviour. So a distinction has to be done between SACL environment and the local environment of an agent (see section 3). These entities can be data coming from sensors, messages from other software, effectors or systems to control. The data perceived by the pattern are called percepts. The Exploitation Mechanism is the acting entity. It has to decide and apply the action which has to be performed by a controlled system in the current context. Its decision is based on its own knowledge, including constraints from the application domain, and additional information provided by the Adaptation Mechanism. The function of the Adaptation Mechanism is to build, maintain, and provide reliable and useful knowledge about the current context and possible actions. To gather this

knowledge the Adaptation Mechanism has to correlate the activity of the Exploitation Mechanism to the observation of the environment. In order to acquire and to keep this knowledge up-to-date with the dynamics of the Exploitation Mechanism and the environment, the Adaptation Mechanism has to learn and self-adapt.

The Adaptation Mechanism and the Exploitation Mechanism are coupled entities and form a control system (figure 1). Adaptation Mechanism acquires (arrow 1) and provides information (arrow 3) about actions (arrow 2) made by the Exploitation Mechanism relatively to the context, while the Exploitation Mechanism relies on the information given by the Adaptation Mechanism to make these actions. Moreover, the Exploitation Mechanism sends feedback (arrow 4) to the Adaptation Mechanism to indicate whether or not the previous received information was useful. This usefulness is computed from its perceptions (arrow 5). The feedback includes two information: the last action done by the Exploitation Mechanism, and a binary qualitative evaluation of the usefulness of the previous received information.

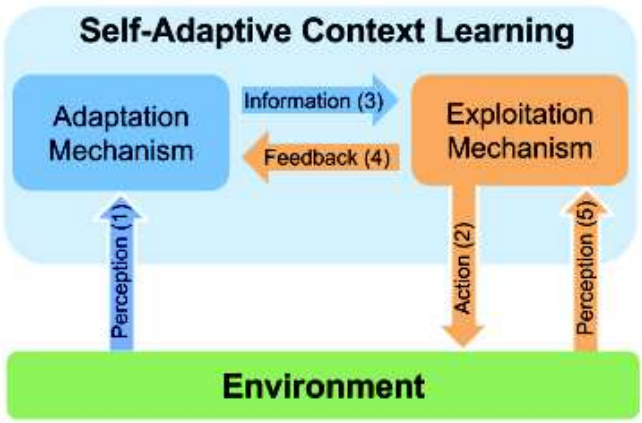

Fig. 1. The Self-Adaptive Context Learning Pattern

3 AMAS4CL

The law of requisite variety states that the complexity of a control system must be greater or equal to the complexity of the controlled system[1]. This means that to control a complex system, one has to build an even more complex system. Hence, to deal with real world complexity, the SACL pattern has to be implemented with approaches able to generate and handle complex behaviours. Our proposal for an implementation of the Adaptation Mechanism is based on such

an approach, called Adaptive Multi-Agent Systems (AMAS). It relies on the cooperative self-organisation of agents, and is called AMAS for Context Learning (AMAS4CL).

3.1 Adaptive Multi-Agent Systems

The Adaptive Multi-Agent Systems (AMAS) approach aims at solving problems in dynamic non-linear environments. This requires learning and self-adaptation. The approach uses the self-organisation of agents as a mean to learn and self-adapt. This self-organisation is a process driven by cooperation principles [16]. The approach relies upon the classification of interactions between a system and its environment from Kalenka[11]. An interaction can be:

- Cooperative, i.e. helpful for both the agent and the environment ;
- Neutral, i.e. without consequences for the environment or the agent ;
- Antinomic, i.e. detrimental to the agent and/or the environment.

In the AMAS approach, an agent is said to be in a *cooperative state* if all its interactions with its environment are cooperative, otherwise it is said to be in a *Non-Cooperative Situation (NCS)*. An agent in a cooperative state executes its *nominal behaviour*, the behaviour that makes it fulfill its task. When the agent detects a NCS, a specific behaviour is triggered instead of the nominal behaviour. An agent is designed to be aware of several types of NCSs:

- Incomprehension if the agent is unable to extract any information from its perceived signals ;
- Ambiguity if the agent can interpret its perceived signals in several ways;
- Incompetence if the decision process of the agent cannot be applied to its current internal representations and perceptions;
- Improductivity if the decision process of the agent leads to no action;
- Conflict if the agent estimates that its action is antinomic with its environment (including other agents);
- Uselessness if the agent estimates that its action has no effect on its environment (including other agents).

It has been shown that if all the agents of a MAS are in a cooperative state, then the MAS is functionally adequate, i.e. it is properly fulfilling its task [6]. Hence, the challenge is to design agents able to eliminate NCSs, in order to reach a cooperative state, and to always try to stay in this state despite changes in their environment. To solve the NCSs the agent changes the way it interacts with its own environment. These changes can be of three types (or any combination of them):

- Tuning: the agent adjusts internal parameters ;
- Reorganisation: the agent changes the way it interacts with its neighborhood, i.e it stops interacting with a given neighbor, or it starts interacting with a new neighbor, or it updates the importance given to its existing neighbors.
- Openness: the agent creates one or several other agents, or deletes itself.

These behaviours are applied locally by the agents, but they have an impact on activity of the whole system. Hence, they are self-organisation mechanisms.

Applying this approach to the problem of context learning leads to a specific type of agents, called Context Agents. They are created at runtime, and self-adapt on-the fly. These agents are the core of our proposition for an implementation of the Adaptation Mechanism of the SACL pattern, described in the next section.

3.2 AMAS4CL Agents

AMAS4CL has two types of agents: Context Agents and a Head Agent.

The Head Agent is a unique agent created at the beginning allowing interactions between the Exploitation Mechanism and the inner agents in AMAS4CL. It is basically a doorman allowing messages to come in and come out. Its behaviour, described in the section 3.3, enables the interoperability of the Adaptive Mechanism with any kind of Exploitation Mechanism. Although this agent performs no control over the system, its observatory position enables it to detect and repair some NCSs (see section 3.4).

AMAS4CL starts with no seed, no predefined Context Agent. They are all created at runtime (see section 3.4). A Context Agent has a tripartite structure: $< context, action, appreciation >$. The first part, the *context*, is a set of intervals called *validity ranges*. There is one validity range for each percept of the Context Agent.

Definition 1. *A validity range r_p, associated to the percept p, is valid if $v_p \in r_p$; where v_p is the current value of p.*

By definition, a Context Agent is valid if all its validity ranges are *valid*. When a Context Agent is valid, it sends an action proposition to the Head Agent. The *action* is a modification of the environment. For instance, it can be the incrementation or decrementation of a variable. It can also be a high level action, such as "go forward" for a robot. It is domain dependent.

The *appreciation* is an estimation of the effect of an action at the creation of the Context Agent. It may be for example the expected effects of the action on the environment, the estimated gain following a given measure, a relevance score, or a combination of these, etc. A Context Agent has to adjust this *appreciation*. For this, it possesses an evaluation function determining after a proposal if the *appreciation* is $wrong$ or $inexact$. $Wrong$ means that the difference between the estimated *appreciation* and the observed effect is unacceptable (from the designer point of view) whereas $inexact$ means that the error is tolerable. Because the *appreciation* depends on the application domain, the estimation function is given by the designer.

A Context Agent can be seen as a tile delimited by its validity ranges. When a Context Agent sends a proposition, it basically says: "here is what I think of this action in the current context". The Exploitation Mechanism receives several propositions and uses them to decide what is the best action to be applied. This is detailed in section 3.3.

3.3 Nominal behaviour

The *nominal behaviour* is the behaviour enabling the agent to perform its tasks in a cooperative situation [4]. According to the AMAS approach, we can model the nominal behaviour of an agent through a Perception-Decision-Action cycle.

Nominal behaviour of Context Agents:

- **Perception**:
  - 1: Receives from the environment a set of variable values (percepts).
  - 2: Receives the feedback from the Exploitation Mechanism. This feedback is composed of a binary qualitative information about the information provided by AMAS4CL at the previous decision cycle ($ENOUGH/NOT\_ENOUGH$) and the last action performed by the Exploitation Mechanism.
- **Decision and Action:**
  - 1: The Context Agent checks whether it is *valid* or not by comparing the current set of variable values to its *context*.
  - 2: If the Context Agent is valid, it sends an *action* proposition associated with the *appreciation* of this *action* to the Head Agent.

Nominal behaviour of Head Agent:

- **Perception:**
  - 1: Receives the feedback from the Exploitation Mechanism.
  - 2: Receives action propositions from Context Agents.
- **Decision and Action:**
  - 1: Gathers all the action propositions and sends them to the Exploitation Mechanism.
  - 2: Forward the feedback from the Exploitation Mechanism only to the Context Agents that had proposed the action contained in the feedback.

There are several cases where these behaviours fail. These cases are NCS. To solve them, the agents have to self-adapt, i.e. to modify their behaviour. This is detailed in the next section.

3.4 Non-Cooperative Situations

NCS are described in two parts: the detection of the NCS, and its resolution. These mechanisms of detection-resolution are completing the behaviour of Context Agents.

NCS 1: Conflict of a Context Agent (wrong appreciation)

- **Detection**: Thanks to the feedback from the Exploitation Mechanism, a Context Agent knows when its action is being applied. When its action is being applied, the Context Agent observes the effects of the action, to check if its *appreciation* is correct. If the agent evaluates that it has given wrong information to the the Exploitation Mechanism, it is a conflict NCS. The interaction between the Context Agent and the Exploitation Mechanism is flawed and prevents one of them to fulfill its task.

– **Resolution**: The Context Agent estimates that it should not have been valid. To solve this NCS, the Context Agent reduces its validity ranges to avoid to make a proposal in a context where it is unable to give a good *appreciation*.

NCS 2: Conflict of a Context Agent (inexact appreciation)

– **Detection**: This NCS is similar to the conflict NCS of wrong *appreciation*. If a Context Agent considers that its *appreciation* is inexact, it is also conflict NCS. The inexactitude of the appreciation prevents an optimal activity of the Exploitation Mechanism.
– **Resolution**: This NCS is less harmful, the Context Agent estimates it was still right to make a proposal, it only needs to give a more accurate *appreciation*. Thus, the agent does not change its validity ranges, but adjusts its *appreciation*, using information from the feedback of the Exploitation Mechanism and from the observation of the environment.

NCS 3: Uselessness of a Context Agent

– **Detection**: Sometimes, after successive adjustments of their ranges, Context Agents may have one or more range greatly reduced. If this range becomes smaller than a user-defined critical size (for instance: zero, or very close to zero), the Context Agent considers itself as useless, since it has no chance of being valid again.
– **Resolution**: The agent self-destroys.

NCS 4: Improductivity of the Head Agent

– **Detection**: If the Head Agent receives a feedback containing an action that was not proposed at the previous step, the decision leads the agent to forward it to no one. It is an NCS of improductivity. It occurs when no proposal was received by the Head Agent (for instance, at the start, when there is no Context Agent in the system), or when none of the proposed actions were applied by the Exploitation Mechanism. This NCS means the Exploitation Mechanism had to explore by executing a new *action*. This exploration is domain dependant.
– **Resolution**: If the Head Agent had received proposals with the action contained in the feedback earlier in its lifetime, it requests the Context Agents that had sent it to expand their validity ranges toward the current context. If nobody is able to do that (Context Agents may reject this request if the adjustment is too big), the Head Agent creates a new Context Agent with the new action received in the feedback, and initializes it with the current context and a first appreciation that depends on the domain.

4 Applications

This section gives examples on how the context-learning pattern is used to solve real-world problems. We focus on two different applications: ALEX, a multi-

agent system that learns from demonstrations to control robotic devices and ESCHER, a multi-agent system for multi-criteria optimization.

4.1 ALEX

Service robotic deals with the design of robotic devices whose objectives are to provide adequate services to their users. User needs are multiple, dynamic and sometimes contradictory. Providing a natural way to automatically adapt the behaviour of robotic devices to user needs is a challenging task. The complexity comes with the lack of way to evaluate user satisfaction without evaluating a particular objective. A good way to handle this challenge is to use Learning from Demonstrations, a paradigm to dynamically learn new behaviours from demonstrations performed by a human tutor. With this approach, each action performed by a user on a device is seen as a feedback. Through the natural process of demonstration, the user not only shows that the current device behaviour is not satisfying him, but also provides the adequate action to perform. Adaptive Learner by EXperiments (ALEX) [17] is a multi-agent system designed to face this challenge.

System overview Our approach considers each robotic device as an autonomous agent in interaction with other components, humans and the environment through sensors [18]. An instance of ALEX is associated with each robotic device composing the system (basically one by effector). For example in the case of a two wheeled rover, each wheel has its own controlling system. ALEX receives a set of signals coming from sensors and the activity of the human tutor performing the demonstration. The tutor can perform at runtime a demonstration by providing to ALEX the adequate action to perform. The problem is then to map the current state of sensors to the activity of the user in order to be proactive the next time a similar situation occurs and the tutor provided no action. ALEX is built on the SACL pattern and AMAS4CL (Figure 2).

AMAS4CL must provide to the Exploitation Mechanism the adequate action to satisfy the tutor. However, tutor satisfaction is not directly observable and it is difficult to correlate the effect of an action on tutor satisfaction without evaluating an a priori known objective. The *appreciation* part of Context Agents is a dynamically built confidence value. This confidence value is increased each time a Context Agent is activated and proposes the same action as the tutor's, and is decreased otherwise. Each Context Agent dynamically manages its confidence value thanks to the feedback from the Exploitation Mechanism. Then, at each decision cycle, the action proposed by the activated Context Agent with the utmost confidence is sent to the Exploitation Mechanism. More information on the implementation of this confidence value can be found on previous works [9].

Results ALEX has been tested both on simulation and on a real Arduino based robot [18]. The experiment is a collecting task involving a two wheeled rover inside an arena. A user performs a five minutes demonstration of a collecting task allowing the rover to learn the desired behaviour. The number of collected

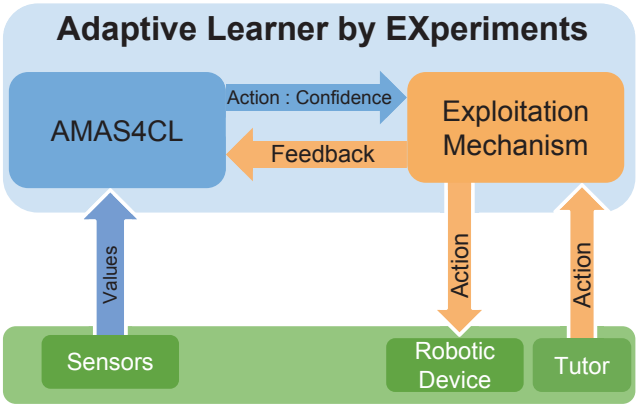

Fig. 2. ALEX is based on the Self-Adaptive Context-Learning Pattern using AMAS4CL

artifact is computed during the demonstration and this score is compared to the score performed by the rover. Experiments have shown that ALEX increases system capacity to perform this task: the rover often collects more artifacts than the user (see figure 3) [18].

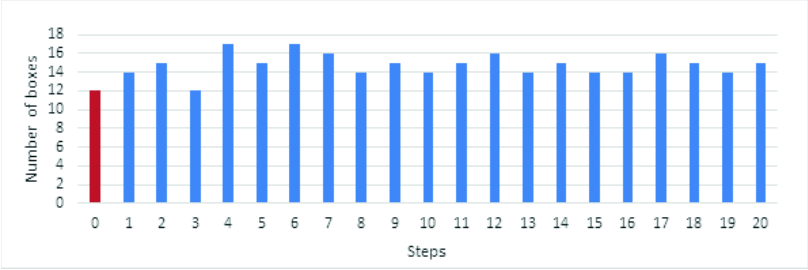

Fig. 3. Number of collected boxes each 5 minutes. The step 0 corresponds to the reference score.

4.2 ESCHER

ESCHER, for Emergent Self-organized Control for Heat Engine calibRation, is a multi-agent system able to learn in real-time how to control another system. It was used to find the best control for heat engines in order to calibrate them [3]. The main problem is to find which action will increase the satisfaction of a set of user-defined criteria, regarding the current state of the controlled system. In

other words, the problem includes the mapping of the controlled system state-space with actions and their effects, hence the use of the SACL pattern.

System overview The environment of ESCHER is composed of the Controlled System along with the user's criteria for the control, such as setpoints, thresholds, and optimization needs. ESCHER performs closed-loop control, while simultaneously learning from its interactions with the environment. This learning is handled by several distributed SACL modules, implemented with AMAS4CL.

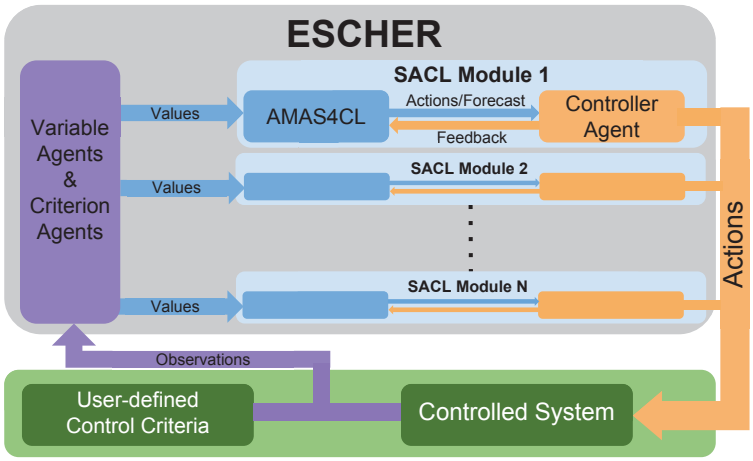

Fig. 4. ESCHER is based on the Self-Adaptive Context-Learning Pattern using AMAS4CL

Figure 4 shows an overview of ESCHER. There is one SACL module per input of the Controlled System. The Exploitation Mechanism of each module is an agent called Controller Agent, in charge of applying the most adequate action on its associated input of the Controlled System. Along with SACL modules, ESCHER includes an internal representation of its environment in the form of Variable Agents and Criteria Agents. There is one Variable Agent per input and output of the Controlled System. A Criterion Agent represents an optimization need, a threshold to meet, or a setpoint to reach. A Criterion Agent expresses its satisfaction in the form of a *critical level*: the lower the better, zero means the criterion represented by the agent is fully satisfied.

ESCHER works with continuous systems and has an explicit representation of criteria. This has led to the following characteristics for the Context Agents of AMAS4CL instances:

- The *context* part of a Context Agent is a set of adaptive range trackers, a specific tool to learn value ranges with simple binary feedback;
- The *action* part is a modification of an input ($+\delta$ or $-\delta$);

– The *appreciation* part is a set of forecasts about the variation of the critical levels.

Controller Agents receive action proposals with their expected effects on the criteria satisfaction, and select the most adequate action to apply, meanwhile sending appropriate feedback to the Context Agents. When no proposal is received, a Controller Agent applies a new action, chosen randomly. When no received proposal includes an adequate action, it does the same, but excluding actions known for their bad effects. There are several additional NCSs to the four of AMAS4CL. In particular, Context Agents are able to adjust their proposed *action*.

Results ESCHER has been experimented on both artificial and real cases. Here are only presented the results of an experiment on a real 125cc monocylinder fuel engine. More detailed results can be found in [3] and will be published in future papers. The goal of this experiment is to perform a classical optimization of the engine: maximizing the torque, minimizing the fuel consumption, and meeting pollution thresholds. ESCHER controls three parameters: the injected mass of fuel, the ignition advance, and the start of injection. Thus, there are three instances of SACL/AMAS4CL.

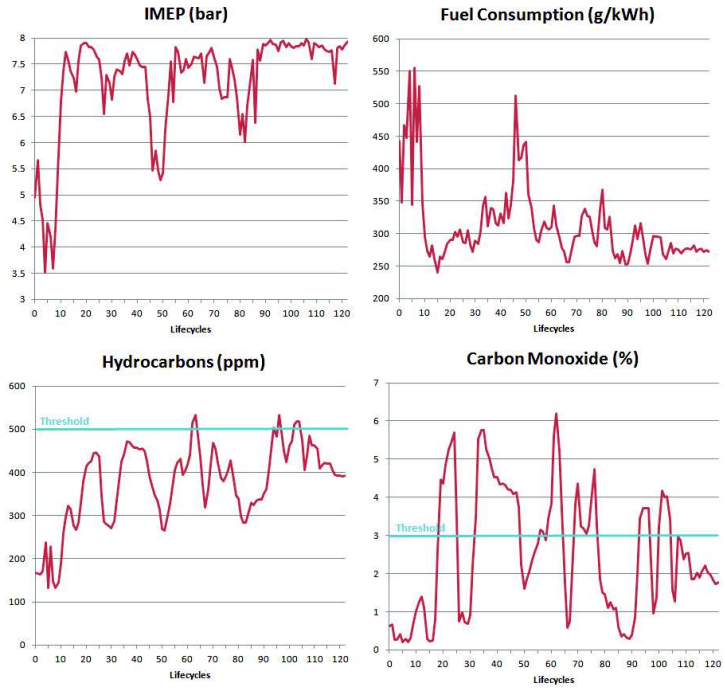

Fig. 5. An engine optimization performed by ESCHER

Figure 5 shows the evolution of the outputs of the engine over time: the Indicated Mean Effective Pressure (IMEP, an image of the torque), the fuel consumption, and the two considered pollutants: hydrocarbons and carbon monoxide. At the beginning, ESCHER has no Context Agent. Controller Agents make mistakes: during the first dozen of lifecycles, the IMEP drops, and the fuel consumption rises. After a while, ESCHER has acquired enough knowledge and is able to maximize the IMEP and to minimize the fuel consumption. During the optimization process, pollutants sometimes rise over their threshold. But, in the end, ESCHER finds a configuration where IMEP and fuel consumption are optimized, and pollutants under their respective threshold. The obtained result is equivalent to the calibration, which currently is only manually performed by experts with full knowledge of this particular engine.

5 Related Work

AMAS4CL learns without any global evaluation function about it activity, but only local evaluation functions at the level of agent. Thus, every approach using objective or global evaluation functions (like neural networks or genetic algorithms) are considered as not related.

SACL pattern was designed from the beginning as a self-organized, bottom-up approach to solve real-world problems. The Context Agents structure, associating context/action/prediction, does not try to mimic nature, human brain or way of thinking. At opposite, other work gets inspiration from human behaviour to build intelligent artificial systems and propose some patterns which have interesting similarities with AMAS4CL.

Drescher proposed the schema mechanism[8] as a way to reproduce the cognitive development of human during infancy as it was described by Piaget. The main part of this approach is the notion of schemas. Each schema asserts that a specific result will be obtained in a specific context by the application of a specific action. This is similar to our Context Agent structure (figure 6).

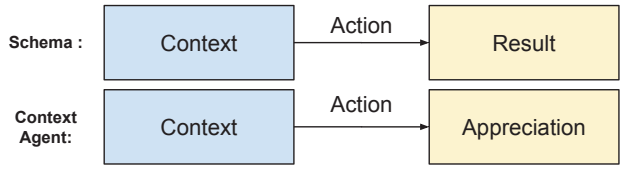

Fig. 6. Schema and Context Agent structures

Schema mechanism is able to learn about its environment by three means: induction (creating new schemas to express regularity in the environment), abstraction (creating new actions as a combination of primitive actions) and

conceptual invention(creating new elements to represent unknown state of the world). As AMAS4CL, schema mechanism does not use any objective function and can learn with very few specifications about the environment. Some of the limitations of the schema mechanism were improved by later works, in particular (but not restricted to) the intense need for resource by Chaput [7] and Perotto [15], or the difficulty to handle a continuous environment [12].

Even if AMAS4CL and constructivist learning approach share some properties, their learning strategies are strongly different. It is interesting that two approaches, one starting from a representation of the child's cognitive development, and the based on cooperative interactions between agents, stemming from system theory, converge towards a similar structure for information-carrying entities (schema and Context Agent).

6 Conclusion

This paper presents the Self Adaptive Context Learning Pattern designed to handle real-world non-linear problems. This pattern relies on two parts: the Exploitation Mechanism which acts on the environment, and the Adaptation Mechanism which provides information to the Exploitation Mechanism. The Adaptation Mechanism acquires and adjusts these information by the observation of the environment.

AMAS4CL is a multi-agent system which enables this adaptation, through the use of cooperative agents. By minimizing the need of assumptions on the studied system, the approach is generic and was used in a wide variety of complex real-world problems, like complex systems control [3], robotics [18], or ambient systems [9].

The ability to handle real-world complexity could be achieved by a better understanding of the context, and by efficient mapping between current context and relevant information for the Exploitation Mechanism. But, in dynamic environments, a static mapping is somewhat limited. The SACL pattern and its implementation brings an adaptive approach to handle this problem.

Several works are currently underway using AMAS4CL, in the fields of networks control, generation of complex system models and human user behaviour understanding. These works are opportunities to improve AMAS4CL. Future work includes formalisation of AMAS4CL and comparison with other approaches, such as schema learning.

References

1. W. Ross Ashby, *An introduction to cybernetics*, Chapman & Hall, London, UK, 1956.
2. Mary Bazire and Patrick Brézillon, *Understanding context before using it*, Modeling and using context, Springer, 2005, pp. 29–40.

3. Jérémy Boes, Frédéric Migeon, Pierre Glize, and Erwan Salvy, *Model-free Optimization of an Engine Control Unit thanks to Self-Adaptive Multi-Agent Systems*, ERTS2, Toulouse, SIA/3AF/SEE, 2014, pp. 350–359.
4. Noelie Bonjean, Wafa Mefteh, Marie-Pierre Gleizes, Christine Maurel, and Frédéric Migeon, *Adelfe 2.0*, Handbook on Agent-Oriented Design Processes, Springer, 2014, pp. 19–63.
5. Patrick Brézillon, *Context in problem solving: a survey*, The Knowledge Engineering Review (1999), no. 01, 47–80.
6. Davy Capera, Jean-Pierre Georgé, M-P Gleizes, and Pierre Glize, *The amas theory for complex problem solving based on self-organizing cooperative agents*, Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on, 2003, pp. 383–388.
7. Harold H Chaput, Benjamin Kuipers, and Risto Miikkulainen, *Constructivist learning: A neural implementation of the schema mechanism*, Proceedings of the Workshop on Self-Organizing Maps (WSOM03), 2003.
8. Gary L Drescher, *Made-up minds: a constructivist approach to artificial intelligence*, MIT press, 1991.
9. Valérian Guivarch, Valérie Camps, and André Péninou, *AMADEUS: an adaptive multi-agent system to learn a user's recurring actions in ambient systems*, Advances in Distributed Computing and Artificial Intelligence Journal, Special Issue 3 (2012).
10. Francis Heylighen, J Bates, and MN Maack, *Encyclopedia of library and information sciences*, 2008.
11. Susanne Kalenka, *Modelling social interaction attitudes in multi-agent systems*, Ph.D. thesis, Citeseer, 2001.
12. Sébastien Mazac, Frédéric Armetta, and Salima Hassas, *On bootstrapping sensorimotor patterns for a constructivist learning system in continuous environments*, Alife 14: Fourteenth International Conference on the Synthesis and Simulation of Living Systems, 2014.
13. Victor Noel and Franco Zambonelli, *Engineering emergence in multi-agent systems: Following the problem organisation*, High Performance Computing & Simulation (HPCS), 2014 International Conference on, IEEE, 2014, pp. 444–451.
14. Liviu Panait and Sean Luke, *Cooperative multi-agent learning: The state of the art*, Autonomous Agents and Multi-Agent Systems 11 (2005), no. 3, 387–434.
15. Filipo Studzinski Perotto, Rosa Vicari, and Luís Otávio Alvares, *An autonomous intelligent agent architecture based on constructivist ai*, Artificial Intelligence Applications and Innovations, Springer, 2004, pp. 103–115.
16. Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, *Self-organising systems*, Springer, 2011.
17. Nicolas Verstaevel, Christine Régis, Marie-Pierre Gleizes, and Fabrice Robert, *Principles and experimentations of self-organizing embedded agents allowing learning from demonstration in ambient robotic*, Procedia Computer Science 52 (2015), no. 0, 194 – 201, The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015).
18. Nicolas Verstaevel, Christine Régis, Valérian Guivarch, Marie-Pierre Gleizes, and Fabrice Robert, *Extreme Sensitive Robotic A Context-Aware Ubiquitous Learning*, ICAART, vol. 1, INSTICC, 2015, pp. 242–248.
19. Sylvain Videau, Carole Bernon, Pierre Glize, and Jean-Louis Uribelarrea, *Controlling bioprocesses using cooperative self-organizing agents*, Advances on Practical Applications of Agents and Multiagent Systems, Springer, 2011, pp. 141–150.