Self-Organizing Multi-Agent Systems for the Control of Complex Systems

Jérémy Boes, Frédéric Migeon

Self-organizing multi-agent systems for the control of complex systems

Jérémy Boes *, Frédéric Migeon

IRIT, University of Toulouse 118, route de Narbonne, F-31062 Toulouse Cedex 9, France

A B S T R A C T

Because of the law of requisite variety, designing a controller for complex systems implies designing a complex system. In software engineering, usual top-down approaches become inadequate to design such systems. The Adaptive Multi-Agent Systems (AMAS) approach relies on the cooperative self-organization of autonomous micro-level agents to tackle macro-level complexity. This bottom-up approach provides adaptive, scalable, and robust systems. This paper presents a complex system controller that has been designed following this approach, and shows results obtained with the automatic tuning of a real internal combustion engine.

Keywords:
Multi-Agent systems
Control
Self-organization
Complex systems
Internal combustion engines

1. Introduction

Controlling a system means being able to perform the adequate modifications on its inputs in order to set the outputs on a desired state. Over the course of History, humans made tremendous efforts to control systems that are more and more complex: non-linear, dynamic, noisy, with a large number of inputs and outputs, and so on. Yet, the law of requisite variety (Ashby, 1956) implies that the complexity of a controller has to be greater than or equal to the complexity of the target system. Thus, the design of a controller involves the design of a complex system. This is a challenge for engineering.

Complexity is often tackled a posteriori, to study existing systems. On the contrary, methods enabling the design of complex systems that meet strict requirements are quite rare. The main feature of a complex system is that its behavior can not be easily predicted (Heylighen, 2008). Usual design methods, for instance in software engineering, seek to a priori eliminate any unexpected event. The design process must ensure that everything will be smooth at runtime. But, as any other complex system, complex programs sometimes have unexpected, unpredictable behaviors, and these classical methods fail.

For instance, in the field of system control, the usual methods in the industry rely on the construction of a fine mathematical model of the target system, that is later used to compute the commands to perform, given some set points. The cost and difficulty of the construction (and the tuning) of a mathematical model is high. An often used alternative is machine learning. Giving the ability to learn to a controller enables it to learn the behavior of the target system and build a model from data. However, this method shows its limits when used with complex systems. Nonlinearities in the learnt model lead to overcostly or impossible computations in the control system. Another possibility exists: directly learning the adequate commands, instead of a model that will later lead to the said commands. We then focus only on the inputs and outputs of the controlled system, without trying to decipher its internal mechanisms.

Another difficulty is scalability. While various control methods exist, they (almost) all fail to scale when a large number of inputs and outputs are involved. Most advanced solutions rely on the distribution of the control. Instead of letting a central controller handle all the inputs, each input is controlled by one local controller, and all controllers try to cooperate to control the whole system.

Multi-Agent Systems (MASs), composed of autonomous entities, are naturally distributed. They can be very useful to the problem of the control of complex systems, for instance with multi-objective optimization (Khamis and Gomaa, 2014). Moreover, they bring innovative design methods. In particular, Adaptive Multi-Agent Systems (AMASs) are designed to be able to self-adapt at runtime to any unexpected event. Instead of wasting time trying to cope with any possible event during the design phase, we let the system deal with the unexpected at runtime. Driven by cooperation principles, agents self-organize locally to produce and maintain the desired global function.

* Corresponding author.
E-mail addresses: boes@irit.fr (J. Boes), migeon@irit.fr (F. Migeon).

http://dx.doi.org/10.1016/j.jss.2017.08.038

This paper presents experimental results obtained with an AMAS designed to control complex systems, and applied to the calibration of real heat engines. This system is fully described in English for the first time in this paper. Able to learn and control simultaneously, it provides a generic and robust solution to the problem of control. It is a good example of the ability of AMASs to be efficient in real life conditions.

Section 2 gives a quick background on control. Section 3 introduces our approach and Section 4 presents our system. Results, obtained in simulated as well as in real conditions, are showed in Section 5. Section 6 concludes with our perspectives.

2. Related works

Our work is at the crossroad of the fields of complex systems, control, and machine learning. It is inspired by the ideas of Edgar Morin on complexity (Morin, 2008), which we apply here to the design of self-adaptive control systems.

2.1. Complex systems

The notion of complexity reflects the difficulty to analyze a system and to forecast its behavior. Nonlinearities, inner feedback loops, large number of inputs/outputs/inner parts, uncertainty on the measures, and unpredictable behaviors are some of the recurring features of complex systems. However, there is no common agreement on a definition. For instance, Kolmogorov defines the complexity of a string as the length of the shortest description of said string (Kolmogorov, 1998). While it is largely accepted, this measure implies that a purely random string is of maximal complexity, as it can only be described by its full enumeration. However this contradicts one of the key features of complexity: it is situated somewhere between total order and total chaos (Heylighen, 2008). Moreover, a complex system is dynamic, it is able to spontaneously change its state. It is important not to neglect this aspect during the analysis or the design of a system. Measures such as Kolmogorov complexity give too much attention to static, structural features of systems, and not enough to their dynamics. To this end, *dynamical depth* is based on the idea that the degree of complexity of a system is not given by its part and their causal relations, but by the imbrication of the different dynamics that drive its behavior (Deacon and Koutroufinis, 2014).

Furthermore, the general system theory states that the classical analytical approach can only be applied on systems whose parts are linear and share negligible interactions (Von Bertalanffy, 1968). This lets a lot of systems out of its scope, in particular complex systems. We need to follow a different approach than the reductionist top-down analysis for complex systems control as well as for complex systems design. The Adaptive Multi-Agent Systems theory is being developed in this regard.

2.2. Control

Control approaches also find their limits when faced with complexity. Artificial Intelligence (AI), and in particular machine learning, are used to overcome these limits.

The objective with AI in control is to automatically learn either the model of the target system, the tuning of the model, the calibration of the controller, or directly control laws from observations. For instance, Jesus and Barbosa (2013) uses a genetic algorithm to learn the optimal tuning of PIDs. This approach gives excellent results but needs a large number of iterations. Moreover, if the behavior of the controlled system changes over time (for instance, because of mechanical wear), the tuning must be entirely redone, it is not adaptive.

The biggest difficulty of dual control is to find the correct balance between probe actions and control actions. A way to do this is to use neural networks to learn this balance from data (Fabri and Bugeja, 2013). This approach is limited to control affine systems, i.e. systems that reacts linearly to modifications on their inputs.

The most promising approach for scaling up, i.e. for controlling a large number of inputs with many criteria on many outputs, is to distribute the control. For instance, Bull et al. (2004) and Choy et al. (2006) control road traffic junction signals on several crossroads. In these approaches, there is no central controller that handles all the traffic junctions, each crossroad is controlled by a local controller. Bull et al. (2004) uses learning classifier systems, while Choy et al. (2006) uses a combination of neural networks, genetic algorithms and fuzzy logic. They obtained very interesting results, but the difficulty to instantiate their approaches to real life problems is a severe drawback.

Our approach uses feedback loops to learn not the model of the controlled system but the control laws themselves, and distributes a controller on each controlled input. Inner feedback loops ensure an adequate balance between exploration and exploitation of the model.

2.3. Machine learning

A program learns when it is able to improve its functionality using its experience, i.e. data acquired during its execution (Mitchell, 2006). Machine learning has been heavily influence by the way we think the human mind works. The two well-known methods for machine learning are supervised learning and unsupervised learning, whether examples of the expected results are presented to the learning program or not. However, this distinction is merely technical and does not allow to highlight the fundamental differences between machine learning algorithms. We prefer the following five categories: Behaviorism, Cognitivism, Connectivism, Evolutionism, and Constructivism.

Behaviorism considers the learner as a black-box. Learning occurs when the observed behavior changes in response to the dynamics of the environment. In machine learning, the behavior is then a product of the initial state of the program and its progressive conditioning by its environment through a feedback loop. Reinforcement learning can be considered as a behviourist machine learning approach. It is notably popular in robotics (Kober et al., 2013). Its most well-known algorithm is Q-Learning (Watkins and Dayan, 1992).

On the contrary, cognitivists consider that what is important is not what the learner does but what he knows. Cognitivist machine learning algorithms classically rely on symbol manipulation, and thus on a predefined set of symbols, which is not adequate when dealing with complexity (Raghavan et al., 2016).

Connectionism considers learning at a lower level in the brain: the dynamic interconnection of neurons. In machine learning, it regroups all the artificial neural network algorithms, from back-propagation perceptrons to the more recent Kohonen maps (Astudillo and Oommen, 2014) and deep learning algorithms (Deng and Yu, 2014). They show impressive results but need a huge amount of data and computing power.

Evolutionism considers learning at the scale of a species rather than an individual. Evolutionary algorithms evolve a population of solutions towards better solutions by evaluating them, mutating them, and crossing the best individuals. These algorithms are interesting because they can tackle problems for which there is no known solution, but they are time-consuming and the fitness function can be difficult to obtain (Bongard, 2013).

Constructivism is the idea that humans have the ability to construct knowledge in their own mind through interactions with the environment. Constructivist artificial intelligence aims at designing

self-constructive systems (Thórisson, 2012). In such systems, not only the knowledge but the means to acquire it are learned. The focus is made on self-organization and bottom design methods.

Note that there are no hard boundaries between these categories. Most advanced machine learning algorithms actually take simultaneously from several of them. For instance Learning Classifier Systems stem from Behaviorism since they are reinforcement learning algorithms, but they also incorporate an evolutionary component (Urbanowicz and Moore, 2009).

The Adaptive Multi-Agent Systems approach is constructivist: it focuses on self-organization and shares the same long term goal of designing a fully self-constructed artificial intelligence. It also has a link with connectionism with the idea that a complex task can be achieved by a set of several simple entities.

3. Approach

Top-down classical methods have severe shortcomings when it comes to complexity: scale, integration, and flexibility (Thórisson, 2012). This section presents the Adaptive Multi-Agent Systems (AMASs) theory, that aims at overcoming these limitations thanks to the natural modularity of MASs and the cooperative self-organization of agents.

3.1. Adaptive multi-Agent systems

Wooldridge defines an agent as follows: *An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives.* Wooldridge (2009) "Autonomous action" means an agent takes its own decision on what to do and when to do it. It indefinitely follows a lifecycle of perception, decision and action without any external control.

A system composed of several agents in interaction in the same environment is called a Multi-Agent System (MAS) (Ferber, 1999). Knowledge, computation, and control are distributed among the agents of a MAS. Such systems are based on collective problem solving, the idea that local behaviors within a group can ensure the achievement of a given global task. Multi-agent systems provides interesting features when dealing with complexity, such as scalability, robustness and adaptivity (Ren and Cao, 2013).

The function of a MAS is dependent on its organization (the agents, their relations, their behavior). A change in the organization of the MAS is a change of its global function. When agents decide themselves to dynamically change their behavior or their relations, the system is self-organizing. Di Marzo Serugendo et al. define self-organization as the process with which a system changes its structure without any external control to respond to changes in its operating conditions and its environment (Di Marzo Serugendo et al., 2011). It is very natural and powerful for a MAS to perform learning and self-adaptation this way.

The Adaptive Multi-Agent Systems approach aims at facilitating the design of multi-agent systems for solving complex problems by designing simple agents that self-organize to generate a complex global function (Georgé et al., 2011). In this approach, the process of self-organization is driven by cooperation principles. Local decisions from each agent may provoke local changes that in turn lead to changes in the global function of the system.

This approach is based on the theorem of functional adequacy (Georgé et al., 2011). Applied to MASs, one of the consequences of this theorem is the assurance that the global function of a system is adequate if all agents maintain interactions with their environment that are favorable to themselves and to their environment (they are said to be in a *cooperative state*). Then, the challenge is to find the behavior for each agent that enables each of them to remain in a cooperative state despite changes in their environment.

To this end, each agent has two sets of rules. *Nominal* rules enable an agent to achieve its function when it is already in a cooperative state. However, it is highly probable that the agent eventually finds itself unable to achieve its function, due to changes in its environment, or to a simple lack of knowledge. Such cases are called *Non-Cooperative Situations* (NCSs) and are probable cause of failure for the global system to achieve its task. There are seven types of NCSs:

- Incomprehension: the agent is not able to extract information from the perceived signal.
- Ambiguity: the agent can interpret the perceived signal in several different manners.
- Incompetence: the agent is not able to decide anything based on its current knowledge and skills.
- Unproductiveness: the decision of an agent is to do nothing.
- Concurrence: the agent thinks its action will have the same effects as the action of another agent.
- Conflict: the agent thinks its action is discordant regarding the action of another agent.
- Uselessness: the agent thinks that its action will have no consequences on its environment.

When a NCS occurs, the involved agents switch from their nominal behavior rules to their *cooperative* rules, which seek to solve the NCS by provoking changes in the MAS (in other words, by triggering self-organization). An agent has several means to solve a NCS: tuning internal parameters, reorganizing its relations with other agents, creating a new agent, or self-destructing.

In the current state of the approach, the AMAS designer has to design the cooperative behavior for each NCS. A methodology named ADELFE (French acronym for Toolkit for Developing Software with Emergent Functionalities) guides the design of AMASs (Bonjean et al., 2014). It is a bottom-up and iterative design process that encourages the designer to focus on the local function of each agent, and to forget the global function of the system. A strong focus is put on decomposing the problem instead of the solution. The resulting agents will often be following simple (yet intricated) reactive behavioral rules, and thus will seem too simple to solve anything. It is the point of our approach: dodging complexity by thinking exclusively within a local scope. If agents behave accordingly to the AMAS principles of cooperation, the emerging global function shall be adequate. Originally based on the Rational Unified Process (Kruchten, 2004), ADELFE incorparates specific steps and guidelines to help identify the entities of the problem and which ones should become agents, and find their NCSs and their cooperative behaviors. It has been used for the development of the system presented in 4.

3.2. Objectives in terms of control

Other than pushing forward the experimental verification of the AMAS approach, the main objective of this work is to design a system able to learn in real time how to put a complex system in a desired state. In our case, the controlled system may have multiple inputs and outputs (MIMO), and the desired state is described as a combination of criteria. A criterion may affect one or several inputs or outputs. There are three types of criteria:

- Constraint: a threshold to meet
- Setpoint: a target value
- Optimization: a value to minimize or to maximize

An additional requirement is that the controller must be easy to implement for real-life complex systems. In particular, this means the controller should not need a heavy tuning and should not require any predefined model. In other words, prerequisite knowledge on the controlled system has to be minimal.

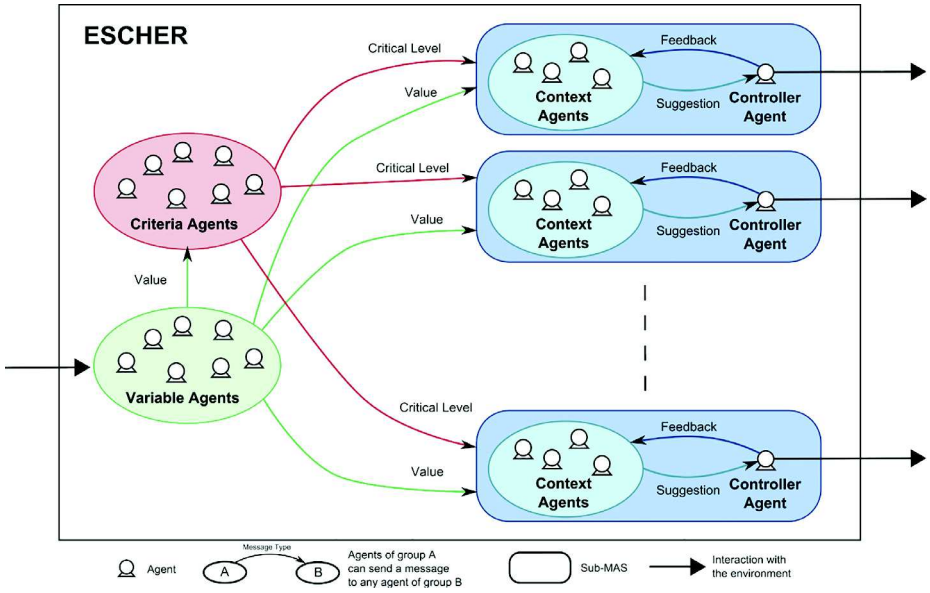

Fig. 1. A view of all the agents of ESCHER.

Moreover, the learning process has to be perpetual and in real time. It has to occur simultaneously to the control, so the controller adapts itself to changes in the controlled system (such as failures, wear, etc). Our controller sees the controlled system as a black box: it only has access to the inputs and the outputs of the black box, not to the internal processes that drives its behavior.

4. ESCHER, An adaptive MAS to learn the control of complex systems

In this section we present a multi-agent system called ESCHER, for Emergent Self-adaptive Controller for Heat Engine calibRation. Thanks to cooperative self-organization, it is able to learn in real-time the control of a system. It has been designed and tested during a project revolving around automotive thermal engines, but has been design under the assumption that nothing is known about the controlled system, except its number of inputs and outputs.

The goal is to make the controller generic enough to be used on any other systems without any modifications other than the interface. Following a "black box" approach of the control, ESCHER plays with the inputs of the controlled system, observes the effects on the outputs and infers the actions that will lead to compliance with the user-defined criteria.

4.1. System overview

The environment of ESCHER is composed of the controlled system and of the user defined criteria. This means that ESCHER observes the inputs and outputs of the controlled system, and also the control criteria defined by the users. Among the inputs of the controlled sytem, there may be some that are not controlled by ESCHER but have an impact on the controlled system. For instance the atmospheric pressure cannot be controlled but can significantly alter the output of a thermal engine. If such a sensor is available, it can be taken into account by ESCHER.

ESCHER itself is composed of four types of agents:

- Variable Agents are the eyes of the system, there is one Variable Agent for each input and output of the controlled system.
- Criterion Agents represent user-defined criteria, the desired state of the controlled system.
- Context Agents can be seen as the memory of the system, they represent a part of the state space of the environment for which the consequences of a given action are known.
- Controller Agents are the hands of the system, they interact with a set of Context Agents to find the most adequate action to perform in the environment.

Fig. 1 shows an overview of the system, with the links between the four types of agents. Note that this view is intended for the reader, agents do not have a global view of the system.

4.1.1. Context agents and controller agents

Each Controller Agent is coupled with a set of Context Agents whose memorized action is related to the effector associated to this same Controller Agent. The Controller Agent selects the next action to perform among the received suggestions and notifies the Context Agents which has sent a suggestion. There is no direct interaction between Context Agents, neither between Controller Agents. The only link between them is through the environment: the action of a Controller Agent will have an impact on the controlled system which will be perceived from other Controller Agents through Variable Agents and Criterion Agents.

A Controller Agent and its set of Context Agents can be seen as an automous MAS. Its environment would be made of Variable Agents and Criterion Agents. A Context-Controller "sub-MAS" is able to synchronize its actions with the other sub-MASs by observing the controlled system's inputs and outputs variations. A Controller Agent does its best to decrease the critical levels by performing actions on only one input, locally, without caring about how the other inputs are handled. There is no global decision process to find the adequate actions on each input at once. This feature is the key to scalability. Moreover, the distribution of control makes ESCHER modular. The addition or the removal of a new Controller Agent does not impact the others.

4.1.2. Variable agents and criteria agents

To fulfill its function, each agent besides Variable Agents, needs to know the current state of the controlled system. This is why Variable Agents send value update to every other types of agents (the relevant Criterion Agents, every Context Agent, every Controller Agent). This broadcast may seem harmful for scalability, but it is not. Indeed, agents of ESCHER are not physically distributed,

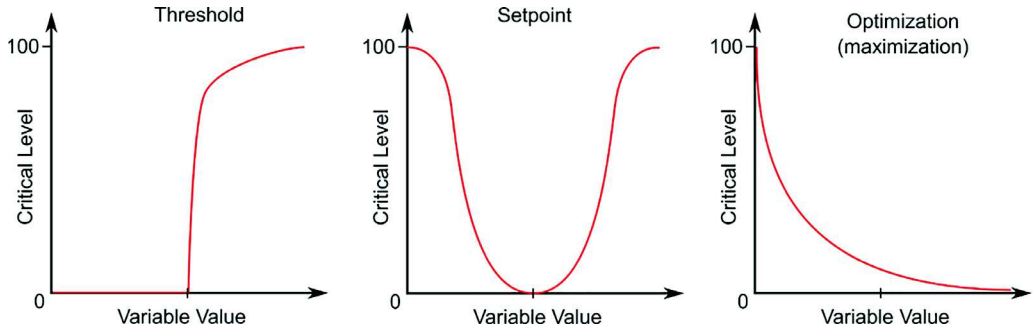

Fig. 2. Examples of criticality functions.

the cost of message sending is very low. On the contrary, the cost of reading the value of a physical sensor is high, since it involves external systems, and probably networking. Hence, it is way more efficient to have an agent per sensor, broadcasting its value to others than to give access to a sensor to every agent needing this particular value.

Criterion Agents transform the variable values into critical levels, representing the satisfaction of the criteria (i.e. a idea of how far from the desired state is the current state of the controlled system). Variable Agents and Criterion Agents give ESCHER a complete representation of its environment.

At a given moment, if every agent in the system is able to properly perform its function, then ESCHER is in a cooperative state and its global function is adequate. However, numerous cases exist where at least one of the agents is unable to execute its function. These cases are the Non-Cooperative Situations, that are presented in Section 4.3.

4.2. Function and nominal behavior of ESCHER agents

This section presents a decomposition of the activity of control in elementary tasks. Agents in charge of these tasks are detailed.

4.2.1. Observing the controlled system

The first thing we need when it comes to controlling a system with a "black box" standpoint is to be able to observe it. A specific type of agents is in charge of the perception of the controlled system: *Variable Agents*. To each input and output of the system is associated a Variable Agent. During its lifecycle, a Variable Agents perceives the value of its designated variable on the controlled system and forwards it to the other agents which may need this information. If necessary, a Variable Agents may embed a noise filtering algorithm.

4.2.2. Representing control criteria

The controller needs to have an internal representation of the objectives of the user, of the desired state for the controlled system. Giving such a representation is the function of *Criterion Agents*. There are three types of Criterion Agents:

- Threshold: the agent expresses the will to maintain a variable either above or below a user-defined threshold.
- Setpoint: the agent expresses the will to set a variable to a user-defined value.
- Optimization: the agent expresses the will to minimize or to maximize the value of a variable.

Each Criterion Agent receives updates from the relevant Variable Agents, computes a *critical level*, and sends it to other agents which may need this information. This critical level reflects the satisfaction of the criterion represented by the agent. The critical level ranges from zero (the critrerion is fully satisfied) to 100 (the criterion is far from being satisfied).

Fig. 2 shows examples of criticality functions used by Criterion Agents to compute their critical level. For instance the threshold criticality function returns zero if the threshold is met, otherwise a value up to 100. The criticality function for a setpoint returns zero only when the target value has been reached. The criticality function of an optimization criterion is asymptotic to zero. The curves of these functions can be adjusted by the user to define the relative significance of its needs.

Criterion Agents apply a transformation from the space of the controlled system variables to the space of the criteria. The critical levels decrease when their criterion is being satisfied. Hence, agents perceiving critical levels seek to decrease them. The only way to do so is to perform adequate actions on the input of the controlled system. Finding these adequate actions requires the analysis of the current state of variables and criteria to try to understand the dynamics of the system.

4.2.3. Analyzing the state of the environment

With the Variable Agents and the Criteria Agents, ESCHER has an internal distributed representation of its environment. To be able to decide which actions to perform, an analysis of this environment is needed. This is the function of *Context Agents*.

A Context Agent memorizes the effect, on each critical level, of a particular action performed on a particular effector. The agent also memorizes the state of the environment when the action is performed. This provides information about the expected consequences of a particular action if the action is performed while the environment is in a particular state.

Concretely, a Context Agent is composed of:

- an action, i.e. an offset to be performed on an input of the controlled system,
- a set of forecasts, which contains a value for each Criteria Agent, representing the expected variations of critical level,
- a set of *validity ranges*, which contains a value range for each Variable Agent, representing the state of the controlled system.

A Context Agent receives value updates from Variable Agents, and critical level updates from Criterion Agents. When the current value of each Variable Agent is inside their corresponding validity range, the Context Agent is said *valid*. This means the controlled system is in a state in which the forecasts of the agent are relevant. When a Context Agent becomes valid, it sends a notification which contains its action and its forecasts. This notification is actually an action suggestion. Let p a suggestion (1)

$$p := (a, F) \tag{1}$$

where a is an action and F is a set of critical levels forecasting functions. Thus, a function $f^i \in F$ returns the critical level of Criterion Agent i forecasted by the Context Agent if a is performed.

Such a function can be expressed as (2)

$$f^i(a) = c^i + \delta^i(a) \tag{2}$$

where c_i is the current critical level of Criterion Agent i, and δ^i is a function resulting from the learning of the Context Agent. In practice, a Context Agent sends an action suggestion together with a set of values $f^i(a)$, not a set of computable functions f^i. We only show expression (2) to explicit a part of the learning of Context Agents, which will be discussed later.

A notification is also sent when the Context Agent becomes non-valid so its suggestion is withdrawn. These suggestions and notifications are received by the *Controller Agent* in charge of the affected effector. This new type of agent is presented in the next paragraphs.

4.2.4. Performing the most adequate action

A Controller Agent is associated to each input controlled by ESCHER. The function of a Controller Agent is to perform the most adequate action on this input, i.e. the action which will provoke the greatest decrease of critical level. An action may be increasing, decreasing, or maintaining the value of the input.

Let u_t the current value of the input controlled by the Controller Agent, and a_t the action performed by the Controller Agent at its lifecycle t. The next value of the input is given by Eq. (3).

$$u_{t+1} = u_t + a_t \tag{3}$$

At each lifecycle t, the Controller Agent chooses a_t according to its internal representations, which are composed of

- $\mathcal{C}_t$, the set of critical levels, updated at lifecycle t.
- $\mathcal{P}_t$, the set of action suggestions from valid Context Agents at lifecycle t.

Among $\mathcal{P}_t$ (the received suggestions), the Controller Agent chooses the action associated with the greatest decrease of the highest critical level. If the highest critical level is not expected to vary, according to the forecasts, then the Controller Agent seeks to decrease the second highest critical level, and so on.

Hence, for each suggestion $p_t^k \in \mathcal{P}_t$, the Controller Agent looks at $f^k_{\max} \in F_t^k$, the function which returns the highest critical level (in other words, the function corresponding to the most critical Criterion Agent). This function is defined by Eq. (4).

$$f^k_{max} := f_t^k \in F_t^k, f_t^k(a_t^k) = \max_{f \in F_t^k}(f(a_t^k)) \tag{4}$$

The chosen a_t is the action from the suggestion with the lowest $f_{\max}(a)$, while being lower to the current highest critical level (Eq. (5)).

$$a_t := a^i \in \mathcal{A}_t, f^i_{max}(a^i) = \min_k(f^k_{max}(a^k)) \wedge f^i_{max}(a^i) \leq \max\ \mathcal{C}_t \tag{5}$$

where $\mathcal{A}_t$ is the set of actions contained in the suggestions from $\mathcal{P}_t$.

The Controller Agent then performs the action a_t and sends:

- an acceptance notification to the currently valid Context Agents whose action has been selected and performed,
- a rejection notification to the currently valid Context Agents whose action has not been selected,
- in case of the current action is different from the action of the previous step, a waiver notification to the Context Agents which suggested the previous action.

Of course, at any given time, a Controller Agent may not be able to make a good decision (i.e. a decision that will lead to the decrease of critical levels), because of false or incomplete information. These cases are Non-Cooperative Situations (NCSs). They occur when ESCHER has not sufficiently learned and is not fully adapted to its environment. For instance, if the condition 6 is not met, then Eq. (5) cannot be applied.

$$\exists p_t^i \in \mathcal{P}_t, \exists f^i_{max} \in F_t^i, f^i_{max}(a^i) \leq \max\ \mathcal{C}_t \tag{6}$$

The occurence of a NCS triggers a specific behavior (the cooperative behavior) of the involved agents to solve it and set the agents in a cooperative state. Solving NCSs drives the whole system towards a state of functional adequacy. NCSs and their resolution are presented in Section 4.3.

4.3. Non-Cooperative situations

This section explains how agents detect and solve NCSs. Since they provoke changes in the organization of the system, NCSs and their resolution are the key to the self-adaptativeness of AMASs. Each agent locally solves the NCSs it detects, thanks to specific actions. In ESCHER, NCSs mainly occur for Context Agents and Controller Agents. They motivate the system to self-organize, in particular by creating, modifying, or deleting Context Agents.

4.3.1. NCS 1 : Controller agent incompetence

Detection: When a Controller Agent does not receive any action suggestion, $\mathcal{P}_t = \emptyset$, hence $\mathcal{A}_t = \emptyset$. In this situation, the agent is not able to choose an adequate action using Eq. (5): it finds itself in a NCS of incompetence.

Resolution: The resolution of this NCS has two steps. First, the Controller Agent has to choose an action on its own. Its choice is based on the effects of its previous action. If the critical levels are increasing, the new action is chosen as the opposite of the previous action, otherwise the previous action is repeated (Eq. 7).

$$a_t := \begin{cases} a_{t-1} \text{ if } \max\ \mathcal{C}_t < \max\ \mathcal{C}_{t-1} \\ -a_{t-1} \text{ otherwise} \end{cases} \tag{7}$$

If $t = 0$, then the new action is randomly chosen.

If the previous action had been selected from $\mathcal{P}_{t-1}$ and is continued, the Controller Agent does not send a waiver notification to the Context Agents that had suggested it at t_{-1}, even if they are now non-valid. They may need this information to learn (see NCS 6).

Otherwise, after having determined its new action, but before performing it, the Controller Agent creates a new Context Agent. This new Context Agent is initialized with the new action, and memorizes the current value of all variables. While the highest critical level decreases, the Controller Agent continues the same action. During this time, the new Context Agent observes the variations of all critical levels to set its forecasts. Finally, when the action is abandoned, the Context Agent sets its validity ranges with the minimum and maximum observed on each variable.

4.3.2. NCS 2: Controller agent unproductiveness

Detection: When none of the received action suggestions contains forecasts of a decrease of the highest critical level (condition 6 is not met), the Controller Agent is in a NCS of unproductiveness. Its nominal decision process (select the action associated to the biggest decrease of the highest critical level) does not produce any action. There are two ways of solving this NCS, depending on the received suggestions. Let $\mathcal{A}$ the set of all possible actions for the Controller Agent, at each time step t we have $\mathcal{A}_t \subseteq \mathcal{A}$.

Resolution 1: If $\mathcal{A}_t = \mathcal{A}$, in other words if every type of actions (increment, decrement, stay) has been suggested, the Controller Agent thinks that the highest critical level can not be decreased, whatever the agent may do. Then, the agent attempts to decrease the second highest critical level (without increasing the highest critical level). If it is not possible, it will look at the third highest critical level, and so on. If there is no forecasted decrease at all,

the agent chooses the least harm: the action associated with the smallest increase of the highest critical level is chosen (Eq. (8)).

$$a_t := a^i \in \mathcal{A}_t,\ f^i_{max}(a^i) = \min_k(f^k_{max}(a^k)) \tag{8}$$

Resolution 2: The second case is when $\mathcal{A}_t \neq \emptyset \wedge \mathcal{A}_t \neq \mathcal{A}$. It means that some actions have not been suggested, they have not been tested in the current state of the environment. Since none of the received action suggestions contains forecasts of decrease of the highest critical level, they actually contain actions to avoid. Let $\mathcal{A}_c = \mathcal{A} - \mathcal{A}_t$ the set of candidate actions, i.e. actions that are not currently suggested. The Controller Agent then decides to select an action among the ones which are not suggested (which we call candidate actions). The selection of the new action is similar to the resolution of the NCS 1 but is, this time, conditioned by the presence of this action in $\mathcal{A}_c$ (9).

$$\begin{cases} a_t = a_{t-1} \text{ if } a_{t-1} \in \mathcal{A}_c \wedge \ \max \mathcal{C}_t < \ \max \mathcal{C}_{t-1} \\ a_t = -a_{t-1} \text{ if } -a_{t-1} \in \mathcal{A}_c \wedge \ \max \mathcal{C}_t \geq \ \max \mathcal{C}_{t-1} \\ a_t = \text{rand}(\mathcal{A}_c) \text{ otherwise} \end{cases} \tag{9}$$

With the same conditions than in the NCS 1, the Controller Agent may create a new Context Agent, initialized in the same manner.

4.3.3. NCS 3: Controller agent conflict

Detection: When a Controller Agent applies an action suggested by a Context Agent, it expects that the critical levels will vary in the way indicated by the forecasts. If the Controller Agent notices that it is not the case, it thinks that the action that has just been performed may be harmful, it is a conflict NCS.

Resolution: The action must be stopped. The Controller Agent abandons the action and notifies the Context Agents which had suggested it when it was selected. Moreover, if the Context Agents which were wrong are still valid, they will be temporarily ignored in future step.

4.3.4. NCS 4: Context agent conflict (false forecasts)

Detection: When the action of a valid Context Agent is being performed, said agent observes the variations of critical levels. When the action is terminated, the agent compares the observed variations with its forecasts. There is a conflict NCS if at least one of the observed variation contradicts the forecast (their direction of variation is different).

Resolution: An error in the direction of variation of a forecast is probably more than a simple mistake in the initial observation, it is not a problem of forecast adjustment. This rather indicates that the Context Agent should not have sent its suggestion, it should not have been valid. To correct this situation, the Context Agent will reduce its validity ranges, bringing closer the nearest bound to the current value of the corresponding variable.

4.3.5. NCS 5: Context agent conflict (inaccurate forecasts)

Detection: This NCS is similar to NCS 4. But this time, the observed variations are in the same direction as the forecasts, but not of the same amount. This kind of observation is sensitive to noise on the perception of variable values, hence small differences (under 5% of criticality) are ignored.

Resolution: An error in the amplitude of variation is less serious than an error in the direction of variation. The Context Agent only needs to adjust its forecast. Thus, in this case, the agent does not change its validity ranges, but rather increase or decrease the erroneous forecasts so they fit its observations.

4.3.6. NCS 6: Context agent incompetence

Detection: It happens that a Context Agent whom action is being performed becomes non-valid, but does not received any reject nor waiver notification from the Controller Agent (it is a possible outcome of NCS 1). The Context Agent is then in an incompetence NCS, this situation is not covered by its nominal behavior.

Resolution: From its standpoint, this situation means that the Controller Agent considered that its action can be kept a little longer. Hence, to keep sending what could be a good suggestion, the Context Agent extends the validity ranges that make him non-valid.

4.3.7. NCS 7: Context agent uselessness

Detection: Sometimes, after several NCS 4, some validity ranges of a Context Agent have been so shrinked that their amplitude is near zero. If the amplitude of at least one validity range falls under the threshold of minimal size, the Context Agent is in a uselessness NCS, the chances of being valid are too low. By default, the threshold is equal to one hundredth of the domain of the variable. This NCS is ignored for unbounded variables.

Resolution: A useless Context Agent can do nothing else than delete itself to solve this situation. Indeed, a Context Agent can only learn if its action is selected while valid. If the agent is never valid, it never brings information to the system and never learns. By deleting itself, the agent frees computation ressources. This NCS is not pivotal for ESCHER. The presence of useless agents does not prevent the adaptation and functional adequacy of the whole system. But this NCS avoids overages of Context Agents. To avoid that too many deletions and a loss of memory, we advise to set

4.3.8. NCS 8: Context agent unproductiveness (validity ranges)

Detection: This NCS concerns a Context Agent which has been valid, selected, then became non-valid, and observed a decrease of critical levels. This is an ideal cases, everything went fine. This is why a Context Agent in this situation considers that its action may still be relevant, even if the agent itself is now non-valid. This is an unproductiveness NCS: the nominal decision process results in doing nothing (since the agent is not valid), while there is good chances that sending an action suggestion should be a good thing to do.

Resolution: The Context Agent expands the validity ranges that make it non-valid, so it is now valid. The agent also sends an action suggestion. If the agent was wrong to send a suggestion, a NCS 4 will occur and the ranges will be shrinked. Likewise NCS 7, this situation is not crucial for the system, but enables a finer adaptation for a limited risk.

4.3.9. NCS 9: Context agent unproductiveness (suggested action)

Detection: A Context Agent whose action has been selected several times in a row considers itself in unproductiveness NCS. Indeed, the agent thinks that the ideal case would be that its action should provoke a greater decrease of critical level so it only has to be performed once. The Context Agent hence seek to adjust the amplitude of the suggested action, in a way to maximize the decrease (or minimize the increase) of critical levels.

Resolution: The adjustment of the amplitude of the action is based on the estimation of the effects of the variation of the amplitude on the variation of critical levels. The idea is to increase or decrease the amplitude of the action in a way to accelerate the decrease (or slow down the increase) of critical levels. To this end, a Context Agent which has been selected several times in a row slightly and randomly changes the amplitude of the suggested action and correlates this variation with the speed variation of critical levels. Hence, if the highest critical level is decreasing:

- quicker while the amplitude has been increased: the Context Agent keeps increasing the amplitude;
- quicker while the amplitude has been decreased: the Context Agent keeps decreasing the amplitude;
- slower while the amplitude has been increased: the Context Agent decreases the amplitude;

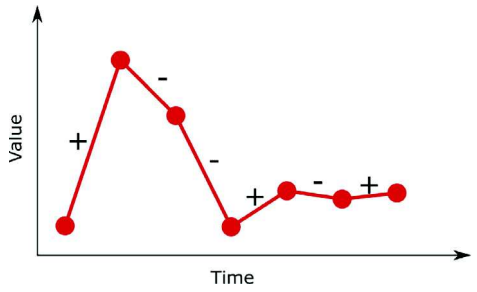

Fig. 3. Typical convergence of an adaptive value tracker.

- slower while the amplitude has been decreased: the Context Agent increases the amplitude;

The Context Agent does the exact opposite if the highest critical level is increasing, although this rarely happen since it is not frequent that an action is continued if it has provoked a rise of the highest critical level. Note that a maximal amplitude can be set in order to avoid too brutal actions.

4.3.10. Conclusion on non-Cooperative situations

This section has presented the NCSs encountered by the agents of ESCHER. In particular, the resolution of these situations provokes the creation, the deletion, and the modification of Context Agents, which are the memory of the system. In other words, NCSs provoke the memorizing, the forgetting, and the correction of knowledge based on observations of the real system: their resolution enables ESCHER to learn and self-adapt.

NCSs 1 and 2 correspond to the acquisition of new informations. They occur when ESCHER is discovering a new part of the state space of its environment. They open the system as they add new Context Agents.

NCS 3 enables ESCHER to not persist in error. It is partially solved thanks to the reorganization of the relations between a Controller Agent and some of its Context Agents. Indeed, the Controller Agent ignores some of the Context Agents if they have been wrong.

Context Agents always self-evaluate. Hence, NCSs 4 to 9 are detected if one of the parts is no longer adapted to the environment. They are solved by the adjustment of the agents (except for NCS 7 which is solved thanks to openness). Hence, ESCHER is always self-evaluating and self-adapting.

4.4. Learning and adjustment

A large part of the learning of the system relies on the tuning of Context Agents' internal parameters during the resolution of a NCS. All these parameters are tuned thanks to Adaptive Value Trackers (AVT, (Lemouzy et al., 2011)). These parameters are: the boundaries of the validity ranges, the amplitude of the suggested action, and the values of the forecasts.

An AVT converges towards a value thanks to binary feedbacks: *lower* if the real value is lower, or *greater* if the real value is greater. Both the value and the variation step of the tracker are dynamically tuned. The variation step is increased when two consecutive feedbacks are equal, and decreased otherwise. These variations follow user-defined coefficients. Fig. 3 shows an example of the variation of the value of an AVT with standard settings (two equal consecutive feedbacks double the variation step, two different consecutive feedback divide the variation step by three). A plus sign means the AVT received a *greater* feedback, a minus sign means it received a *lower* feedback.

A Context Agent transforms its observations and received notifications into feedbacks for its numerous numerous AVTs. For instance, a Context Agent in NCS 5 observing a greater variation of critical levels than what its forecast indicates will send a *greater* feedback to the corresponding AVT. The tracker then increases its value. Of course, the new value of the forecast may not be equal to the observation. But given the dynamics of the environment and the inevitable noise on real sensors, perfectly fitting to the observations is not desirable.

AVTs quickly converge toward a value, are able to stabilise, and to move again quickly toward a new further value. They match our needs, as the parameters of agents often have to change, often drastically.

4.5. Comparison with existing approaches

ESCHER has been presented as a control system because it has been designed to control. Nevertheless, learning plays a crucial role in this system. This section explores this two complementary sides of our system and their links through comparisons with the Dual Control Theory and with Learning Classifier Systems.

4.5.1. Comparison with dual control.

In the Dual Control Theory, the controlled system is partially known. The controller applies either probe actions to learn and refine its model of the controlled system, or control actions to put the controlled system in the desired state (Feldbaum, 1961). Too many probes hampers the control, but too many control actions makes a small gain. Finding the balance between probe actions and control actions requires to solve the difficult Bellman equation, which is not easily feasible in real cases.

Like dual controllers, ESCHER faces unknown systems and learns from its actions. However, it learns from all of its actions and all of its actions seek to put the controlled system in the desired state. All of its actions are probes and control actions at the same time. Moreover, unlike dual controllers, ESCHER does not need a predefined model that is later adjusted by learning.

The need to lower the critical levels (even when no agent indicates how to do it), combined to the fact that ESCHER learns from each of its actions, can be seen as an approach to solve the problem of balance between probes and control actions. The control process drives the learning process towards interesting states of the environment, while getting closer to the desired state (and thus preventing to stray away and visit uninteresting distant states).

4.5.2. Comparison with learning classifier systems.

Learning Classifier Systems (LCSs) are reinforcement learning systems (Urbanowicz and Moore, 2009).They are composed of a set of behavior rules, a pairing system which matches states of the environment with rules conditions, a selection mechanism between simultaneously triggered rules, and a genetic algorithm to tune the set of rules.

There are several similarities between a LCS and a Controller Agent coupled with its set of Context Agents. Context Agents play the same role than the pairing system (with their validity ranges) and the set of rules (each Context Agent can be seen as a behavior rule since it suggests an action under certain conditions). The Controller Agent plays a similar role than the selection mechanism, chosing an action among several suggestions from valid Context Agents.

The main difference comes from the fact that Context Agents are autonomous, they learn by themselves. On the contrary, the rules of a LCS are processed by a genetic algorithm, to withdraw the weakest and generate new and presumably more adapted rules. The fitness function of this algorithm is usually a reward signal, perceived from the environment. A great difficulty in the instanciation of a LCS is to adequately split the reward between the different rules. This difficulty does not exist in ESCHER, because

Table 1
Parameters of ESCHER and their significance.

Parameters	Significance
Number of controlled variables	Important
Number of observed variables	Important
Variables references	Important
Criticality functions	Important
Variation ranges	Optional
Maximal size of an action	Incidental
Minimal size of a validity range	Incidental
Minimal step of an AVT	Incidental
Coefficients of an AVT	Incidental

of the autonomy of Context Agents. They evaluate their adequacy themselves, and adjust themselves if needed. On certain aspects, the notion of critical levels may be assimilated to the reward signal, as it enables to evaluate the adequacy of the rules.

By self-adjusting, Context Agents suggest actions that are more and more adequate, with a more and more adequate timing, along with more and more reliable forecasts. Thus, the learning process feeds the control process.

4.6. Settings

For ESCHER to be easy to instantiate to a particular system, the number of parameters has to be as low as possible, and setting them should not require the use of elaborate calibration methods.

The only knowledge about the controlled system that ESCHER needs is quite simple

- the number of controlled variables, and their references;
- the number of observed variables, and their references.

It is possible to give the lower and higher bound for each variable. ESCHER works without this information, but it can be of use for the criticality functions. Anyway, this is basic knowledge about the controlled system, it is not an obstacle.

The only difficulty in the instanciation ESCHER is the definition of the criticality functions. Controller Agents focus on the most critical Criterion Agent. This means that the compromise between several criteria is expressed through the definition of the criticality functions. For instance, in an absurd case, if we want to maximize and minimize the same variable, ESCHER will stabilize on the value where the two criticality functions meet. This knowledge concerns not only the controlled system, but also the objectives of the user.

Finally, some other parameters are secondary. They have a very limited impact on the overall performance of the system, they do not require to be specifically set each time, their default values work fine. It is, for instance, the minimal size of validity ranges (that triggers NCS 7), the maximal size of an action (to prevent ESCHER to perform brutal actions, for safety reasons), or the internal parameters of AVTs. The strong and quick adaptiveness of the agents reduces the impact of these parameters. Table 1 shows all the parameters of ESCHER and their significance.

5. Experiments: Real-Time control of combustion engines

The first experiments presented in this section have been conducted on automatically generated synthetic black boxes. Then, experiments on a real combustion engine are shown. The implementation of ESCHER used for these experiments is a prototype written in Java 1.7 using Eclipse and a component-based multi-agent architecture generator called Make Agent Yourself (Noël, 2012). It runs on a laptop with an Intel i7 2.67 GHz CPU and 4 GB of RAM. The duration of a lifecycle of ESCHER (i.e. a lifecycle of each of its agents) depends mainly on the number of agents. It is approximately 20 ms with 10 agents, and 500 ms with 800 agents. This is something that should be improved by code optmization, but this is not the immediate concern for ESCHER. Here the goal is to show that the agents are indeed able to learn how to control several inputs of an unknown system, regarding several criteria.

5.1. Criticality functions

The function[1] used in our experiments to compute critical levels is defined over $\mathbb{R}$ as follows (Eq. (10)) :

$$f(x) = \begin{cases} 100 & \text{if } x \leq 0 \\ \gamma \frac{(x-\eta)^2}{2\eta} + \gamma(x-\eta) + \delta & \text{if } 0 < x \leq \eta \\ -\gamma \frac{(x-\eta)^2}{2(\epsilon-\eta)} + \gamma(x-\eta) + \delta & \text{if } \eta < x \leq \epsilon \\ 0 & \text{if } \epsilon < x \leq sup - \epsilon \\ -\gamma \frac{(sup-x-\eta)^2}{2(\epsilon-\eta)} + \gamma(sup-x-\eta) + \delta & \text{if } sup - \epsilon < x \leq sup - \eta \\ \gamma \frac{(sup-x-\eta)^2}{2\eta} + \gamma(sup-x-\eta) + \delta & \text{if } sup - \eta < x \leq sup \\ 100 & \text{if } sup < x \end{cases} \tag{10}$$

with

$$\gamma = -2\frac{100}{\epsilon}$$

and

$$\delta = -\gamma\frac{(\epsilon - \eta)}{2}$$

Parameters *sup*, ϵ et η are defined by the user. The curve of this function is symmetrical with respect to the center of [0; *sup*], it decreases on [0; ϵ], and increases on [$sup - \epsilon$; *sup*]. Parameter η defines the inflection point, *sup* acts as the upper bound of the function, above this value the critical level is always 100, and ϵ defines the interval [ϵ; $sup - \epsilon$] where the critical level is always zero.

In our implementation, it is possible to shift the function so the slopes happen in an arbitrary interval instead of [0; *sup*]. It is also possible to make the function asymmetrical by defining different ϵ and η for each half of the interval. For instance, by setting $\epsilon = 0$ for the left half only, we obtain a curve similar to the threshold one from Fig. 2.

It is worth remembering that each Criterion Agent has its own function set differently. It is up to an expert of the controlled system domain to set the parameter of each criticality function. This is how the balance between all criteria is expressed to ESCHER, as it always try to lower the most critical criterion before the others. However, our prototype has a simplified procedure regarding the experiments. The user does not have to directly manipulate Eq. (10), she or he only needs to select each critical variable and indicate whether the function he or she wants is a threshold, a setpoint, a minimization or a maximization. After specifying the threshold value or the setpoint value, ϵ and η are generated automatically.

5.2. Experiments on synthetic black boxes

The use of a black box generation tool (Boes et al., 2013) enabled us to test ESCHER over 50 cases of various complexity, with up to dozens of inputs and outputs. We present here two very simple cases to provide a better understanding on how ESCHER reaches a compromise between several criteria, and how it is robust to perturbations. In these experiments, a cycle corresponds to a lifecycle of each agent followed by a simulation step of the black box.

[1] Function whose formula was proposed by our colleague Sophie Jan, at the Toulouse Institute of Mathematics.

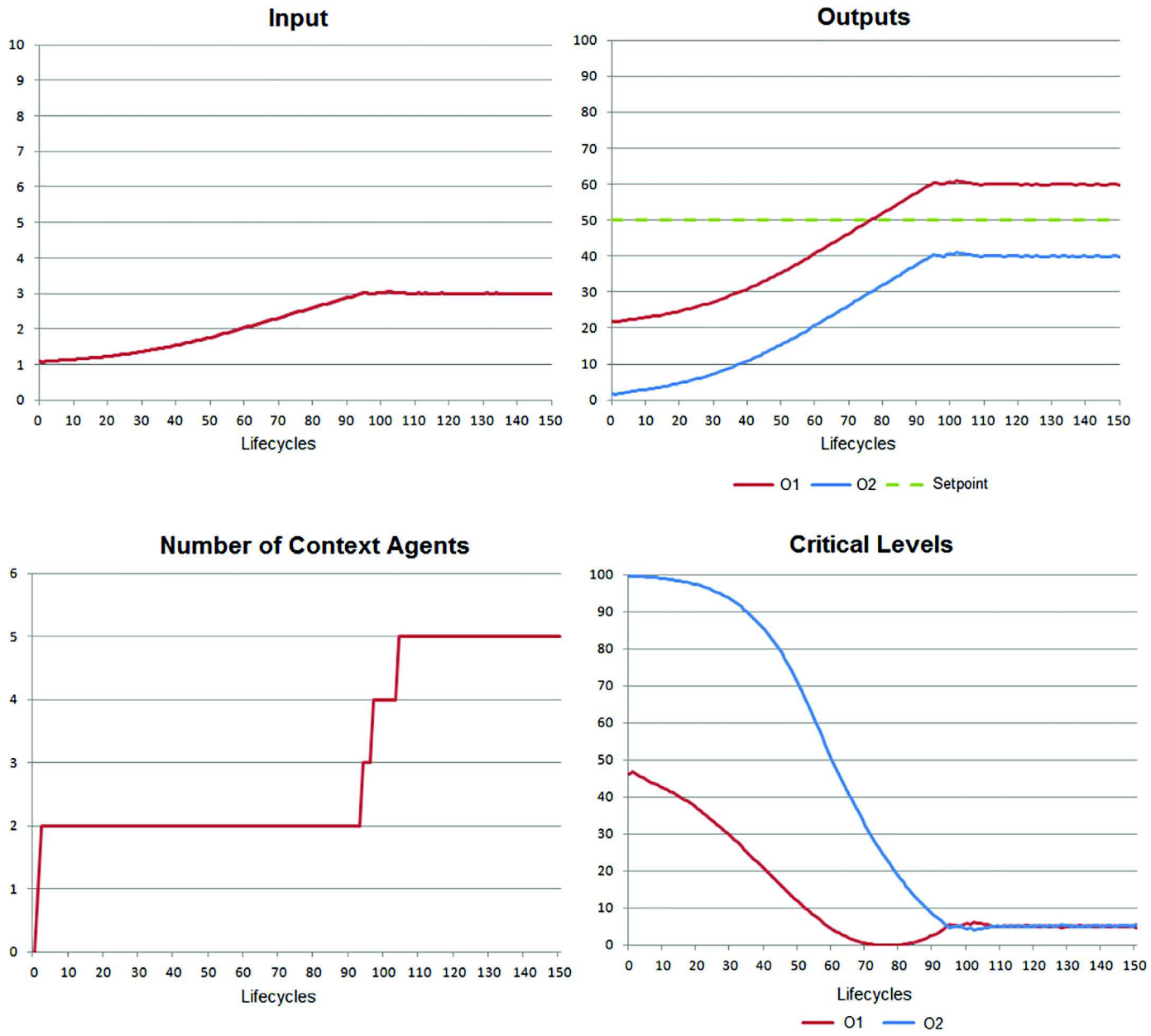

Fig. 4. Optimization of two criteria.

5.2.1. Optimizing two criteria

In this experiment, the black box has one input (I1) and two outputs (O1 and O2) varying from zero to 100. The setpoint on both outputs is 50. There are two Criterion Agents, one for each output, each with the same criticality function. Hence, both criteria have the same weight. However, this setpoint is not reachable on both output at the same time, there is no value for the input that put both output at 50. ESCHER has to find a compromise, i.e. to minimize the highest critical level.

Fig. 4 shows the variations of the input and outputs of the controlled black box, of the number of Context Agents in the system, and of the critical levels. The input is initialized to 1.1, which sets O1 to 21.8 and O2 to 1.8. O2 is further from the setpoint than O1, its critical level is therefore higher. ESCHER has no preliminary knowledge on the black box. Its action at the first step is a mistake, ESCHER slightly increase the input which provokes a small increase of both critical levels. A Context Agent for this action is created. The following step, ESCHER corrects this mistake, and find the action which push the outputs towards the setpoint. A second Context Agent is created, which action is kept until the highest critical level stops decreasing.

The critical level of O1 reaches 0 at lifecycle 76. However, the critical level of O2 is then at 26.1, and still decreasing. The action is continued, since the highest critical level is decreasing, even though the other critical level is increasing.

At lifecycle 96, critical level of O1 becomes higher than critical level of O2. In consequence, ESCHER modifies its action, and critical levels cross again. A serie of oscillations follows, during which 3 new Context Agents are created. Finally, the value of the input is stabilized, slightly oscillating around 3. O1 oscillates around 60 and O2 around 40. Both critical levels oscillate around 5. ESCHER has reached the best compromise (according to the criticality functions), since the highest critical level is the lowest possible.

This experiment shows how a Controller Agent is able to deal with an input that control several outputs with antinomic criteria. Different criticality functions would have lead to a different com-

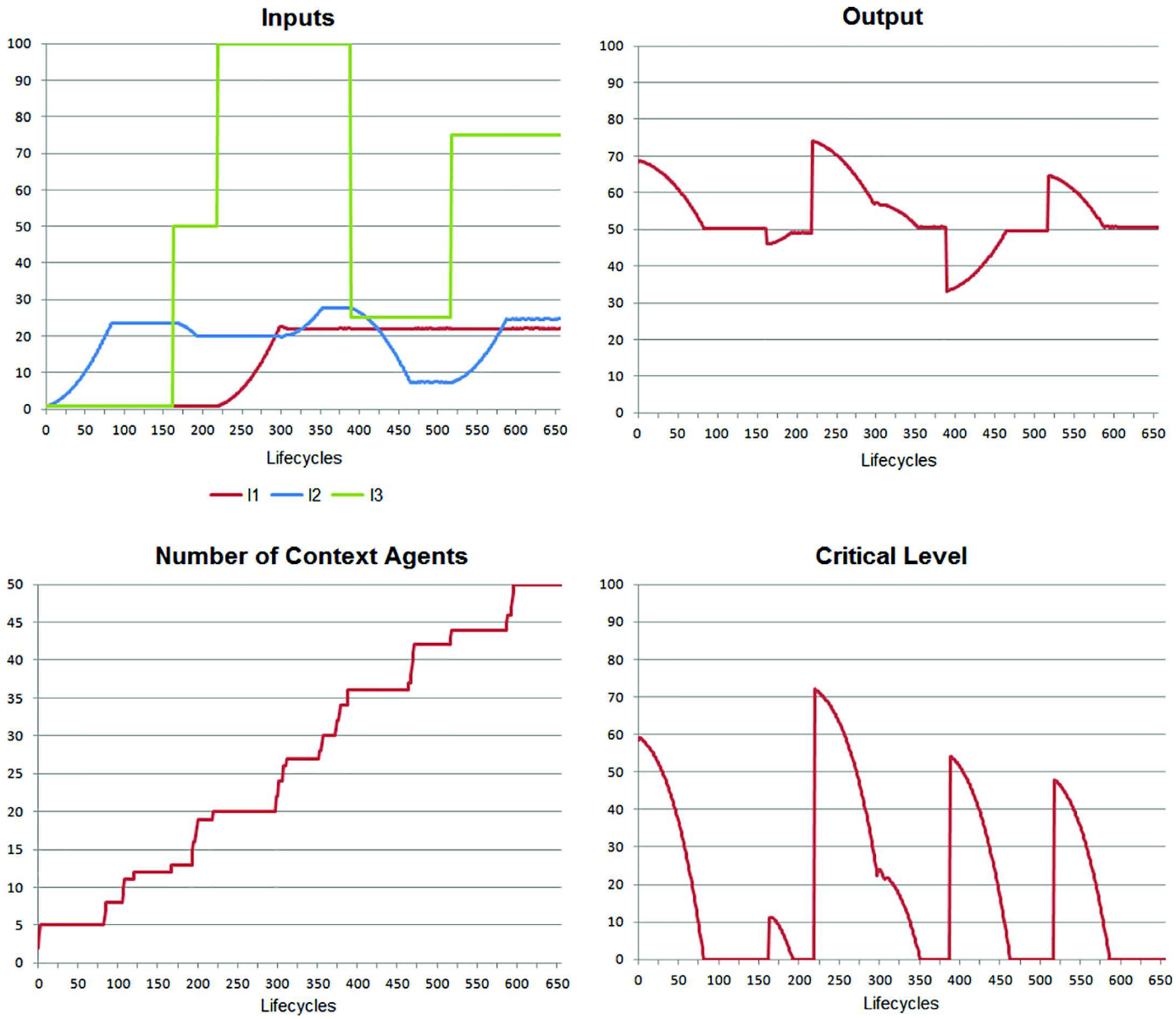

Fig. 5. Robustness to perturbations at runtime.

promise. For instance, one can prioritize one output over the other by making a criticality function always greater than the other.

5.2.2. Robustness

This experiment shows how ESCHER reacts to perturbations in its environment. Here, ESCHER controls two of the three inputs (I1 and I2) of a black box. The third input (I3) is manually controlled. These three inputs have an influence on the same output (O1), on which a setpoint criterion is applied. First, we let ESCHER make O1 meet the setpoint by modifying I1 and I2. Then, we manually change the value of I3, provoking a perturbation on O1, which abruptly go away from the setpoint. ESCHER must adapt itself to this modification by finding new values for I1 and I2.

Fig. 5 shows the variations of the input and outputs of the black box, along with the number of Context Agents and the critical level of the setpoint criterion. Inputs are initialized to 1, which sets the output to 68. The setpoint is 50. ESCHER reaches the setpoint in less than 100 lifecycles by increasing I2 only.

At lifecycle 160, I3 is manually set to 50. This makes O1 decrease, jumping out of the setpoint, resulting in a peak of critical level, which rises from 0 to 12. This is resorbed by ESCHER, which decreases I2 until the setpoint is reached again.

I3 is once again modified at lifecycle 220, from 50 to 100. This provokes a huge increase of the output, therefore a rise of critical level (from 0 to 72). Once again, ESCHER self-adapts to this perturbation. First, I1 is increased, then I2. The critical level is brought back to 0 at lifecycle 350, while new Context Agents have been created. Two other perturbations are later performed. Each time, ESCHER is able to bring back the output on the setpoint.

This experiment shows that ESCHER is able to react to perturbations on the controlled system. It self-adapts to changes to maintain an adequate control. Here, each perturbation is big enough to provoke the creation of new Context Agents.

5.3. Experiments in real conditions

The results presented in this section have been obtained during tests driven on a 125 cc monocylinder fuel engine. The engine was instrumented so ESCHER has access to temperatures, pres-

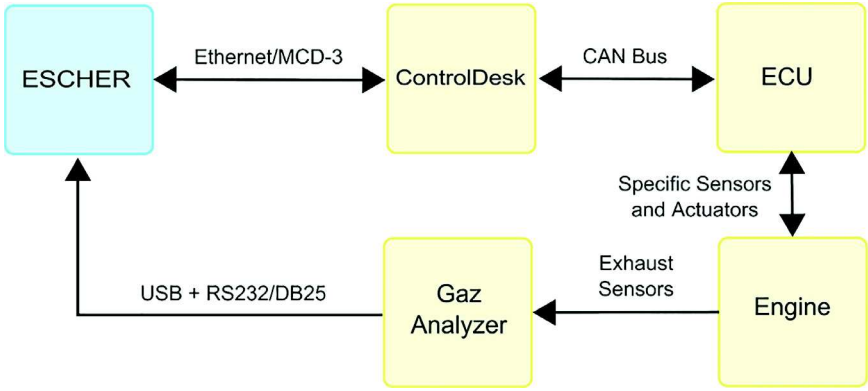

Fig. 6. Experimental Set-Up for the Tests on a Real Engine.

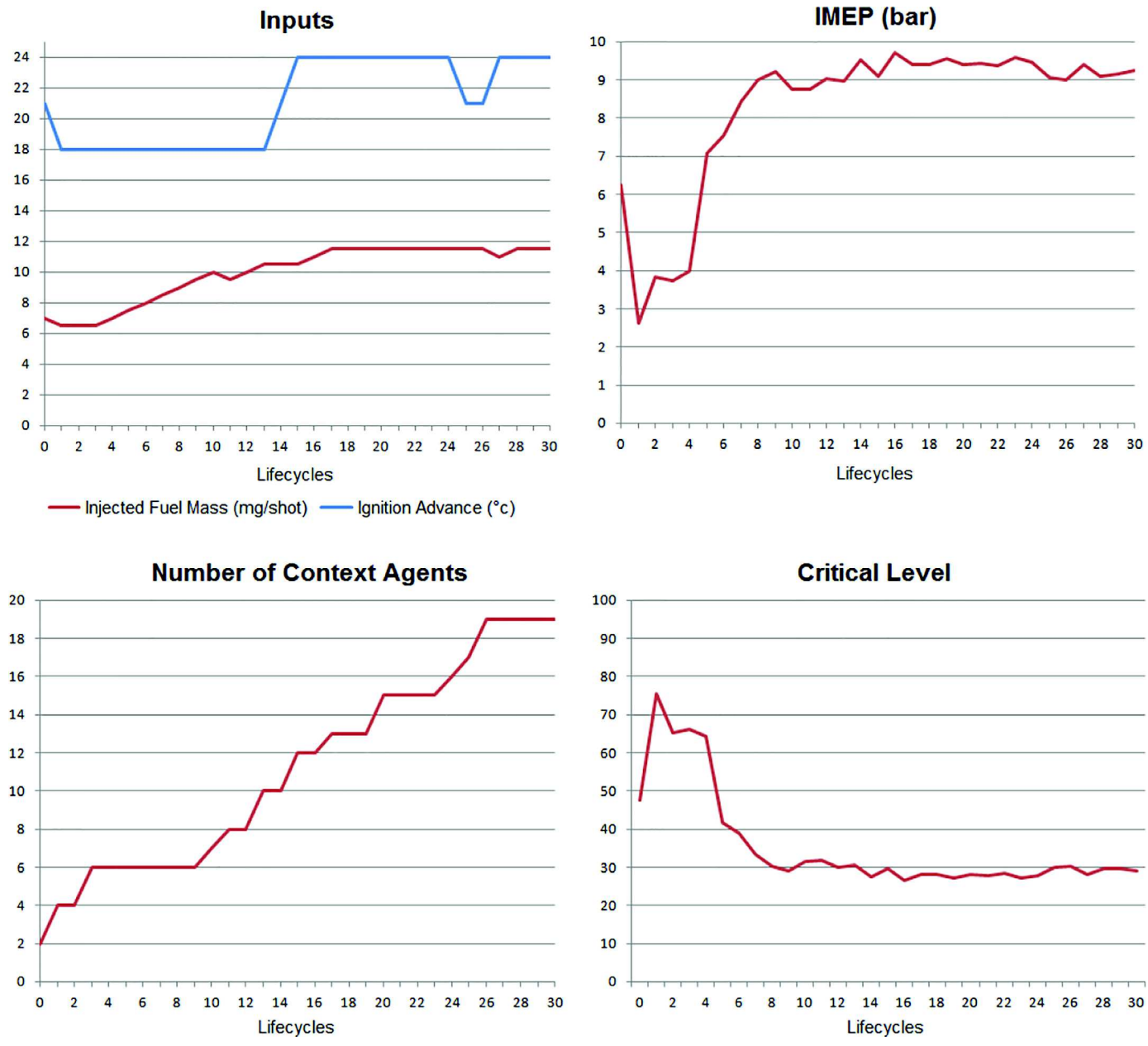

Fig. 7. IMEP optimization while controlling two parameters.

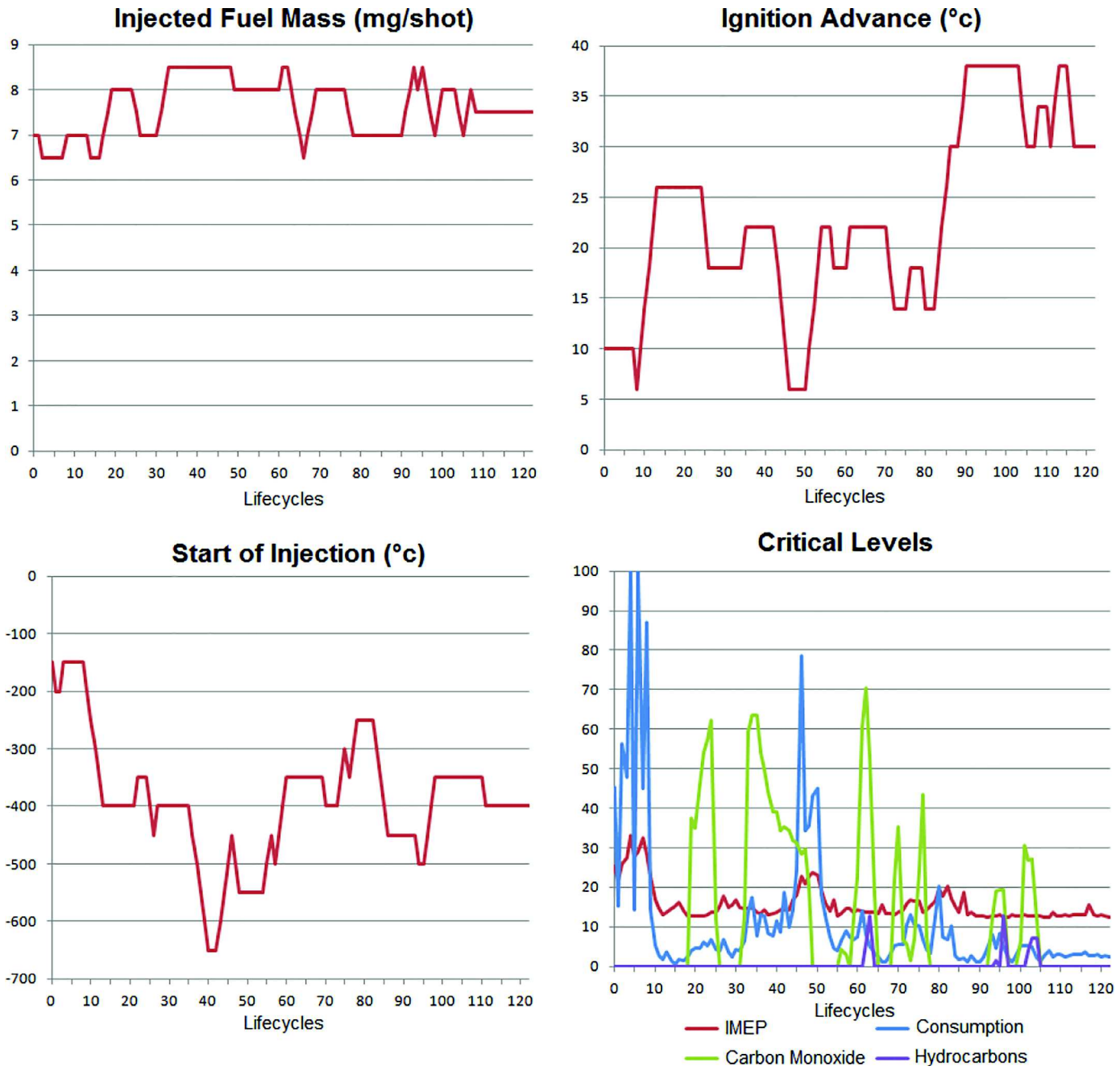

Fig. 8. Inputs and critical levels during a multi-objective optimization.

sures, and others, via the Engine Control Unit (ECU) and a gas analyzer.

The link between the engine and the ECU is assured thanks to various specific instruments. A Controller Area Network (CAN) bus enables the communication of external systems with the ECU. CAN buses are widely used in the automotive industry. A computer software called ControlDesk enables the reading on the ECU (in particular of the variables measured by the sensors), the computation of values from read variables, and the modifications of parameters (such as the ignition advance). ESCHER is connected to ControlDesk via a specific communication protocol, MCD-3 (stands for Measurement, Calibration, Diagnostics) over Ethernet, enabling our system to read and write values on the ECU. Finally, a gaz analyzer is plugged onto the engine exhaust. It measures gas concentration of various pollutants (carbon monoxide, for instance), and sends data via a serial output (RS232/DB25) interfaced with the USB port of the computer on which ESCHER runs. Fig. 6 shows this set-up. For these experiments, ESCHER had to be slowed down and wait at least 10 s between each lifecycle in order to let the engine stabilize after changing its parameters. For the second experiment, ESCHER had to wait 10 more seconds between each lifecycle for the gas analyser to provide data.

5.3.1. Torque optimization

In this experiment, the engine is put at 5000 rpm, with a load of 870 mbar in the intake manifold. ESCHER controls the total injected fuel mass and the ignition advance. The only control criterion is to maximize the indicated mean effective pressure (IMEP), which reflects the torque.

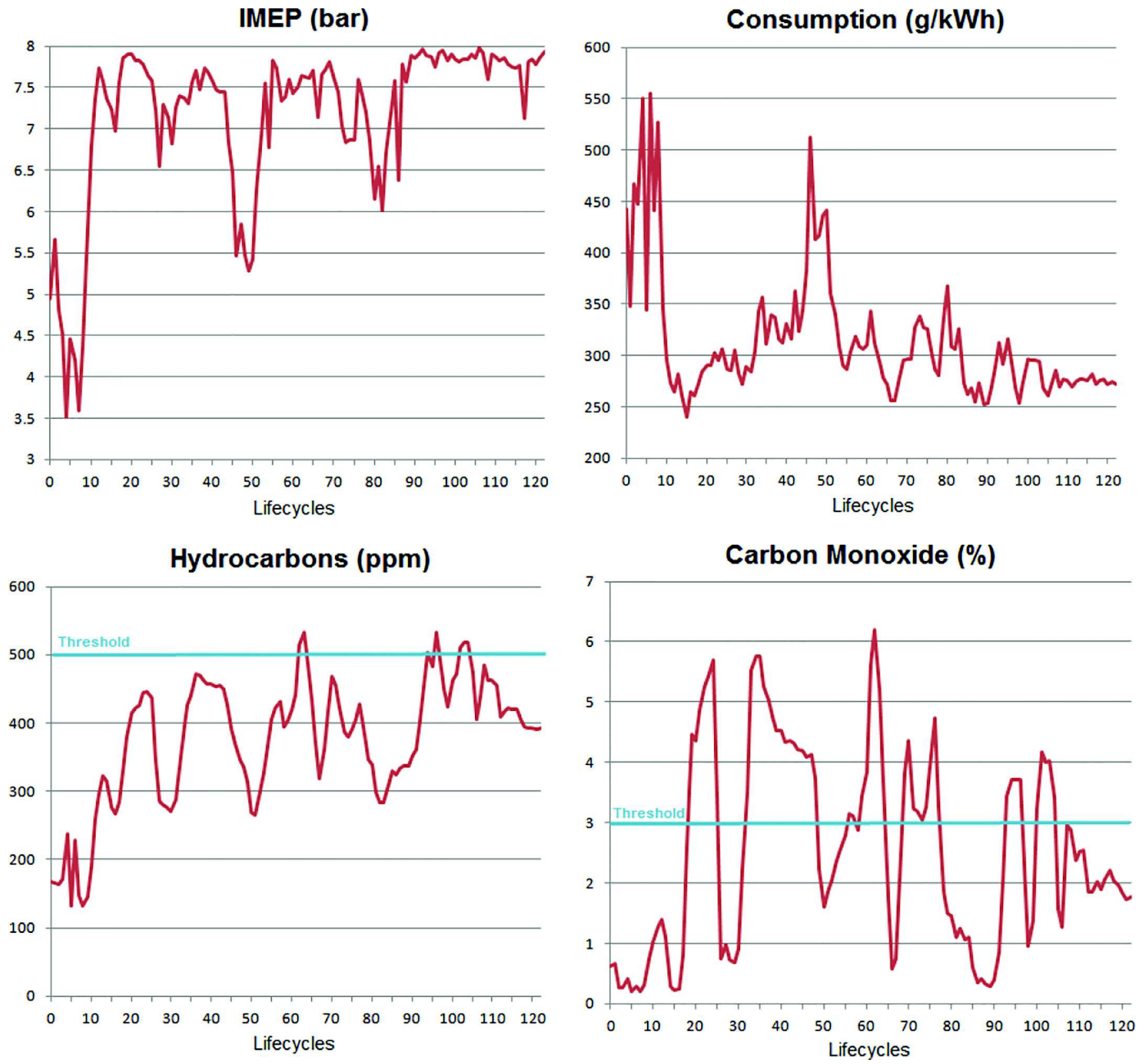

Fig. 9. Engine outputs during a multi-objective optimization.

The injected fuel mass is measured in milligrams per shot (mg/shot), and the ignition advance in crankshaft degrees (°c), i.e. the position of the piston in the cylinder when the combustion is triggered. IMEP is measured in bars. IMEP is a very unstable variable, in particular with monocylinder engines. Working at high rpm and high load, as it is the case in this experiment, reduces the instability.

The criticality function is strictly decreasing (since we want to maximize IMEP). We do not know a priori what is the maximal reachable IMEP, therefore we can not set the criticality function in a way that it returns 0 when the maximal PMI is reached. Thus, we do not expect the critical level to be zero at the end of the test, but we do expect it to be lower at the end than at the start. This is true for every criticality function used with the real engine.

Fig. 7 shows the variations of the controlled inputs, the optimized output, the number of Context Agents and the critical level. At the start, the injected fuel mass is low (7 mg/shot) regarding the current operating point. The engine is on the verge of stalling. Of course, ESCHER which does not have any knowledge about the engine, is not aware of this fact. Its first action is a mistake: ESCHER decreases both parameters, which leads to a drop of IMEP (and a rise of critical level).

ESCHER quickly finds a way to make the critical level decrease, by increasing first the injected fuel mass, then the ignition advance. IMEP finally reaches its maximum (about 9 bars), the critical level stops decreasing. ESCHER stabilizes itself at 11.50 mg/shot of injected fuel, with a 2424°c ignition advance. The decrease of these inputs at lifecycle 24 is explained by nose on the IMEP. But the system quickly corrects itself.

ESCHER managed to improve the IMEP by 3 bar in 9 lifecycles (about 90 s), reaching the maximal IMEP possible for the considered operating point. Obtaining the same result takes a skilled en-

gineer, used to this particular engine, around 20 min with usual methods.

5.3.2. Multi-Objective optimization

For this test, the engine is put in another operating point (2500 rpm, 750 mbar). ESCHER controls the injected fuel mass the ignition advance, but also the start of injection (SOI). This new parameter is the timing of the injection relatively to the position of the piston, it is measured in crankshaft degrees. There are criteria on four outputs:

- IMEP must be maximized;
- fuel consumption, measured in g/kWh, must be minimized;
- hydrocarbons (HC) emission must be under 500 ppm (parts par million);
- carbon monoxide concentration (CO) must be under 3%.

The last three criteria are contradictory with the first one. Indeed, the most efficient way to improve IMEP is to inject more fuel. However, this also increase fuel consumption and pollutants emissions. We need to adjust ignition advance and SOI to extract more power from the combustion. This is what ESCHER has to learn.

Fig. 8 shows the variations of the controlled parameters and the critical levels, while Fig. 9 shows the variations of the outputs. At the beginning, the highest critical levels is that of fuel consumption. Thus, ESCHER seeks to decrease the fuel consumption critical level in priority. The system manages to do so during the first 20 lifecycles, in particular by increasing the ignition advance from 10 to 26°C and by decreasing the SOI from -150 to −400°C, while the fuel injection oscillates between 6 and 7 mg/shot.

At lifecycle 10, IMEP maximization becomes the most critical criterion, however, its critical level is decreasing, so the same actions are continued. At lifecycle 20, the CO threshold is crossed, its critical level rises. ESCHER explores new actions to solve this problem. It continues to decrease SOI but start to decrease ignition advance. This lead to a peak of consumption and a drop of IMEP between lifecycles 45 and 50, along with small excesses of hydrocarbons. Finally, after some oscillations, ESCHER manages to put the pollutants under their respective thresholds, while maintaining a high IMEP and a low consumption.

At the end of the test, IMEP is around 8 bar (2 bar higher than the begining), while fuel consumption is around 275 g/kWh (165 g/kWh less than the initial value). Pollutants emissions are higher than their initial values, but they meet their threshold. ESCHER has successfully completed a standard engine optimization (i.e. optimizing torque and consumption while respecting pollution thresholds) without having any prior knowledge about engines. This test lasted 123 lifecycles, around 41 min (ESCHER has to wait for the gaz analyzer). This is about twice as fast than a human expert with usual methods for a similar end result.

6. Conclusion and perspectives

This article presented ESCHER, a system that illustrates the contributions of the AMAS approach to the field of control systems and calibration. This article focused on the full presentation of the system, and showed results obtained both with unrelated black-box simulations and real engines. The goal with the experiments on black-boxes was to illustrate how ESCHER works on basic cases. Experiments on the real engine show its applicability in real conditions and its robustness to noisy data. Overall, the automatic calibration performed by ESCHER is faster than methods used in the industry for a similar resutlt. However these experiments highlight a limitation of ESCHER. We had to make it wait between its lifecycles for the engine to stabilize and for the gaz analyzer to provide data. This is due to its inability to correlate actions and effects if the effects become sensible too long after the action. Further papers will present comparisons with other learning methods, detailing the advantages and limitations of each.

The AMAS approach breaks with the traditional top-down design of artificial systems. It focuses on the local behavior of agents, leaving them the task of controlling their own organization. An adequate global function emerges from this local self-organization process. We hope this is the first step towards a fully self-reconfigurable ECU.

Other AMASs have tackled the problem of learning and control with similar Context Agents, for instance with model generation (Nigon et al., 2016) and ambient robotics (Verstaevel et al., 2016). Context Agents are being generalized and standardized to become a pattern for context learning in a multi-agent system (Boes et al., 2015).

AMASs are a young technology compared to the majority of AI methods used in intelligent control, such as artificial neural networks or genetic algorithms. Our future work must focus on the formalization of the approach to enable a priori proofs of AMAS properties. This is a work in progress, which first steps have been made with Event-B (Graja et al., 2014) and continuous approximation (Stuker et al., 2014).

References

Ashby, W.R., 1956. An Introduction to Cybernetics. Chapman & Hall, London, UK.

Astudillo, C.A., Oommen, B.J., 2014. Topology-oriented self-organizing maps: a survey. Pattern Anal. Appl. 17 (2), 223–248.

Boes, J., Glize, P., Migeon, F., 2013. Mimicking complexity: automatic generation of models for the development of self-adaptive systems. In: Proceedings of International Conference on Simulation and Modeling Methodologies, Technologies and Applications. INSTICC Press, Reykjavik, Iceland, pp. 243–250.

Boes, J., Nigon, J., Verstaevel, N., Gleizes, M.-P., Migeon, F., 2015. The Self-adaptive context learning pattern: overview and proposal. In: Proceedings of International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT). Springer, Larnaca, Cyprus, pp. 91–104.

Bongard, J.C., 2013. Evolutionary robotics. Commun. ACM 56 (8), 74–83.

Bonjean, N., Mefteh, W., Gleizes, M.-P., Maurel, C., Migeon, F., 2014. Adelfe 2.0. In: Cossentino, M., Hilaire, V., Molesini, A., Seidita, V. (Eds.), Handbook on Agent-Oriented Design Processes. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 19–63.

Bull, L., Sha'Aban, J., Tomlinson, A., Addison, J.D., Heydecker, B.G., 2004. Towards distributed adaptive control for road traffic junction signals using learning classifier systems. In: Applications of Learning Classifier Systems. Springer, pp. 276–299.

Choy, M.C., Srinivasan, D., Cheu, R.L., 2006. Neural networks for continuous online learning and control. IEEE Trans. Neural Netw. 17 (6), 1511–1531.

Deacon, T., Koutroufinis, S., 2014. Complexity and dynamical depth. Information 5 (3), 404–423.

Deng, L., Yu, D., 2014. Deep learning: methods and applications. Found. Trends® in Signal Process. 7 (3–4), 197–387.

Di Marzo Serugendo, G., Gleizes, M.-P., Karageorgos, A., 2011. Self-organising Systems. In: Di Marzo Serugendo, G., Gleizes, M.-P., Karageogos, A. (Eds.), Self-Organising Software: From Natural to Artificial Adaptation. Springer Berlin Heidelberg, pp. 7–32.

Fabri, S.G., Bugeja, M.K., 2013. Kalman filter-based estimators for dual adaptive neural control: a comparative analysis of execution time and performance issues. In: Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics. INSTICC Press, Reykjavik, Iceland, pp. 169–176.

Feldbaum, A.A., 1961. Dual control theory, I-IV. Automation Remote Control 21–22. 874–880, 1–12, 109–121, 1033–1039

Ferber, J., 1999. Multi-agent systems: an introduction to distributed artificial intelligence. Addison-Wesley Reading.

Georgé, J.-P., Gleizes, M.-P., Camps, V., 2011. Cooperation. In: Di Marzo Serugendo, G., Gleizes, M.-P., Karageogos, A. (Eds.), Self-organising Software: From Natural to Artificial Adaptation. In: Natural Computing Series. Springer Berlin Heidelberg, pp. 193–226.

Graja, Z., Migeon, F., Maurel, C., Gleizes, M.-P., Laibinis, L., Regayeg, A., Kacem, A.H., 2014. A pattern based modelling for self-organizing multi-agent systems with event-b. In: Proceedings of International Conference on Agents and Artificial Intelligence. INSTICC Press, Angers, France, pp. 223–236.

Heylighen, F., 2008. Complexity and Self-organization. In: Bates, M.N., Marcia, J., Maack (Eds.), Encyclopedia of Library and Information Sciences, 3rd Edition. Taylor and Francis, pp. 1215–1224.

Jesus, I.S., Barbosa, R.S., 2013. Tuning of fuzzy fractional pd^{β}+i controllers by genetic algorithm. In: Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2013). INSTICC Press, Reykjavik, Iceland, pp. 282–287.

Khamis, M.A., Gomaa, W., 2014. Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multi--agent framework. Eng. Appl. Artif. Intell. 29, 134–151.
Kober, J., Bagnell, J.A., Peters, J., 2013. Reinforcement learning in robotics: a survey. Int. J. Robot. Res. 32 (11), 1238–1274.
Kolmogorov, A.N., 1998. On tables of random numbers. Theor. Comput. Sci. 207 (2), 387–395.
Kruchten, P., 2004. The rational Unified Process: An Introduction. Addison-Wesley Professional.
Lemouzy, S., Camps, V., Glize, P., 2011. Principles and properties of a MAS learning algorithm: a comparison with standard learning algorithms applied to implicit feedback assessment. In: Proceedings of 2011 International Conference on Web Intelligence and Intelligent Agent Technology. Springer, Lyon, France, pp. 228–235.
Mitchell, T.M., 2006. The Discipline of Machine Learning. Carnegie Mellon University, School of Computer Science, Machine Learning Department.
Morin, E., 2008. On Complexity. Hampton Press.
Nigon, J., Gleizes, M.-P., Migeon, F., 2016. Self-adaptive model generation for ambient systems. In: Proceedings of the 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016). Elsevier, pp. 675–679.
Noël, V., 2012. (Ph.D. thesis). Université de Toulouse, Toulouse, France.
Raghavan, V.V., Gudivada, V.N., Govindaraju, V., Rao, C.R., 2016. Cognitive Computing: Theory and Applications. Elsevier.
Ren, W., Cao, Y., 2013. Distributed Coordination of Multi-Agent Networks: Emergent Problems, Models, and Issues. Springer Publishing Company, Incorporated.
Stuker, S., Adreit, F., Couveignes, J.-M., Gleizes, M.-P., 2014. Continuous approximation of a discrete situated and reactive multi-agent system: contribution to agent parameterization. In: Dam, H.K., Pitt, J., Xu, Y., Governatori, G., Ito, T. (Eds.), Proceedings of the 17th International Conference on Principles and Practice of Multi-Agent Systems PRIMA 2014: December 1–5, 2014. Springer International Publishing, Gold Coast, Australia, pp. 365–380.
Thórisson, K.R., 2012. A new constructivist ai: from manual methods to self-constructive Systems. In: Theoretical Foundations of Artificial General Intelligence. Springer, pp. 145–171.
Urbanowicz, R.J., Moore, J.H., 2009. Learning classifier systems: a complete introduction, review, and roadmap. J. Artif. Evol. Appl. 2009, 1.
Verstaevel, N., Régis, C., Gleizes, M.-P., Robert, F., 2016. Principles and experimentations of self-organizing embedded agents allowing learning from demonstration in ambient robotics. Future Gen. Comput. Syst. Emerg. Ambient Ubiquitous Syst. 64, 78–87.
Von Bertalanffy, L., 1968. General System Theory: Foundations, Development, Applications. George Braziller, New York.
Watkins, C., Dayan, P., 1992. Q-learning. Mach Learn 8 (3–4), 279–292.
Wooldridge, M., 2009. An Introduction to Multiagent Systems - Second Edition. John Wiley & Sons.

Jérémy Boes, Ph.D., University of Toulouse, IRIT - Team SMAC, He is currently a postdoctoral fellow working on multi-agent systems and artificial intelligence, focusing on how to steer emergent processes towards a desirable outcome.

Frédéric Migeon, University of Toulouse, IRIT - Team SMAC, Frédéric is an Assistant Professor at the University of Toulouse. His research focuses on software engineering methods for self-adaptive multi-agent systems.